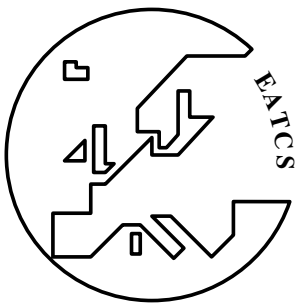
45th International Colloquium on Automata, Languages, and Programming

ICALP 2018, Prague, Czech Republic, July 9–13, 2018

Edited by

Ioannis Chatzigiannakis
Christos Kaklamanis
Dániel Marx
Donald Sannella



Editors

Ioannis Chatzigiannakis
Department of Computer, Control,
and Management Engineering
Sapienza University of Rome
ichatz@dis.uniroma1.it

Christos Kaklamanis
Department of Computer Engineering and Informatics
University of Patras and
CTI "Diophantus"
ckakl@cti.gr

Dániel Marx
Institute for Computer Science and Control
Hungarian Academy of Sciences
dmarx@cs.bme.hu

Donald Sannella
School of Informatics
University of Edinburgh
dts@inf.ed.ac.uk

ACM Classification 2012
Theory of computation

ISBN 978-3-95977-076-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-076-7.

Publication date
July, 2018

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

Digital Object Identifier: 10.4230/LIPIcs.ICALP.2018.0

ISBN 978-3-95977-076-7 **ISSN 1868-8969** **http://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

Invited Papers

Track A: Algorithms, Complexity, and Games

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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

Track C: Foundations of Networked Computation: Models, Algorithms, and Information Management

Preface

This volume contains the papers presented at ICALP 2018, the 45th edition of the International Colloquium on Automata, Languages and Programming, held in Prague, Czech Republic during July 9–13, 2018. ICALP is a series of annual conferences of the European Association for Theoretical Computer Science (EATCS), which first took place in 1972. This year, the ICALP program consisted of three tracks:

- Track A: Algorithms, Complexity, and Games,
- Track B: Logic, Semantics, Automata and Theory of Programming,
- Track C: Foundations of Networked Computation: Models, Algorithms, and Information Management.

In response to the call for papers, a total 502 submissions were received: 346 for track A, 96 for track B, and 60 for track C. Each submission was assigned to at least three Program Committee members, aided by many subreviewers. Out of these, the committee decided to accept 147 papers for inclusion in the scientific program: 98 papers for Track A, 30 for Track B, and 19 for Track C. The selection was made by the Program Committees based on originality, quality, and relevance to theoretical computer science. The quality of the manuscripts was very high, and many deserving papers could not be selected.

This year ICALP also solicitated brief announcements of work in progress with substantial interest for the community. In total 14 brief announcements were accepted for publication: 10 for Track A and 4 for Track C. The selection of the brief announcements was made by the Program Committees.

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

The best paper awards were given to the following papers:

- Track A: Heng Guo and Mark Jerrum. "A polynomial-time approximation algorithm for all-terminal network reliability".
- Track B: Dirk Nowotka and Aleksi Saarela. "An optimal bound on the solution sets of one-variable word equations and its consequences".
- Track C: Dariusz Kowalski and Miguel A. Mosteiro. "Polynomial Counting in Anonymous Dynamic Networks with Applications to Anonymous Dynamic Algebraic Computations".

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

- Track A: Shashwat Garg. "Quasi-PTAS for Scheduling with Precedences using LP Hierarchies".
- Track B: Sarah Winter. "Uniformization problems for synchronizations of automatic relations on words".

Apart from the contributed talks and the brief announcements, ICALP 2018 included invited presentations by Jaroslav Nešetřil, Alexander Schwarzmann, Sam Staton and Ryan Williams. This volume of the proceedings contains all contributed papers and brief announcements presented at the conference together with the papers and abstracts of the invited speakers.

The program of ICALP 2018 also included presentation of the EATCS Award 2018 to Noam Nisan, the Gödel Prize 2018 to Oded Regev, the Presburger Award 2018 to Aleksander

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Mądry, and the EATCS Distinguished Dissertation Award to Bas Ketsman, Ilya Razenshteyn and Aviad Rubinstein.

The program also included a memorial session for Maurice Nivat, the founder of ICALP and EATCS, who passed away in September 2017.

Six satellite events of ICALP were held on 9 July, 2018:

- Modern Online Algorithms (MOLI)
- Game Solving: Theory and Practice
- Parameterized Approximation Algorithms Workshop (PAAW)
- Infinity
- Algorithmic Aspects of Temporal Graphs
- Constrained Recognition Problems

The Summer School on Algorithms and Lower Bounds was organized immediately before ICALP during 6-9 July, 2018, with a follow-up workshop on Monday afternoon. The workshop was a satellite ICALP workshop devoted to presentations by selected participants of the school.

The Summer School on Discrete Mathematics was organized after the conference during 16-20 July, 2018. The event was organized by the Institute of Mathematics of the Czech Academy of Sciences and the Computer Science Institute of Charles University.

We wish to thank all authors who submitted extended abstracts for consideration, the Program Committees for their scholarly effort, and all referees who assisted the Program Committees in the evaluation process. We are also grateful to Anna Kotěšovcová from CONFORG and to Jiří Sgall, Andreas Emil Feldmann, Tomáš Masařík, Michal Opler, Jiří Fiala and Jan Musílek and all the support staff of the Organizing Committee from Charles University for organizing ICALP 2018.

We are grateful for generous support from AVAST and RSJ companies which included both travel grants for young women researchers and students and a direct support of the conference. We thank the School of Computer Science (Charles University, Faculty of Mathematics and Physics) and Center of Excellence - Institute for Theoretical Computer Science (project P202/12/G061 of Czech Science Foundation) for their support.

We would like to thank Jiří Sgall for his continuous support and Paul Spirakis, the president of EATCS, for his generous advice on the organization of the conference.

July 2018

Ioannis Chatzigiannakis
Christos Kaklamanis
Dániel Marx
Donald Sannella

Organization

Program Committee

Track A

Dániel Marx	Hungarian Academy of Sciences, Chair
Alexandr Andoni	Columbia University, USA
Nikhil Bansal	Eindhoven University of Technology, Netherlands
Markus Bläser	Saarland University, Germany
Glencora Borradaile	Oregon State University, USA
Sergio Cabello	University of Ljubljana, Slovenia
Joseph Cheriyan	University of Waterloo, Canada
Leah Epstein	University of Haifa, Israel
Samuel Fiorini	Université libre de Bruxelles, Belgium
Craig Gentry	IBM Research, USA
Kasper Green Larsen	Aarhus University, Danmark
Giuseppe F. Italiano	Università di Roma "Tor Vergata", Italy
Bart M.P. Jansen	Eindhoven University of Technology, Netherlands
Petteri Kaski	Aalto University, Finland
Michal Koucký	Charles University, Czech Republic
Elias Koutsoupias	Oxford, UK
Robert Krauthgamer	Weizmann Institute, Israel
Stephan Kreutzer	TU Berlin, Germany
Troy Lee	Nanyang Technological University, Singapore
Moshe Lewenstein	Bar-Ilan University, Israel
Monaldo Mastrolilli	IDSIA, Switzerland
Ankur Moitra	MIT, USA
Seffi Naor	Technion, Israel
Seth Pettie	University of Michigan, USA
Michał Pilipczuk	University of Warsaw, Poland
Alon Rosen	Herzliya Interdisciplinary Center, Israel
Günter Rote	Freie Universität Berlin, Germany
Barna Saha	University of Massachusetts Amherst, USA
Anastasios Sidiropoulos	University of Illinois at Chicago, USA
Daniel Štefankovič	University of Rochester, USA
Maxim Sviridenko	Yahoo Research, USA
Virginia Vassilevska Williams	MIT, USA
Gerhard Woeginger	RWTH Aachen, Germany
Ronald de Wolf	CWI and University of Amsterdam, Netherlands
Stanislav Živný	Oxford, UK

Track B

Donald Sannella	Univ of Edinburgh, UK, Chair
Nathalie Bertrand	IRISA/INRIA Rennes, France
Mikołaj Bojańczyk	Warsaw University, Poland
Udi Boker	Interdisciplinary Center Herzliya, Israel

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Yuxin Deng	East China Normal University, China
Floris Geerts	Univ Antwerp, Belgium
Dan Ghica	Univ Birmingham, UK
Alexey Gotsman	IMDEA, Spain
Jan Hoffmann	CMU, USA
Naoki Kobayashi	Univ Tokyo, Japan
Martin Lange	Univ Kassel, Germany
Dirk Pattinson	Australian National Univ, Australia
Femke van Raamsdonk	VU Amsterdam, Netherlands
Jean-François Raskin	Univ libre de Bruxelles, Belgium
Vladimiro Sassone	Univ Southampton, UK
Thomas Schwentick	TU Dortmund, Germany
Alex Simpson	Univ Ljubljana, Slovenia
Jiří Srba	Aalborg Univ, Denmark
Mirco Tribastone	IMT Lucca, Italy
Tomáš Vojnar	Brno Univ of Technology, Czech Republic
Igor Walukiewicz	CNRS and Univ Bordeaux, France
Scott Weinstein	Univ Pennsylvania, USA

Track C

Christos Kaklamanis	CTI “Diophantus” and University of Patras, Greece, Chair
Susanne Albers	TU Munich, Germany
Luca Becchetti	Sapienza University of Rome, Italy
Ioannis Caragiannis	University of Patras, Greece
Andrea Clementi	University of Rome “Tor Vergata”, Italy
Michele Flammini	Gran Sasso Sci Inst and Univ of L’Aquila, Italy
Pierre Fraigniaud	CNRS and Université Paris Diderot, France
Aristides Gionis	Aalto University, Finland
Sudipto Guha	University of Pennsylvania, USA
Tomasz Jurdzinski	University of Wroclaw, Poland
Evangelos Kranakis	Carleton University, Canada
Danny Krizanc	Wesleyan University, USA
Katrina Ligett	California Institute of Technology, USA and Hebrew University, Israel
Marios Mavronicolas	University of Cyprus, Cyprus
Kobbi Nissim	Georgetown University, USA
Marina Papatriantafilou	Chalmers University of Technology, Sweden
Andrzej Pelc	Université du Québec en Outaouais, Canada
David Peleg	Weizmann Institute of Science, Israel
Geppino Pucci	University of Padova, Italy
Christian Scheideler	Paderborn University, Germany
Roger Wattenhofer	ETH Zurich, Switzerland

Organizing Committee

Jiří Sgall	Charles University, Czech Republic, Conference Chair
Anna Kotěšovcová	CONFORG, Czech Republic
Andreas Emil Feldmann	Charles University, Czech Republic
Tomáš Masařík	Charles University, Czech Republic
Michal Opler	Charles University, Czech Republic
Jiří Fiala	Charles University, Czech Republic
Jan Musílek	Charles University, Czech Republic

Financial Sponsors

Additional Reviewers

Anders Aamand
Mohammad Ali Abam
Amir Abboud
Rupam Acharyya
Anil Ada
Raghavendra Addanki
Peyman Afshani
Divesh Aggarwal
Manindra Agrawal
Saba Ahmadi
S. Akshay
Xavier Allamigeon
Shaull Almagor
Josh Alman
Noga Alon
Helmut Alt
Joel Alwen
Andris Ambainis
Nima Anari
Leonardo Aniello
Simon Apers
Itai Arad
Myrto Arapinis
Srinivasan Arunachalam
Kazuyuki Asada
Gilad Asharov
Andrei Asinowski
Cigdem Aslay
Sepehr Assadi
Chen Attias
Chen Avin
Yossi Azar
Haris Aziz
Yakov Babichenko
Arturs Backurs
Saikrishna Badrinarayanan
Eric Balkanski
Marshall Ball
Alkida Balliu
János Balogh
Stephanie Balzer
Jørgen Bang-Jensen
Bahareh Banyassady
Amotz Bar-Noy
Ran Ben Basat
Felix Baschenis
Mohammadhossein Bateni
Tugkan Batu
Paul Beame
Djamal Belazzougui
Paul Bell
Alexander Belov
Omri Ben-Eliezer
Souha Ben-Rayana
Nikola Benes
Iddo Bentov
Suman Kalyan Bera
Sebastian Berndt
Aaron Bernstein
Ivona Bezakova
Aditya Bhaskara
Sayan Bhattacharya
Marcin Bienkowski
Philip Bille
Davide Bilò
Vittorio Bilò
Ahmad Biniaz
Nir Bitansky
Henrik Björklund
Jaroslaw Blasiok
Ivan Bliznets
Achim Blumensath
Hans L. Bodlaender
Greg Bodwin
Magnus Bordewich
Ralph Bottesch
Patricia Bouyer
Zvika Brakerski
Cornelius Brand
Vladimir Braverman
Thomas Brihaye
Karl Bringmann
Sabine Broda
Gerth Stølting Brodal
Joshua Brody
Vaclav Brozek
Kevin Buchin
Boris Bukh
Sam Buss
Jaroslaw Byrka
Karthik C. S.
Yang Cai
Clément Canonne
Arnaud Carayol
Clément Carbonnel
Timothy Carpenter
Matteo Ceccarello
Keren Censor-Hillel
Amit Chakrabarti
Deeparnab Chakrabarty
Diptarka Chakraborty
Sourav Chakraborty
Parinya Chalermsook
Jeremie Chalopin
T-H. Hubert Chan
Timothy M. Chan
Karthekeyan Chandrasekaran
Hsien-Chih Chang
Yi-Jun Chang
Melissa Chase
Arkadev Chattopadhyay
Shiri Chechik
Jiecao Chen
Sitan Chen
Xi Chen
Yijia Chen
Yu-Fang Chen
Siu-Wing Cheng
Victor Chepoi
Dmitry Chistikov
Rajesh Chitnis
Eden Chlamtac
Keerti Choudhary
George Christodoulou
Lorenzo Clemente
Jonas Cleve
Raphaël Clifford
Adrien Le Coënt
Avi Cohen
Edith Cohen
Ilan Cohen
Vincent Cohen-Addad
Thomas Colcombet
Denis Cornaz
Ágnes Cseh

Radu Curticapean
Dana Dachman-Soled
Anupam Das
Anindya De
Stéphane Demri
David Doty
Adrian Dumitrescu
Rüdiger Ehlers
David Eisenstat
Jörg Endrullis
Guy Even
Yaron Fairstein
Bill Fefferman
Andreas Emil Feldmann
Henning Fernau
Emmanuel Filiot
Francesca Fiorenzi
Tamás Fleiner
Fedor Fomin
Tobias Friedrich
Vincent Froese
Benjamin Fuller
Andreas Galanis
Pierre Ganty
Olivier Gauwin
Samir Genaim
Shayan Oveis Gharan
Alex Gittens
Leslie Ann Goldberg
Isaac Goldstein
Alexander Golovnev
Thorsten Götte
Fabrizio Grandoni
Alex Bredariol Grilo
Roberto Grossi
Heng Guo
Anupam Gupta
Tom Gur
Torben Hagerup
Samuel Haney
Nicolas Hanusse
Tobias Harks
Tim Hartmann
Jacob Hendricks
Danny Hermelin
Denis Hirschfeldt
Dorit Hochbaum
Michael Hoffmann
Mathieu Hoyrup
Dawei Huang

Łukasz Czajka
Hadassa Daltrophe
Laure Daviaud
Éric Colin de Verdière
Michael Dinitz
Ran Duan
Romaric Dunignau
Eduard Eiben
Khaled Elbassioni
Matthias Englert
Esther Ezra
Jittat Fakcharoenphol
Uriel Feige
Michael Feldmann
Diego Figueira
Aris Filos-Ratsikas
Johannes Fischer
Krzysztof Fleszar
Casper Benjamin Freksen
Alan Frieze
Radoslav Fulek
Peter Gacs
Nicolas Gama
Sumegha Garg
Pawel Gawrychowski
Georgios Georgiadis
Panos Giannopoulos
Vasilis Gkatzelis
Paul Goldberg
Stefan Göller
Michael T. Goodrich
Lee-Ad Gottlieb
Roland Grappe
Martin Grohe
Luciano Gualà
Jiong Guo
Manoj Gupta
Venkatesan Guruswami
Mohammadtaghi Hajiaghayi
Kristoffer Arnsfelt Hansen
Sariel Har-Peled
David Harris
Carmit Hazay
Frédéric Herbreteau
John Hershberger
Petr Hlineny
Martin Hoefer
Stepan Holub
Justin Hsu
Zhiyi Huang

Artur Czumaj
Ankush Das
Anuj Dawar
Giorgio Delzanno
Michael Gene Dobbins
Vida Dujmovic
Martin Dyer
Friedrich Eisenbrand
Lior Eldar
Hossein Esfandiari
Yuri Faenza
Carlo Fantozzi
Moran Feldman
Stefan Felsner
Nathanaël Fijalkow
Arnold Filtser
Orr Fischer
Till Fluschnik
Dominik D. Freydenberger
Zachary Friggstad
Peter Fulla
Travis Gagie
Venkata Gandikota
Paul Gastin
Gilles Geeraerts
Loukas Georgiadis
Hugo Gimbert
Shay Golan
Elazar Goldenberg
Petr Golovach
Sivakanth Gopi
Themis Gouleakis
Alexander Grigoriev
Allan Grønlund
Bruno Guillon
Krystal Guo
Sushmita Gupta
Julian Gutierrez
Nir Halman
Thomas Dueholm Hansen
Tero Harju
Prahladh Harsha
Brett Hemenway
Kieran Herley
Hiroshi Hirai
Rebecca Hoberg
Frank Hoffmann
Stefan Hougardy
Chien-Chung Huang
Pavel Hubáček

Vincent Hugot
Thore Husfeldt
Zvonko Iljazovic
Vincenzo Iovino
David Janin
Mark Jerrum
Shaofeng Jiang
Hossein Jowhari
Christos Kalaitzis
Lior Kamma
Iyad Kanj
Aikaterini Karanasiou
Zohar Karnin
Telikepalli Kavitha
Marcel Keller
Daniel Kernberger
Samir Khuller
Eunjung Kim
Philip Klein
Max Klimm
Jens Knoop
Tomasz Kociumaka
Christina Kolb
Spyros Kontogiannis
Guy Kortsarz
Jan Kratochvil
Jean Krivine
Amit Kumar
Marvin Künnemann
Thijs Laarhoven
Bundit Laekhanukit
Julien Lange
Sophie Laplante
Silvio Lattanzi
Euiwoong Lee
Ondrej Lengal
Peter Leupold
Maxwell Levit
Jason Li
Ray Li
Didier Lime
Nathan Lindzey
Christof Löding
Daniel Lokshtanov
Anand Louis
Pinyan Lu
Vladimir Lysikov
Michael Mahoney
Yury Makarychev
David Manlove

Norbert Hundeshagen
Tony Huynh
Neil Immerman
Zahra Jafargholi
David N. Jansen
Artur Jeż
Adrian Johnstone
Chiraag Juvekar
Sagar Kale
Frank Kammer
Erez Kantor
Jarkko Kari
Takashi Katoh
Ken-Ichi Kawarabayashi
Hans Kellerer
Thomas Kesselheim
Stefan Kiefer
Sándor Kisfaludi-Bak
Robert Kleinberg
Jan Willem Klop
Dušan Knop
Ioannis Kokkinis
Balagopal Komarath
Tsvi Kopelowitz
Yiannis Koutis
Klaus Kriegel
Robert Kübler
Mrinal Kumar
Denis Kuperberg
Anthony Labarre
Michael Lampis
Stefan Langerman
Kim S. Larsen
Philip Lazos
Karoliina Lehtinen
Jérôme Leroux
Reut Levi
Avivit Levy
Jerry Li
Yuanzhi Li
Andrea Lincoln
Andre Linhares
Maarten Löffler
Federico Lombardi
Shachar Lovett
Jack H Lutz
Ramanujan M. S.
Hemanta Maji
Pasquale Malacaria
Giovanni Manzini

Christoph Hunkenschröder
John Iacono
Piotr Indyk
Ragesh Jaiswal
T.S. Jayram
Łukasz Jeż
Peter Jonsson
Volker Kaibel
Gautam Kamath
Daniel Kane
Michael Kapralov
Juha Kärkkäinen
Isabella Kaufmann
Steven Kelk
Dominik Kempa
Shahbaz Khan
Daniel Kifer
Kim-Manuel Klein
Boris Klemz
Katharina Klost
Yusuke Kobayashi
Sudeshna Kolay
Christian Konrad
Swastik Kopparty
Karel Král
S Krishna
Oliver Kullmann
Ravi Kumar
Salvatore La Torre
Arnaud Labourel
Patrick Landwehr
Elmar Langetepe
Sławomir Lasota
Hung Le
Steffen Lempp
Stefano Leucci
Asaf Levin
Nathan Lhote
Jian Li
Nutan Limaye
Steven Lindell
Quanquan Liu
Markus Lohrey
Julian Loss
Hsueh-I Lu
James F. Lynch
Mohammad Mahdian
Konstantin Makarychev
Andreas Maletti
Andrea Margheri

Nicolas Markey
Tomas Masopust
Manuel Mauro
Andrew McGregor
Kurt Mehlhorn
Massimo Merro
Tom Meyerovitch
Marius Mikučionis
Carl Miller
Joseph Mitchell
Rolf H. Möhring
Cris Moore
Dana Moshkovitz
Wolfgang Mulzer
Cameron Musco
Daniel Nagaj
Guyslain Naves
Mark-Jan Nederhof
Ofer Neiman
Van Chan Ngo
Matthias Niewerth
Krzysztof Nowicki
Alexander Okhotin
Dennis Olivetti
Aurélien Ooms
Yota Otachi
Sang-Il Oum
Rasmus Pagh
Igor Pak
Gopal Pandurangan
Nikos Parotsidis
Francesco Pasquale
Lehilton L. C. Pedrosa
Richard Peng
Will Perkins
Andrea Pietracaprina
Jean-Eric Pin
Vladimir Podolskii
M. Praveen
Pavel Pudlak
Youming Qiao
Balaji Raghavachari
Govind Ramnarayan
Julian Rathke
Dror Rawitz
Vojtech Rehak
Bruce Richter
Liam Roditty
Dana Ron
Jurriaan Rot

Euripides Markou
Antonis Matakos
Richard Mayr
Moti Medina
Ruta Mehta
Julian Mestre
Theresa Migler-Vondollen
Martin Milanič
Tillmann Miltzow
Matthias Mnich
Tobias Mömke
Pat Morin
Elchanan Mossel
Kamesh Munagala
Christopher Musco
Viswanath Nagarajan
Amir Nayyeri
Jesper Nederlof
Adrian Neumann
Huy Nguyen
Filip Niksic
Marc Noy
Igor Carboni Oliveira
Feyishayo Olukoya
Sebastian Ordyniak
Joël Ouaknine
Megan Owen
Linda Pagli
Konstantinos Panagiotou
Debmalya Panigrahi
Merav Parter
Erik Paul
Chris Peikert
Guillermo Perez
Jeff Phillips
Oleg Pikhurko
Thomas Place
Ely Porat
Eric Price
Simon Puglisi
Daowen Qiu
Ajitha Rajan
Narad Rampersad
Gaurav Rattan
Jean-Florent Raymond
Daniel Reichman
Havana Rika
Heiko Röglin
Adi Rosén
Marc Roth

Francisco Martins
Bastien Maubert
Arya Mazumdar
Mohammad Syed Meesum
Manor Mendel
Pierre-étienne Meunier
Matúš Mihalák
Kevin Milans
Pranabendu Misra
Ali Mohades
Benjamin Moore
Ben Moseley
Shay Mozes
Ian Munro
Torsten Mütze
Anand Natarajan
Amir Nayyeri
Joe Neeman
Ilan Newman
Denis Nicole
Gali Noti
Timm Oertel
Rafael Oliveira
Krzysztof Onak
Mikhail Ostrovskii
Youssouf Oualhadj
Kenta Ozeki
Dominik Pajak
Anurag Pandey
Periklis Papakonstantinou
Anat Paskin-Cherniavsky
Subhabrata Paul
Pan Peng
Pablo Pérez-Lantero
Astrid Pieterse
Marcin Pilipczuk
Wojciech Plandowski
Gustavo Posta
Kirk Pruhs
Manish Purohit
Karin Quaas
Rajmohan Rajaraman
Sofya Raskhodnikova
Malin Rau
Ilya Razenshteyn
Ahmed Rezine
Andrej Risteski
Lars Rohwedder
Peter Rossmanith
Ron Rothblum

Thomas Rothvoss
Michał Różański
Aleksi Saarela
Ken Sakayori
Laura Sanita
Rahul Santhanam
Jayalal Sarma
Srinivasa Rao Satti
Nitin Saurabh
Raghuvansh Saxena
Nadja Scharf
Georg Schnitger
Roy Schwartz
Elizabeth Scott
Danny Segev
Alexander Setzer
Don Sheehy
Tong-Wook Shinn
Rodrigo Silveira
Alistair Sinclair
Alexander Skopalik
Christian Sommer
Christopher Spinrath
Vijay Sridhar
Nikhil Srivastava
Damien Stehle
Noah Stephens-Davidowitz
Martin Sulzmann
Toshio Suzuki
Navid Talebanfard
Li-Yang Tan
Justin Thaler
Dimitrios Thilikos
Simone Tini
Patrick Totzke
Elias Tsakas
Takeshi Tsukada
Yuya Uezato
Ali Vakilian
Rob van Stee
Prashant Vasudevan
László A. Végh
Nikolay Vereshchagin
Cosimo Vinci
Nils Vortmeier
Magnus Wahlström
Johannes Waldmann
Yuyi Wang
Oren Weimann
Stefan Weltge

Reuben Rowe
Polina Rozenshtein
Sushant Sachdeva
Michael Saks
Piotr Sankowski
Ramprasad Saptharishi
Kanthi Sarpatwar
Ignasi Sau
Saket Saurabh
Guillaume Scerri
Sven Schewe
Tselil Schramm
Chris Schwiegelshohn
Adam Sealfon
Ilya Sergey
Asaf Shapira
Tetsuo Shibuya
Aaron Sidford
Francesco Silvestri
Kritika Singhal
Michał Skrzypczak
Fu Song
Sophie Spirkl
Akshayaram Srinivasan
B Srivathsan
Fabian Stehn
Sebastian Stiller
Scott Summers
Ola Svensson
Ohad Talmon
Jakub Tarnawski
Sharma V. Thankachan
Dilys Thomas
Andreas Tönnis
Henry Towsner
Max Tschaikowski
Iddo Tzameret
Seeun William Umboh
Leo van Iersel
Anke van Zuylen
Yann Vaxès
Santosh Vempala
José Verschae
Sundar Vishwanathan
Satyanarayana Vusirikala
Erik Waingarten
Erik Walsberg
Justin Ward
Omri Weinstein
Anthony Widjaja Lin

Eric Rowland
Atri Rudra
S. Cenk Sahinalp
Ario Salmasi
Ocan Sankur
Thatchaphol Saranurak
Tetsuya Sato
Thomas Sauerwald
Zdenek Sawa
Michael Schapira
Stefan Schmid
Dominique Schroeder
Giacomo Scornavacca
Erel Segal-Halevi
C. Seshadhri
Micha Sharir
Igor Shinkar
Sebastian Siebertz
Ryoma Sin'Ya
Rene Sitters
Shay Solomon
Aikaterini Sotiraki
Joachim Spoerhase
Aravind Srinivasan
Tatiana Starikovskaya
Florian Steinberg
Thomas Sturm
Aarthi Sundaram
Avishay Tal
Suguru Tamaki
Yael Tauman Kalai
Johan Thapper
Francesco Tiezzi
Tigran Tonoyan
Ohad Trabelsi
Philippas Tsigas
Marc Uetz
Rohit Vaish
Erik Jan van Leeuwen
Shai Vardi
Rahul Vaze
Carmine Ventre
Aravindan Vijayaraghavan
Jan Vondrak
Nikhil Vyas
David Wajc
Di Wang
Pascal Weil
Mathias Weller
Andreas Wiese

Max Willert
Karl Wimmer
Dominik Wojtczak
James Worrell
Pei Wu
Ning Xie
Grigory Yaroslavtsev
Neal Young
Meirav Zehavi
Hanmeng Zhan
Baigong Zheng
Jack Williams
Sarah Winter
Bruno Woltzenlogel Paleo
Marcin Wrochna
Zhilin Wu
Lin F. Yang
Yitong Yin
Huacheng Yu
Marc Zeitoun
Hantao Zhang
Vassilis Zikas
Ryan Williams
Philipp Woelfel
David P. Woodruff
David Wu
Christian Wulff-Nilsen
Mu Yang
Yuichi Yoshida
Or Zamir
Georg Zetzsche
Jie Zhang
Charilaos Zisopoulos

List of Authors

Anders Aamand
BARC, University of Copenhagen, Universitetsparken 1, Copenhagen, Denmark
aa@di.ku.dk
https://orcid.org/0000-0002-0402-0514

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

Nathanael L. Ackerman
Harvard Univ

Anna Adamaszek
University of Copenhagen, Denmark
anad@di.ku.dk

Navneet Agarwal
Indian Institute of Technology Bombay
navneet@cse.iitb.ac.in

Pankaj K. Agarwal
Department of Computer Science, Duke University, Durham, NC 27708, USA
pankaj@cs.duke.edu

Alejandro Aguirre
IMDEA Software Institute, Madrid, Spain

Saeed Akhoondian Amiri
Max-Planck Institute of Informatics, Germany
samiri@mpi-inf.mpg.de

Eleni C. Akrida
Department of Computer Science, University of Liverpool, UK
Eleni.Akrida@liverpool.ac.uk
https://orcid.org/0000-0002-1126-1623

Shaull Almagor
Department of Computer Science, Oxford University, UK
shaull.almagor@cs.ox.ac.uk

Antoine Amarilli
LTCI, Télécom ParisTech, Université Paris-Saclay

Sanat Anand
Indian Institute of Technology Bombay
sanat@cse.iitb.ac.in

Megumi Ando
Computer Science Department, Brown University, Providence, RI 02912 USA
mando@cs.brown.edu

Moab Arar
Tel Aviv University, Tel Aviv, Israel

Rotem Arnon-Friedman
ETH Zürich, Switzerland
rotema@itp.phys.ethz.ch

Boris Aronov
Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY 11201, USA
boris.aronov@nyu.edu

Albert Atserias
Universitat Politècnica de Catalunya, Barcelona, atserias@cs.upc.edu

Amy Babay
Johns Hopkins University, Baltimore, MD, USA
babay@cs.jhu.edu

Miriam Backens
Department of Computer Science, University of Oxford, UK
miriam.backens@cs.ox.ac.uk

Achiya Bar-On
Department of Mathematics, Bar-Ilan University, Ramat Gan, Israel
abo1000@gmail.com

Gali Bar-On
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel
galibar@post.bgu.ac.il

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Gill Barequet
Technion - Israel Inst. of Technology, Haifa, Israel
barequet@cs.technion.ac.il

Gilles Barthe
IMDEA Software Institute, Madrid, Spain

MohammadHossein Bateni
Google Research, New York

Soheil Behnezhad
University of Maryland

Ran Ben Basat
Technion, Haifa, Israel
sran@cs.technion.ac.il

Eli Ben-Sasson
Department of Computer Science, Technion, Haifa, Israel
eli@cs.technion.ac.il
https://orcid.org/0000-0002-0708-0483

Iddo Bentov
Cornell University, Ithaca, NY, USA
iddo333@gmail.com

Ben Berger
Weizmann Institute of Science, Rehovot, Israel
ben.berger@weizmann.ac.il

Juan Jose Besa
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
jjbesavi@uci.edu
https://orcid.org/0000-0002-5676-7011

Amey Bhangale
Weizmann Institute of Science, Rehovot, Israel
amey.bhangale@weizmann.ac.il

Aditya Bhaskara
School of Computing, University of Utah, USA
bhaskara@cs.utah.edu

Arnab Bhattacharyya
Indian Institute of Science, Bangalore, India
arnabb@iisc.ac.in

Therese Biedl
School of Computer Science, University of Waterloo, Canada
biedl@uwaterloo.ca

Davide Bilò
Department of Humanities and Social Sciences, University of Sassari, Via Roma 151, 07100 Sassari (SS), Italy
davide.bilo@uniss.it
https://orcid.org/0000-0003-3169-4300

Vittorio Bilò
Department of Mathematics and Physics, University of Salento, Lecce, Italy
vittorio.bilo@unisalento.it

Ahmad Biniaz
School of Computer Science, University of Waterloo, Canada
ahmad.biniaz@gmail.com

Jeremiah Blocki
Department of Computer Science, Purdue University, West Lafayette, Indiana, USA
jblocki@purdue.edu

Avrim Blum
TTI-Chicago, Chicago, United States
avrim@ttic.edu

Achim Blumensath
Masaryk University Brno
blumens@fi.muni.cz

Thomas Bläsius
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
thomas.blaesius@hpi.de

Andrej Bogdanov
Department of Computer Science and Engineering and , Institute of Theoretical Computer Science and Communications, Chinese University of Hong Kong
andrejb@cse.cuhk.edu.hk

Prosenjit Bose
School of Computer Science, Carleton University, Ottawa, Canada
jit@scs.carleton.ca

Sébastien Bouchard
Sorbonne Universités, UPMC Univ Paris 06, CNRS, INRIA, LIP6 UMR 7606, Paris, France
sebastien.bouchard@lip6.fr

Elette Boyle
IDC Herzliya
elette.boyle@idc.ac.il

Zvika Brakerski
Weizmann Institute of Science, Rehovot, Israel
zvika.brakerski@weizmann.ac.il

Vladimir Braverman
Johns Hopkins University, Baltimore, United States
vova@cs.jhu.edu

Karl Bringmann
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
kbringma@mpi-inf.mpg.de

Jarosław Byrka
University of Wrocław, Wrocław, Poland
jby@cs.uni.wroc.pl

Mathias Bæk Tejs Knudsen
University of Copenhagen and Supwiz, Copenhagen, Denmark
mathias@tejs.dk
https://orcid.org/0000-0001-5308-9609

Karthik C. S.
Weizmann Institute of Science, Rehovot, Israel
karthik.srikanta@weizmann.ac.il

Paz Carmi
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel
carmip@cs.bgu.ac.il

Marco L. Carmosino
Department of Computer Science, University of California San Diego, La Jolla, CA, USA
marco@ntime.org

L. Elisa Celis
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Deeparnab Chakrabarty
Department of Computer Science, Dartmouth College, 9 Maynard St, Hanover, NH, USA
deeparnab@dartmouth.edu

Timothy M. Chan
Dept. of Computer Science, University of Illinois at Urbana-Champaign, USA
tmc@illinois.edu

L. Sunil Chandran
Department of Computer Science and Automation, Indian Institute of Science, India
sunil@csa.iisc.ernet.in

Steven Chaplick
Lehrstuhl für Informatik I, Universität Würzburg, Germany
steven.chaplick@uni-wuerzburg.de
https://orcid.org/0000-0003-3501-4608

Moses Charikar
Department of Computer Science, Stanford University, Stanford, CA, USA
moses@cs.stanford.edu

Shiri Chechik
Tel Aviv University, Tel Aviv, Israel

Jiehua Chen
Ben-Gurion University of the Negev, Beer Sheva, Israel
jiehua.chen2@gmail.com

Jing Chen
Department of Computer Science, Stony Brook University, Stony Brook, NY 11794, USA
jingchen@cs.stonybrook.edu

Siu-Wing Cheng
Department of Computer Science and Engineering, HKUST, Hong Kong
scheng@cse.ust.hk
https://orcid.org/0000-0002-3557-9935

Yun Kuen Cheung
Max Planck Institute for Informatics, Saarland Informatics Campus, Germany
ycheung@mpi-inf.mpg.de
https://orcid.org/0000-0002-9280-0149

Flavio Chierichetti
Dipartimento di Informatica, Sapienza University, Rome, Italy
flavio@di.uniroma1.it
https://orcid.org/0000-0001-8261-9058

Suryajith Chillara
Department of CSE, IIT Bombay, Mumbai, India
suryajith@cse.iitb.ac.in

Dmitry Chistikov
Centre for Discrete Mathematics and its Applications (DIMAP) & , Department of Computer Science, University of Warwick, UK
d.chistikov@warwick.ac.uk

George Christodoulou
Department of Computer Science, University of Liverpool, Liverpool, UK
G.Christodoulou@liverpool.ac.uk

Julia Chuzhoy
Toyota Technological Institute at Chicago, 6045 S. Kenwood Ave., Chicago, Illinois 60637, USA
cjulia@ttic.edu

Lorenzo Clemente
University of Warsaw
clementelorenzo@gmail.com
https://orcid.org/0000-0003-0578-9103

Sarel Cohen
Tel Aviv University, Tel Aviv, Israel

Riccardo Colini-Baldeschi
Core Data Science Group, Facebook Inc., 1 Rathbone Place, London, W1T 1FB, UK
rickuz@fb.com
https://orcid.org/0000-0001-5739-1178

Alex Conway
Rutgers University, New Brunswick, NJ, USA
alexander.conway@rutgers.edu

Robert Cummings
School of Computer Science, University of Waterloo, Canada
rcummings000@gmail.com

Wojciech Czerwiński
University of Warsaw, Poland
https://orcid.org/0000-0002-6169-868X

Samira Daruki
Expedia Research, USA
sdaruki@expedia.com

Shantanu Das
LIS, Aix-Marseille University, University of Toulon, CNRS, Marseille, France
shantanu.das@lif.univ-mrs.fr

Samir Datta
Chennai Mathematical Institute & UMI ReLaX, Chennai, India
sdatta@cmi.ac.in

Laure Daviaud
University of Warwick, Coventry, UK
L.Daviaud@warwick.ac.uk

Minati De
Department of Mathematics, Indian Institute of Technology Delhi, India
minati@maths.iitd.ac.in

Sina Dehghani
University of Maryland, College Park, MD 20742, USA

Holger Dell
Saarland University and Cluster of Excellence (MMCI), Saarbrücken, Germany
hdell@mmci.uni-saarland.de
https://orcid.org/0000-0001-8955-0786

Mahsa Derakhshan
University of Maryland

Dariusz Dereniowski
Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Gdańsk, Poland
deren@eti.pg.edu.pl
https://orcid.org/0000-0003-4000-4818

María Emilia Descotte
LaBRI, Université de Bordeaux

William E. Devanny
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
wdevanny@uci.edu

Ilias Diakonikolas
USC, Los Angeles, CA, USA
diakonik@usc.edu

Yoann Dieudonné
Laboratoire MIS & Université de Picardie Jules Verne, Amiens, France
yoann.dieudonne@u-picardie.fr

Michael Dinitz
Johns Hopkins University, Baltimore, MD, USA
mdinitz@cs.jhu.edu

Itai Dinur
Computer Science Department, Ben-Gurion University, Beer Sheva, Israel
dinuri@cs.bgu.ac.il

Gaëtan Douéneau-Tabot
École Normale Supérieure Paris-Saclay, Université Paris-Saclay, Cachan, France
gaetan.doueneau@ens-paris-saclay.fr

Ran Duan
Institute for Interdisciplinary Information Sciences, Tsinghua University, China
duanran@mail.tsinghua.edu.cn

Bartłomiej Dudek
Institute of Computer Science, University of Wrocław, Poland
bartlomiej.dudek@cs.uni.wroc.pl

Szymon Dudycz
University of Wroclaw, Poland
szymon.dudycz@gmail.com

Vida Dujmovic
School of Computer Science and Electrical Engineering, University of Ottawa, Ottawa, Canada
vida.dujmovic@uottawa.ca

Orr Dunkelman
Computer Science Department, University of Haifa, Haifa, Israel
orrd@cs.haifa.ac.il
https://orcid.org/0000-0001-5799-2635

Lech Duraj
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland
lech.duraj@uj.edu.pl

Zdeněk Dvořák
Charles University, Malostranske namesti 25, 11800 Prague, Czech Republic
rakdver@iuuk.mff.cuni.cz
https://orcid.org/0000-0002-8308-9746

Soheil Ehsani
University of Maryland, College Park, MD 20742, USA

Eduard Eiben
Department of Informatics, University of Bergen, Bergen, Norway
eduard.eiben@uib.no
https://orcid.org/0000-0003-2628-3435

Gil Einziger
Nokia Bell Labs, Kfar Saba, Israel
gil.einziger@nokia.com

Friedrich Eisenbrand
EPFL, 1015 Lausanne, Switzerland
friedrich.eisenbrand@epfl.ch

David Eppstein
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
eppstein@uci.edu

Martín Farach-Colton
Rutgers University, New Brunswick, NJ, USA
farach@rutgers.edu

John Fearnley
University of Liverpool, UK
john.fearnley@liverpool.ac.uk

Uriel Feige
Weizmann Institute of Science, Rehovot, Israel
uriel.feige@weizmann.ac.il

Björn Feldkord
Paderborn University, Paderborn, Germany

Matthias Feldotto
Paderborn University, Paderborn, Germany

Hendrik Fichtenberger
TU Dortmund, Dortmund, Germany
hendrik.fichtenberger@tu-dortmund.de
https://orcid.org/0000-0003-3246-5323

Diego Figueira
CNRS, LaBRI, Université de Bordeaux

Fedor V. Fomin
Department of Informatics, University of Bergen, Norway
Fedor.Fomin@uib.no
https://orcid.org/0000-0003-1955-4612

Michael A. Forbes
University of Illinois at Urbana-Champaign, USA
miforbes@illinois.edu

Jacob Fox
Department of Mathematics, Stanford University, Stanford, CA 94305, USA
jacobfox@stanford.edu

Cameron E. Freer
Borelian

Cedric Freiberger
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
cedric.freiberger@student.hpi.de

Roy Friedman
Technion, Haifa, Israel
roy@cs.technion.ac.il

Tobias Friedrich
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
tobias.friedrich@hpi.de

Martin Fränzle
Department of Computing Science, University of Oldenburg, Germany
martin.fraenzle@informatik.uni-oldenburg.de

Hao Fu
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijng, China
fu-h13@mails.tsinghua.edu.cn

Martin Gairing
Department of Computer Science, University of Liverpool, Liverpool, UK
gairing@liverpool.ac.uk

Jakub Gajarský
Technical University Berlin, Germany

Buddhima Gamlath
École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
buddhima.gamlath@epfl.ch

Moses Ganardi
Universität Siegen, Germany
ganardi@eti.uni-siegen.de

Venkata Gandikota
Department of Computer Science, Johns Hopkins University, Baltimore, Maryland, USA
gv@jhu.edu

Sumit Ganguly
Indian Institute of Technology, Kanpur, India
sganguly@cse.iitk.ac.in

Shashwat Garg
Eindhoven University of Technology, Netherlands
s.garg@tue.nl

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Ofir Geri
Department of Computer Science, Stanford University, Stanford, CA, USA
ofirgeri@cs.stanford.edu

Sumanta Ghosh
Department of Computer Science, IIT Kanpur, India
sumghosh@cse.iitk.ac.in

Suprovat Ghoshal
Indian Institute of Science, Bangalore, India
suprovat@iisc.ac.in

Yiannis Giannakopoulos
Department of Mathematics, TU Munich, Munich, Germany
Yiannis.Giannakopoulos@tum.de
https://orcid.org/0000-0003-2382-1779

Shay Golan
Bar Ilan University, Ramat Gan, Israel
golansh1@cs.biu.ac.il

Petr A. Golovach
Department of Informatics, University of Bergen, Norway
Petr.Golovach@uib.no
https://orcid.org/0000-0002-2619-2990

Michael T. Goodrich
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
goodrich@uci.edu

Themis Gouleakis
CSAIL,MIT, Cambridge, MA, USA
tgoule@mit.edu

Daniel Graf
Department of Computer Science, ETH Zürich, Switzerland
daniel.graf@inf.ethz.ch

Anael Grandjean
Laboratoire d'Algorithmique, Complexité et Logique, Université Paris-Est Créteil, France
anael.grandjean@u-pec.fr

Petr Gregor
Department of Theoretical Computer Science and Mathematical Logic, Charles University, Prague, Czech Republic
gregor@ktiml.mff.cuni.cz

Elena Grigorescu
Department of Computer Science, Purdue University, West Lafayette, Indiana, USA
elena-g@purdue.edu

Martin Grohe
RWTH Aachen University, Aachen, Germany
grohe@informatik.rwth-aachen.de
https://orcid.org/0000-0002-0292-9142

Ofer Grossman
EECS Departent, MIT, Cambridge, MA, USA
ofer.grossman@gmail.com

Yong Gu
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
guyong12@mails.tsinghua.edu.cn

Heng Guo
School of Informatics, University of Edinburgh, Informatics Forum, Edinburgh, EH8 9AB, United Kingdom
hguo@inf.ed.ac.uk
https://orcid.org/0000-0001-8199-5596

Anupam Gupta
Carnegie Mellon University, Pittsburgh, USA
anupamg@cs.cmu.edu

Manoj Gupta
IIT Gandhinagar, Gandhinagar, India
gmanoj@iitgn.ac.in

Tom Gur
UC Berkeley, Berkeley, USA
tom.gur@berkeley.edu

Rohit Gurjar
California Institute of Technology, USA

Guru Guruganesh
Carnegie Mellon University, Pittsburgh, USA

Grzegorz Gutowski
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland
grzegorz.gutowski@uj.edu.pl
https://orcid.org/0000-0003-3313-1237

Bernd Gärtner
Department of Computer Science, ETH Zürich, Switzerland
gaertner@inf.ethz.ch

Shahrzad Haddadan
Dipartimento di Informatica, Sapienza University, Rome, Italy
shahrzad.haddadan@uniroma1.it
https://orcid.org/0000-0002-7702-8250

Theophanis Hadjistasi
University of Connecticut, Storrs CT, USA
theo@uconn.edu

Bernhard Haeupler
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
haeupler@cs.cmu.edu

MohammadTaghi Hajiaghayi
University of Maryland, College Park, MD 20742, USA

Magnús M. Halldórsson
School of Computer Science, Reykjavik University, Iceland
mmh@ru.is

Thomas Dueholm Hansen
Department of Computer Science, University of Copenhagen, Denmark
dueholm@di.ku.dk

Sariel Har-Peled
Department of Computer Science, University of Illinois, Urbana, IL, USA
sariel@illinois.edu

Tobias Harks
Universität Augsburg, Institut für Mathematik, Augsburg, Germany
tobias.harks@math.uni-augsburg.de

Koyo Hayashi
Department of Mathematical Informatics, University of Tokyo, Tokyo 113-8656, Japan
koyo_hayashi@mist.i.u-tokyo.ac.jp

Benjamin Hellouin de Menibus
Laboratoire de Recherche en Informatique, Université Paris-Sud, CNRS, CentraleSupélec, Université Paris-Saclay, France
hellouin@lri.fr
https://orcid.org/0000-0001-5194-929X

Danny Hermelin
Ben-Gurion University of the Negev, Beer Sheva, Israel
hermelin@bgu.ac.il

Niklas Hjuler
University of Copenhagen, Copenhagen, Denmark
niklashjuler@gmail.com
https://orcid.org/0000-0002-0815-670X

Rani Hod
Department of Mathematics, Bar-Ilan University, Ramat Gan, Israel
rani.hod@math.biu.ac.il

Martin Hoefer
Goethe-Universität Frankfurt am Main, Institut für Informatik, Frankfurt am Main, Germany
mhoefer@cs.uni-frankfurt.de

Piotr Hofman
University of Warsaw, Poland
https://orcid.org/0000-0001-9866-3723

Jacob Holm
University of Copenhagen, Copenhagen, Denmark
jaho@di.ku.dk
https://orcid.org/0000-0001-6997-9251

Yinon Horesh
Technion - Israel Institute of Technology, Haifa, Israel
ynon980@gmail.com

Mathieu Hoyrup
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
mathieu.hoyrup@inria.fr

Justin Hsu
University College London, London, UK

Sangxia Huang
Sony Mobile Communications, Lund, Sweden
huang.sangxia@gmail.com

Zhiyi Huang
Department of Computer Sicence, The University of Hong Kong, Hong Kong
zhiyi@cs.hku.hk

Anja Huber
Universität Augsburg, Institut für Mathematik, Augsburg, Germany
anja.huber@math.uni-augsburg.de

Pavel Hubáček
Computer Science Institute, Charles University, Prague, Czech Republic
hubacek@iuuk.mff.cuni.cz

Danny Hucke
Universität Siegen, Germany
hucke@eti.uni-siegen.de

Christoph Hunkenschröder
EPFL, 1015 Lausanne, Switzerland
christoph.hunkenschroder@epfl.ch

Russell Impagliazzo
Department of Computer Science, University of California San Diego, La Jolla, CA, USA
russell@cs.ucsd.edu

Piotr Indyk
Department of Computer Science, MIT, Cambridge, MA, USA
indyk@mit.edu

Davis Issac
Max Planck Institute for Informatics, Saarland Informatics Campus, Germany
dissac@mpi-inf.mpg.de
https://orcid.org/0000-0001-5559-7471

Abhishek Jain
Johns Hopkins University
abhishek@cs.jhu.edu

Jisu Jeong
Department of Mathematical Sciences, KAIST, Daejeon, Korea
jisujeong89@gmail.com

Mark Jerrum
School of Mathematical Sciences, Queen Mary, University of London, Mile End Road, London, E1 4NS, United Kingdom
m.jerrum@qmul.ac.uk
https://orcid.org/0000-0003-0863-7279

Timothy Johnson
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
tujohnso@uci.edu

Marcin Jurdziński
University of Warwick, Coventry, UK
Marcin.Jurdzinski@warwick.ac.uk

Sven Jäger
Institut für Mathematik, Technische Universität Berlin, Germany
jaeger@math.tu-berlin.de

Daniel M. Kane
Department of Computer Science and Engineering/Department of Mathematics, University of California, San Diego
dakane@ucsd.edu
https://orcid.org/0000-0002-5884-3487

Iyad Kanj
School of Computing, DePaul University, Chicago, USA
ikanj@cs.depaul.edu

Haim Kaplan
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
haimk@tau.ac.il

Adam Karczmarz
University of Warsaw, Poland
a.karczmarz@mimuw.edu.pl

Matthew J. Katz
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel
matya@cs.bgu.ac.il

Maximilian Katzmann
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
maximilian.katzmann@hpi.de

Ken-ichi Kawarabayashi
National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
k_keniti@nii.ac.jp
https://orcid.org/0000-0001-6056-4287

Nathan Keller
Department of Mathematics, Bar-Ilan University, Ramat Gan, Israel
nkeller@math.biu.ac.il

Thomas Kesselheim
University of Bonn, Institute of Computer Science, Bonn, Germany
thomas.kesselheim@uni-bonn.de

Stefan Kiefer
University of Oxford, United Kingdom

David H. K. Kim
Computer Science Department, University of Chicago, 1100 East 58th Street, Chicago, Illinois 60637, USA
hongk@cs.uchicago.edu

Eun Jung Kim
Université Paris-Dauphine, PSL Research University, CNRS, Paris, France
eun-jung.kim@dauphine.fr

Michael P. Kim
Department of Computer Science, Stanford University, Stanford, CA, USA
mpk@cs.stanford.edu

Kim-Manuel Klein
EPFL, 1015 Lausanne, Switzerland
kim-manuel.klein@epfl.ch

Ines Klimann
Univ Paris Diderot, Sorbonne Paris Cité, IRIF, UMR 8243 CNRS, F-75013 Paris, France
klimann@irif.fr

Max Klimm
School of Business and Economics, HU Berlin, Spandauer Str. 1, 10099 Berlin, Germany
max.klimm@hu-berlin.de
https://orcid.org/0000-0002-9061-2267

Sang-Ki Ko
Korea Electronics Technology Institute, South Korea
sangkiko@keti.re.kr

Bojana Kodric
MPI for Informatics and Saarland University, Saarbrücken, Germany
bojana@mpi-inf.mpg.de

Zhuan Khye Koh
Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada
zkkoh@uwaterloo.ca

Alexandra Kolla
Department of Computer Science, University of Colorado at Boulder
alexandra.kolla@colorado.edu

Tsvi Kopelowitz
Bar Ilan University, Ramat Gan, Israel
kopelot@gmail.com

Guy Kortsarz
Rutgers University, Camden, NJ, USA
guyk@camden.rutgers.edu

Martin Koutecký
Technion - Israel Institute of Technology, Haifa, Israel, and , Charles University, Prague, Czech Republic
koutecky@technion.ac.il
https://orcid.org/0000-0002-7846-0053

Ioannis Koutis
Department of Computer Science, New Jersey Institute of Technology
ioannis.koutis@njit.edu

Dariusz R. Kowalski
Computer Science Department, University of Liverpool, Liverpool, UK
D.Kowalski@liverpool.ac.uk

Jakub Kozik
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland
jakub.kozik@uj.edu.pl

Stephan Kreutzer
Technical University Berlin
stephan.kreutzer@tu-berlin.de

Karel Král
Computer Science Institute, Charles University, Prague, Czech Republic
kralka@iuuk.mff.cuni.cz

Amit Kumar
IIT Delhi, New Delhi, India

Ananya Kumar
Carnegie Mellon University, Pittsburgh, United States
skywalker94@gmail.com

Orna Kupferman
School of Computer Science and Engineering, The Hebrew University, Israel

Dietrich Kuske
Technische Universität Ilmenau, Germany
dietrich.kuske@tu-ilmenau.de

William Kuszmaul
Department of Computer Science, Stanford University, Stanford, CA, USA
kuszmaul@cs.stanford.edu

Karim Labib
Department of Computer Science, ETH Zürich, Switzerland
labibk@student.ethz.ch

Anissa Lamani
Laboratoire MIS & Université de Picardie Jules Verne, Amiens, France
anissa.lamani@u-picardie.fr

Michael Lampis
Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243 , LAMSADE, 75016, Paris, France
michail.lampis@dauphine.fr

Gad M. Landau
University of Haifa, Israel
landau@cs.haifa.ac.il

Harry Lang
Johns Hopkins University, Baltimore, United States
hlang8@math.jhu.edu

Sławomir Lasota
University of Warsaw
sl@mimuw.edu.pl
https://orcid.org/0000-0001-8674-4470

Ranko Lazić
University of Warwick, Coventry, UK
R.S.Lazic@warwick.ac.uk

Christoph Lenzen
Max Planck Institute for Informatics, Saarbrücken, Germany
clenzen@mpi-inf.mpg.de

Jérôme Leroux
Univ.Bordeaux, CNRS, Bordeaux-INP, Talence, France
jerome.leroux@labri.fr

Reut Levi
Weizmann Institute of Science, Rehovot, Israel
reut.levi@weizmann.ac.il
https://orcid.org/0000-0003-3167-1766

Asaf Levin
Technion - Israel Institute of Technology, Haifa, Israel
levinas@ie.technion.ac.il

Bo Li
Department of Computer Science, Stony Brook University, Stony Brook, NY 11794, USA
boli2@cs.stonybrook.edu

Jason Li
Carnegie Mellon University

Jian Li
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijng, China
Correspondingauthorlijian83@mail.tsinghua.edu.cn

Yingkai Li
Department of Computer Science, Stony Brook University, Stony Brook, NY 11794, USA
yingkli@cs.stonybrook.edu

Vahid Liaghat
Facebook, Building 25, 190 Jefferson Dr, Menlo Park, CA 94025, USA

Nutan Limaye
Department of CSE, IIT Bombay, Mumbai, India
nutan@cse.iitb.ac.in

Sixue Liu
Department of Computer Science, Princeton University , 35 Olden Street, Princeton, NJ 08540, USA
sixuel@cs.princeton.edu

Yang P. Liu
MIT, Cambridge, MA
yangpatil@gmail.com

Markus Lohrey
Universität Siegen, Germany
lohrey@eti.uni-siegen.de

Daniel Lokshtanov
Department of Informatics, University of Bergen, Norway
daniello@uib.no

Anand Louis
Indian Institute of Science, Bangalore, India
anandl@iisc.ac.in

Shachar Lovett
Department of Computer Science and Engineering, University of California, San Diego
slovett@cs.ucsd.edu
https://orcid.org/0000-0003-4552-1443

Pinyan Lu
Institute for Theoretical Computer Science, Shanghai University of Finance and Economics, Shanghai 200433, China
lu.pinyan@mail.shufe.edu.cn

Anna Lubiw
School of Computer Science, University of Waterloo, Canada
alubiw@uwaterloo.ca

Anna Lysyanskaya
Computer Science Department, Brown University, Providence, RI 02912 USA
anna@cs.brown.edu

Kaifeng Lyu
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
lkf15@mails.tsinghua.edu.cn

Vivek Madan
Department of Computer Science, University of Illinois, Urbana-Champaign
vmadan2@illinois.edu

Sepideh Mahabadi
Data Science Institute, Columbia University, New York, NY, USA
mahabadi@mit.edu

Frederik Mallmann-Trenn
CSAIL, MIT, US
mallmann@mit.edu

Nil Mamano
University of California, Irvine, U.S
nmamano@uci.edu

Florin Manea
Department of Computer Science, Kiel University, D-24098 Kiel, Germany
flm@zs.uni-kiel.de

Pasin Manurangsi
University of California, Berkeley, USA
pasin@berkeley.edu

Yuchen Mao
Department of Computer Science and Engineering, HKUST, Hong Kong
ymaoad@cse.ust.hk
https://orcid.org/0000-0002-1075-344X

Liran Markin
University of Haifa, Israel
liran.markin@gmail.com

Filip Mazowiecki
Université de Bordeaux, Bordeaux, France
filip.mazowiecki@u-bordeaux.fr

Saeed Mehrabi
School of Computer Science, Carleton University, Ottawa, Canada
saeed.mehrabi@carleton.ca

Ruta Mehta
University of Illinois Urbana-Champaign, Champaign, USA
rutameht@illinois.edu

George B. Mertzios
Department of Computer Science, Durham University, UK
George.Mertzios@durham.ac.uk
https://orcid.org/0000-0001-7182-585X

Daniele Micciancio
Department of Computer Science and Engineering, University of California - San Diego, CA, USA
daniele@cs.ucsd.edu

Vahab Mirrokni
Google Research, New York

Pradipta Mitra
Google Research, New York, USA
ppmitra@gmail.com

Matthias Mnich
Universität Bonn, Germany , and Maastricht University, The Netherlands
mmnich@uni-bonn.de
https://orcid.org/0000-0002-4721-5354

Sidhanth Mohanty
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
sidhanth@cmu.edu

Marco Molinaro
PUC-Rio, Rio de Janeiro, Brazil
mmolinaro@inf.puc-rio.br

Fabrizio Montecchiani
Department of Engineering, University of Perugia, Perugia, Italy
fabrizio.montecchiani@unipg.it

Felix Montenegro-Retana
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
felix.montenegro-retana@student.hpi.de

Shay Moran
Institute for Advanced Study, Princeton
shaymoran@ias.edu
https://orcid.org/0000-0002-8662-2737

Pat Morin
School of Computer Science, Carleton University, Ottawa, Canada
morin@scs.carleton.ca

Hagar Mosaad
Department of Computer Science and Engineering, German University in Cairo, Egypt
hagar.omar@student.guc.edu.eg

Luca Moscardelli
Department of Economic Studies, University of Chieti-Pescara, Pescara, Italy
luca.moscardelli@unich.it

Miguel A. Mosteiro
Computer Science Department, Pace University, New York, NY, USA
mmosteiro@pace.edu

Anish Mukherjee
Chennai Mathematical Institute, Chennai, India
anish@cmi.ac.in

Cameron Musco
CSAIL, MIT, US
cnmusco@mit.edu

Christopher Musco
CSAIL, MIT, US
cpmusco@mit.edu

Torsten Mütze
Institut für Mathematik, Technische Universität Berlin, Germany
muetze@math.tu-berlin.de

Hiroshi Nagamochi
Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Japan
nag@amp.i.kyoto-u.ac.jp

Vasileios Nakos
Harvard University, Cambridge, USA
vasileiosnakos@g.harvard.edu

Maryam Negahbani
Department of Computer Science, Dartmouth College, 9 Maynard St, Hanover, NH, USA
maryam@cs.dartmouth.edu

Yakov Nekrich
Cheriton School of Computer Science, University of Waterloo, Canada
yakov.nekrich@googlemail.com

Daniel Neuen
RWTH Aachen University, Aachen, Germany
neuen@informatik.rwth-aachen.de

Jaroslav Nešetřil
Computer Science Institute, Charles University, Prague, Czech Republic
nesetril@iuuk.mff.cuni.cz

Rachit Nimavat
Toyota Technological Institute at Chicago, 6045 S. Kenwood Ave., Chicago, Illinois 60637, USA
nimavat@ttic.edu

Chinmay Nirkhe
Electrical Engineering and Computer Sciences, University of California, Berkeley , 387 Soda Hall Berkeley, CA 94720, U.S.A
nirkhe@cs.berkeley.edu

Reino Niskanen
Department of Computer Science, University of Liverpool, UK
r.niskanen@liverpool.ac.uk

Dirk Nowotka
Department of Computer Science, Kiel University, D-24098 Kiel, Germany
dn@zs.uni-kiel.de

Marc Noy
Universitat Politècnica de Catalunya, Barcelona, marc.noy@upc.edu

Krzysztof Onak
IBM Research, TJ Watson Research Center, Yorktown Heights, New York, USA

Shmuel Onn
Technion - Israel Institute of Technology, Haifa, Israel
onn@ie.technion.ac.il

Patrice Ossona de Mendez
CAMS (CNRS, UMR 8557), Paris, France

Rafail Ostrovsky
Department of Computer Science, University of California Los Angeles, USA
rafail@cs.ucla.edu

Joël Ouaknine
Max Planck Institute for Software Systems, Germany & , Department of Computer Science, Oxford University, UK
joel@mpi-sws.org

Sang-il Oum
Department of Mathematical Sciences, KAIST, Daejeon, Korea
sangil@kaist.edu

Katarzyna Paluch
Wroclaw University, Poland
abraka@cs.uni.wroc.pl
https://orcid.org/0000-0002-7504-6340

Fahad Panolan
Department of Informatics, University of Bergen, Norway
Fahad.Panolan@uib.no
https://orcid.org/0000-0001-6213-8687

Charles Paperman
Université de Lille

Merav Parter
Weizmann IS, Rehovot, Israel
merav.parter@weizmann.ac.il

Sarvar Patel
Google LLC, Mountain View, USA
sarvar@google.com

Boaz Patt-Shamir
Tel Aviv University, Tel Aviv, Israel
boaz@tau.ac.il

John Peebles
CSAIL,MIT, Cambridge, MA, USA
jpeebles@mit.edu

Giuseppe Persiano
Google LLC, Mountain View, USA and Università di Salerno, Salerno, Italy
giuper@gmail.com

Michał Pilipczuk
University of Warsaw, Warsaw, Poland

Thomas Place
LaBRI, University of Bordeaux and IUF, France

Ely Porat
Bar Ilan University, Ramat Gan, Israel
porately@cs.biu.ac.il

Igor Potapov
Department of Computer Science, University of Liverpool, UK
potapov@liverpool.ac.uk

Manoj Prabhakaran
Indian Institute of Technology Bombay
mp@cse.iitb.ac.in

Eric Price
UT Austin, Austin, TX, USA
ecprice@cs.utexas.edu

Gabriele Puppis
CNRS, LaBRI, Université de Bordeaux

Guillermo A. Pérez
Université libre de Bruxelles, Brussels, Belgium
gperezme@ulb.ac.be
https://orcid.org/0000-0002-1200-4952

Yuval Rabani
The Rachel and Selim Benin School of Computer Science and Engineering, The Hebrew University of Jerusalem, Israel
yrabani@cs.huji.ac.il

Saladi Rahul
Dept. of Computer Science, University of Illinois at Urbana-Champaign, USA
saladi@uiuc.edu

M. S. Ramanujan
Algorithms and Complexity Group, TU Wien, Austria
ramanujan@ac.tuwien.ac.at

M. S. Ramanujan
University of Warwick, United Kingdom
R.Maadapuzhi-Sridharan@warwick.ac.uk

Sofya Raskhodnikova
Boston University, Boston, USA
sofya@bu.edu

Mikhail Raskin
LaBRI, University of Bordeaux, 351, cours de la Libération F-33405 Talence cedex, France
raskin@mccme.ru
https://orcid.org/0000-0002-6660-5673

Gaurav Rattan
RWTH Aachen University, Aachen, Germany
grohe@informatik.rwth-aachen.de
https://orcid.org/0000-0002-5095-860X

Alexander Ravsky
Pidstryhach Institute for Applied Problems of Mechanics and Mathematics, National Academy of Science of Ukraine, Lviv, Ukraine
oravsky@mail.ru

Felix Reidl
Royal Holloway, University of London, TW20 0EX, UK
Felix.Reidl@rhul.ac.uk

Hanlin Ren
Institute for Interdisciplinary Information Sciences, Tsinghua University, China
rhl16@mails.tsinghua.edu.cn

Michael Riabzev
Technion - Israel Institute of Technology, Haifa, Israel
riabzevmichael@gmail.com

Sören Riechers
Paderborn University, Paderborn, Germany

Andrei Romashchenko
LIRMM, Univ Montpellier, CNRS, Montpellier, France; on leave from IITP RAS
andrei.romashchenko@lirmm.fr

Eyal Ronen
Computer Science Department, The Weizmann Institute, Rehovot, Israel
eyal.ronen@weizmann.ac.il

Eva Rotenberg
Technical University of Denmark, Lyngby, Denmark
erot@dtu.dk
https://orcid.org/0000-0001-5853-7909

Ron D. Rothblum
MIT and Northeastern University, Cambridge, MA
ronr@mit.edu

Tim Roughgarden
Department of Computer Science, Stanford University, Stanford, CA 94305, USA

Daniel M. Roy
Univ. Toronto

Clemens Rösner
Department of Theoretical Computer Science, University of Bonn, Germany
roesner@cs.uni-bonn.de

Aleksi Saarela
Department of Mathematics and Statistics, University of Turku, 20014 Turku, Finland
amsaar@utu.fi
https://orcid.org/0000-0002-6636-2317

Manuel Sabin
Computer Science Division, University of California Berkeley, Berkeley, CA, USA
msabin@berkeley.edu

Eden Saig
Department of Computer Science, Technion, Haifa, Israel
edens@cs.technion.ac.il
https://orcid.org/0000-0002-0810-2218

Laura Sanità
Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada
lsanita@uwaterloo.ca

Piotr Sankowski
Institute of Informatics, University of Warsaw
sank@mimuw.edu.pl

Diego Nava Saucedo
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
diego.nava-saucedo@wanadoo.fr

Saket Saurabh
Institute of Mathematical Sciences, HBNI, India and UMI ReLax
saket@imsc.res.in

Rahul Savani
University of Liverpool, UK
rahul.savani@liverpool.ac.uk

Joe Sawada
School of Computer Science, University of Guelph, Canada
jsawada@uoguelph.ca

Nitin Saxena
Department of Computer Science, IIT Kanpur, India
nitin@cse.iitk.ac.in

Marco Scarsini
Dipartimento di Economia e Finanza, LUISS, Viale Romania 32, 00197 Roma, Italy
marco.scarsini@luiss.it
https://orcid.org/0000-0001-6473-794X

Baruch Schieber
IBM Research, TJ Watson Research Center, Yorktown Heights, New York, USA

Andreas Schmid
Max Planck Institute for Informatics, Saarbrücken, Germany
aschmid@mpi-inf.mpg.de

Stefan Schmid
University of Vienna, Austria
stefan_schmid@univie.ac.at

Jens M. Schmidt
Technische Universität Ilmenau, Ilmenau, Germany
jens.schmidt@tu-ilmenau.de

Melanie Schmidt
Department of Theoretical Computer Science, University of Bonn, Germany
melanieschmidt@uni-bonn.de

Alexander A. Schwarzmann
University of Connecticut, Storrs CT, USA
ass@uconn.edu

Nicole Schweikardt
Humboldt-Universität zu Berlin, Germany
schweikn@informatik.hu-berlin.de

Pascal Schweitzer
Technische Universität Kaiserslautern, Kaiserslautern, Germany
schweitzer@cs.uni-kl.de

Saeed Seddighin
University of Maryland, College Park, MD 20742, USA

C. Seshadhri
Department of Computer Science, University of California, Santa Cruz, CA 95064, USA

Amirbehshad Shahrasbi
Carnegie Mellon University, Pittsburgh, PA, USA
shahrasbi@cs.cmu.edu

Jeffrey Shallit
School of Computer Science, University of Waterloo, Canada
shallit@cs.uwaterloo.ca

Adi Shamir
Computer Science Department, The Weizmann Institute, Rehovot, Israel
adi.shamir@weizmann.ac.il

Micha Sharir
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
michas@tau.ac.il

Roohani Sharma
Institute of Mathematical Sciences, HBNI, India and UMI ReLax
roohani@imsc.res.in

Xiaofei Shi
Carnegie Mellon University, Pittsburgh, USA
xiaofeis@andrew.cmu.edu

Philip Shilane
Dell EMC, Newtown, PA, USA
shilane@dell.com

Mahsa Shirmohammadi
CNRS & LIS, France
mahsa.shirmohammadi@lis-lab.fr

Sebastian Siebertz
University of Warsaw, Warsaw, Poland

Alexandra Silva
University College London, London, UK

Luis Fernando Schultz Xavier da Silveira
School of Computer Science and Electrical Engineering, University of Ottawa, Ottawa, Canada
schultz@ime.usp.br

Aditi Singh
IIT Gandhinagar, Gandhinagar, India
aditi.singh@iitgn.ac.in

Ali Kemal Sinop
TOBB University of Economics and Technology, Ankara, Turkey
asinop@gmail.com

Piotr Skowron
University of Warsaw, Warsaw, Poland
p.skowron@mimuw.edu.pl

Michał Skrzypczak
University of Warsaw, Banacha 2, 02-097 Warsaw, Poland
mskrzypczak@mimuw.edu.pl
https://orcid.org/0000-0002-9647-4993

Veronika Slívová
Computer Science Institute, Charles University, Prague, Czech Republic
slivova@iuuk.mff.cuni.cz

Shay Solomon
IBM Research, T. J. Watson Research Center, Yorktown Heights, New York, USA

Tasuku Soma
The University of Tokyo, Tokyo, Japan
tasuku_soma@mist.i.u-tokyo.ac.jp
https://orcid.org/0000-0001-9519-2487

Manuel Sorge
Ben-Gurion University of the Negev, Beer Sheva, Israel
sorge@post.bgu.ac.il

Krzysztof Sornat
University of Wrocław, Wrocław, Poland
krzysztof.sornat@cs.uni.wroc.pl

Jessica Sorrell
Department of Computer Science and Engineering, University of California, San Diego, CA, USA
jlsorrel@cs.ucsd.edu

Paul G. Spirakis
Department of Computer Science, University of Liverpool, UK, Department of Computer Engineering & Informatics, University of Patras, Greece
P.Spirakis@liverpool.ac.uk
https://orcid.org/0000-0001-5396-3749

Joachim Spoerhase
Lehrstuhl für Informatik I, Universität Würzburg, Germany and Institute of Computer Science, University of Wrocław, Poland
joachim.spoerhase@uni-wuerzburg
https://orcid.org/0000-0002-2601-6452

Srikanth Srinivasan
Department of Mathematics, IIT Bombay, Mumbai, India
srikanth@math.iitb.ac.in

Sam Staton
Department of Computer Science, University of Oxford, Oxford OX1 3QD UK
sam.staton@cs.ox.ac.uk

Cliff Stein
Columbia University, New York, USA

Dario Stein
Univ. Oxford

Damian Straszak
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Don M. Stull
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
donald.stull@inria.fr

Madhu Sudan
Harvard University, Cambridge, MA, USA
madhu@cs.harvard.edu

Wing-Kin Sung
National University of Singapore, Singapore
ksung@comp.nus.edu.sg

Manuel Surek
Universität Augsburg, Institut für Mathematik, Augsburg, Germany
manuel.surek@math.uni-augsburg.de

Ola Svensson
École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
ola.svensson@epfl.ch

Mani Swaminathan
Department of Computing Science, University of Oldenburg, Germany
mani.swaminathan@informatik.uni-oldenburg.de

Chaitanya Swamy
Dept. of Combinatorics and Optimization, Univ. Waterloo, Waterloo, ON N2L 3G1, Canada
cswamy@uwaterloo.ca

Géraud Sénizergues
LABRI, Bordeaux, France
geraud.senizergues@u-bordeaux.fr

Zhihao Gavin Tang
Department of Computer Sicence, The University of Hong Kong, Hong Kong
zhtang@cs.hku.hk

Marianne Thieffry
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
marianne.thieffry@student.hpi.de

Thomas Thierauf
Aalen University, Germany

Mikkel Thorup
BARC, University of Copenhagen, Universitetsparken 1, Copenhagen, Denmark
mikkel2thorup@gmail.com
https://orcid.org/0000-0001-5237-1709

Tigran Tonoyan
School of Computer Science, Reykjavik University, Iceland
ttonoyan@gmail.com

Szymon Toruńczyk
University of Warsaw, Warsaw, Poland

Konstantinos Tsakalidis
Dept. of Computer and Information Science, Tandon School of Engineering, New York University, USA
kt79@nyu.edu

Eli Upfal
Computer Science Department, Brown University, Providence, RI 02912 USA
eli@cs.brown.edu

Przemysław Uznański
Department of Computer Science, ETH Zürich, Switzerland
przemyslaw.uznanski@inf.ethz.ch

Pascal Vanier
Laboratoire d'Algorithmique, Complexité et Logique, Université Paris-Est Créteil, France
pascal.vanier@lacl.fr

Gal Vardi
School of Computer Science and Engineering, The Hebrew University, Israel

Shai Vardi
California Institute of Technology, Pasadena, CA, USA
svardi@caltech.edu

Nithin Varma
Boston University, Boston, USA
nvarma@bu.edu

Yadu Vasudev
Indian Institute of Technology Madras, Chennai, India
yadu@cse.iitm.ac.in
https://orcid.org/0000-0001-7918-7194

Umesh Vazirani
Electrical Engineering and Computer Sciences, University of California, Berkeley , 387 Soda Hall Berkeley, CA 94720, U.S.A
vazirani@cs.berkeley.edu

Rakesh Venkat
Hebrew University of Jerusalem, Israel
rakesh@cs.huji.ac.il

Suresh Venkatasubramanian
School of Computing, University of Utah, USA
suresh@cs.utah.edu

Cosimo Vinci
Department of Information Engineering Computer Science and Mathematics, University of L'Aquila, L'Aquila, Italy - Gran Sasso Science Institute, L'Aquila, Italy
cosimo.vinci@univaq.it

Emanuele Viola
Northeastern University
viola@ccs.neu.edu

Nisheeth K. Vishnoi
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Ellen Vitercik
Carnegie Mellon University, Pittsburgh, PA, USA
vitercik@cs.cmu.edu

Nils Vortmeier
TU Dortmund University, Dortmund, Germany
nils.vortmeier@tu-dortmund.de

Magnus Wahlström
Royal Holloway, University of London, TW20 0EX, UK
Magnus.Wahlstrom@rhul.ac.uk

David Wajc
Carnegie Mellon University, Pittsburgh, USA

Stefan Walzer
Technische Universität Ilmenau, Germany
stefan.walzer@tu-ilmenau.de
https://orcid.org/0000-0002-6477-0106

Fan Wei
Department of Mathematics, Stanford University, Stanford, CA 94305, USA

Oren Weimann
University of Haifa, Israel
oren@cs.haifa.ac.il

Nicole Wein
EECS, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

Armin Weiß
Universität Stuttgart, FMI, Germany
armin.weiss@fmi.uni-stuttgart.de

Daniel Wiebking
RWTH Aachen University, Aachen, Germany
wiebking@informatik.rwth-aachen.de

Sebastian Wiederrecht
TU Berlin, Germany
sebastian.wiederrecht@tu-berlin.de

Kaja Wille
Institut für Mathematik, Technische Universität Berlin, Germany
wille@math.tu-berlin.de

Richard Ryan Williams
MIT EECS and CSAIL, 32 Vassar St, Cambridge, MA 02139 USA
rrw@mit.edu
https://orcid.org/0000-0003-2326-2233

Sarah Winter
RWTH Aachen University, Germany
winter@cs.rwth-aachen.de

Felix Wolf
Technische Universität Darmstadt, Institute TEMF, Graduate School of Excellence Computational Engineering
wolf@gsc.tu-darmstadt.de

David P. Woodruff
Carnegie Mellon University, School of Computing, Pittsburg, USA
dwoodruf@cs.cmu.edu

James Worrell
University of Oxford, Oxford, UK
James.Worrell@cs.ox.ac.uk
https://orcid.org/0000-0001-8151-2443

Xiaowei Wu
Department of Computing, The Hong Kong Polytechnic University, Hong Kong
wxw0711@gmail.com

Maximilian Wötzel
BGSMath and UPC Barcelona, Barcelona, Spain
maximilian.wotzel@upc.edu
https://orcid.org/0000-0001-7591-0998

Mingyu Xiao
School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China
myxiao@gmail.com
https://orcid.org/0000-0002-1012-2373

Yuanhang Xie
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
xieyh15@mails.tsinghua.edu.cn

Pan Xu
Department of Computer Science, University of Maryland, College Park, USA
panxu@cs.umd.edu

Hongseok Yang
KAIST

Lin F. Yang
Princeton University, Princeton, United States
lin.yang@princeton.edu

Harel Yedidsion
Ben-Gurion University of the Negev, Beer Sheva, Israel
yedidsio@post.bgu.ac.il

Kevin Yeo
Google LLC, Mountain View, USA
kwlyeo@google.com

Yuichi Yoshida
National Institute of Informatics, Preferred Infrastructure, Tokyo, Japan
yyoshida@nii.ac.jp
https://orcid.org/0000-0001-8919-8479

Arman Yousefi
Department of Computer Science, University of California Los Angeles, USA
armany@cs.ucla.edu

Ching-Hua Yu
University of Illinois at Urbana-Champaign
cyu17@illinois.edu

Henry Yuen
Electrical Engineering and Computer Sciences, University of California, Berkeley , 387 Soda Hall Berkeley, CA 94720, U.S.A
hyuen@cs.berkeley.edu

Viktor Zamaraev
Department of Computer Science, Durham University, UK
Viktor.Zamaraev@durham.ac.uk
https://orcid.org/0000-0001-5755-4141

Meirav Zehavi
Department of Computer Science, Ben-Gurion University, Israel
meiravze@bgu.ac.il

Marc Zeitoun
LaBRI, University of Bordeaux, France

Georg Zetzsche
IRIF (Université Paris-Diderot & CNRS), France
https://orcid.org/0000-0002-6421-4388

Thomas Zeume
TU Dortmund University, Dortmund, Germany
thomas.zeume@tu-dortmund.de

Hongyang Zhang
Carnegie Mellon University, Pittsburgh, USA
hongyanz@cs.cmu.edu

Le Zhang
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
le-zhang12@mails.tsinghua.edu.cn

Yuhao Zhang
Department of Computer Sicence, The University of Hong Kong, Hong Kong
yhzhang2@cs.hku.hk

Zeyu Zhang
Johns Hopkins University, Baltimore, MD, USA
zyzhang92@gmail.com

Samson Zhou
Department of Computer Science, Purdue University, West Lafayette, Indiana, USA
samsonzhou@gmail.com

Marius Zimand
Department of Computer and Information Sciences, Towson University, Baltimore, MD

Consistent Distributed Memory Services: Resilience and Efficiency

Theophanis Hadjistasi
University of Connecticut, Storrs CT, USA
theo@uconn.edu

Alexander A. Schwarzmann
University of Connecticut, Storrs CT, USA
ass@uconn.edu

Abstract

Reading, *'Riting*, and *'Rithmetic*, the three *R*'s underlying much of human intellectual activity, not surprisingly, also stand as a venerable foundation of modern computing technology. Indeed, both the Turing machine and von Neumann machine models operate by reading, writing, and computing, and all practical uniprocessor implementations are based on performing activities structured in terms of the three *R*'s. With the advance of networking technology, communication became an additional major systemic activity. However, at a high level of abstraction, it is apparently still more natural to think in terms of reading, writing, and computing. While it is hard to imagine distributed systems—such as those implementing the World-Wide Web—without communication, we often imagine browser-based applications that operate by retrieving (i.e., reading) data, performing computation, and storing (i.e., writing) the results. In this article, we deal with the storage of shared readable and writable data in distributed systems that are subject to perturbations in the underlying distributed platforms composed of computers and networks that interconnect them. The perturbations may include permanent failures (or crashes) of individual computers, transient failures, and delays in the communication medium. The focus of this paper is on the implementations of distributed atomic memory services. Atomicity is a venerable notion of consistency, introduced in 1979 by Lamport [35]. To this day atomicity remains the most natural type of consistency because it provides an illusion of equivalence with the serial object type that software designers expect. We define the overall setting, models of computation, definition of atomic consistency, and measures of efficiency. We then present algorithms for single-writer settings in the static models. Then we move to presenting algorithms for multi-writer settings. For both static settings we discuss design issues, correctness, efficiency, and trade-offs. Lastly we survey the implementation issues in dynamic settings, where the universe of participants may completely change over time. Here the expectation is that solutions are found by integrating static algorithms with a reconfiguration framework so that during periods of relative stability one benefits from the efficiency of static algorithms, and where during the more turbulent times performance degrades gracefully when reconfigurations are needed. We describe the most important approaches and provide examples.

2012 ACM Subject Classification Networks → Cloud computing,General and reference → Surveys and overviews

Keywords and phrases Atomicity, shared-memory, read/write objects, fault-tolerance, latency

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.1

Category Invited Paper

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 1; pp. 1:1–1:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Shared storage services are located at the core of most information-age systems. Shared memory systems surveyed in this work provide objects that support two different access operations. That is, a *read*, that obtains the current value of the object, and a *write* that replaces the old value of the object with a new one. To be useful, such objects need to be *resilient* to failures and perturbations in the underlying computing medium, and must be *consistent* in that there are guarantees regarding relationships between previously written values and the values read by subsequent read operations. Such resilient and consistent object are also called registers. Here we focus on read/write objects, however for objects with more complicated semantics, such as transactions or read-modify-write operations, there exist common implementation challenges that any distributed storage system faces and needs to resolve. Imagine a storage system that is implemented as a central server. The server accepts client requests to perform operations on its data objects and returns responses. Conceptually, this approach is simple, however, two major problems can already be observed. The first is that the central server is a performance bottleneck. The second is that the server is a single point of failure. The quality of service in such an implementation degrades rapidly as the number of clients grows, and the service becomes unavailable if the server crashes (imagine how inadequate a web news service would be were it implemented as a central server). Thus the system must be *available*. This means it must provide its services despite failures within the scope of its specification, for example, the system must be able to mask certain server and communication failures. The system must also support multiple concurrent accesses without imposing unreasonable degradation in performance. The only way to guarantee availability is through *redundancy*, that is, by using multiple servers and by replicating the contents of objects among these servers.

Replication introduces the challenge of ensuring *consistency*. How does the system record new values so that consequently the it can find and return the latest value of a replicated object? This problem was not present with a central server implementation: the server always contains the latest value. In a replicated implementation, one may attempt to consult all replicas in search of the latest value, but this approach is expensive and not fault-tolerant as it assumes that all replicas are accessible. A trivial solution would be in each operation to consult all replicas servers in search of the latest value, however, this is not fault-tolerant (as it assumes all replicas are accessible) and expensive. In any case, none of the implementation issues should be a concern for the clients of the distributed memory service. What the clients should expect to see is the illusion of a single-copy object that serializes all accesses so that each read operation returns the value of the preceding write operation, and that this value is at least as recent as that returned by any preceding read. More generally, the behavior of the object, as observed externally, must be consistent with the abstract sequential data type of the object, and in developing applications that use such objects the clients must be able to rely on the abstract data type of the object. This notion of consistency is formalized as *atomicity* [35] for read/write objects, and equivalently, as *linearizability* [32] that extends atomicity to arbitrary data types. While there is no argument that atomicity is the most convenient notion of consistency, we note that weaker notions have also been proposed and implemented, motivated primarily by efficiency considerations. Atomicity provides strong guarantees, making it more expensive to provide than weaker consistency guarantees [7]. We take the view that it is nevertheless important to provide simple and intuitive, be it more expensive, atomic consistency. Barbara Liskov, a Turing Prize laureate, in a keynote address (at [37]) remarked that atomicity is not cheap, however, if we do not guarantee it, this creates headaches for developers.

Contemporary storage systems may also provide more complex data access primitives implementing atomic *read-modify-write* operations. Such access primitives are much stronger than separate *read* and *write* primitives we consider in this work. Implementing such operations is expensive, and at its core requires atomic updates that in practice are implemented by reducing parts of the system to a *single-writer* model (ex., Microsoft's Azure [10]), by depending on clock synchronization hardware (ex., Google's Spanner [13]), or by relying on complex mechanisms for resolving event ordering such as *vector clocks* (ex., Amazon's Dynamo [15]). Our exposition of atomic read/write storage illustrates challenges that are common to all distributed storage systems.

Document structure. Section 2 describes the general distributed setting for implementing consistent shared memory services, defines atomic conistency, and describes the measures of efficiency. In Sections 3 and 4 we present several approaches that implement consistent shared memory in static services for the SWMR and the MWMR setting respectively. Lastly, we survey several approaches for providing consistent shared memory in dynamic systems in Section 5. We conclude with a discussion in Section 6.

2 Distribution and Consistency

We now describe a general distributed setting for implementing consistent shared memory services.

Modeling distributed platforms. We model the system as a collection of interconnected computers, or nodes, that communicate by sending point-to-point messages. Each node has a unique identifier from some well-ordered set $\mathcal{I}$, local storage, and it can perform local computation. A node may fail by *crashing* at any point of the computation. The time of failure is chosen by and adversary that has knowledge of the past computation and the algorithms implementing a particular distributed service. Any node that crashes stops operating: it does not perform any local computation, it does not send any messages, and any messages sent to it are not delivered. Common approaches to implementing resilient in the face of failures algorithm specify failure models that provide qualitative or quantitative restrictions on the power of adversaries, e.g., by limiting the adversary to causing at most f crashes for some algorithm-specific parameter f.

The system is *asynchronous*, and the nodes have no access to a global clock or synchronization mechanisms. This means that relative processing speeds at the nodes can be arbitrary, and that the nodes do not know the upper bound on time that it takes to perform a local computation. The message delays can also be arbitrary, and the nodes do not know bounds on message latency (although such bounds may exist). Thus algorithms may not rely on assumptions about global time or delays.

We assume that messages can be reordered in transit, however, the messages cannot be corrupted, duplicated, or generated spontaneously. If a message is received then it must have been previously sent. The messages are not lost, but message loss can be modeled as long delays (we do not address techniques for constructing more dependable communication services, e.g., by using retransmission or gossip).

The nodes with ids in the set $\mathcal{I}$ include a set of writers $\mathcal{W}$, a set of readers $\mathcal{R}$, and a set of replica servers $\mathcal{S}$. These sets need not be disjoint, but it is helpful to view separately the roles a participant in the service can play. We categorize a distributed networked system as either *static* or *dynamic* as follows. In the *static* system the set of participating nodes is fixed,

and each node may know the identity of all participants; crashes (or voluntary departures) may remove nodes from the system. Static algorithms are commonly designed to tolerate up to $f < |\mathcal{S}|/2$ server crashes and arbitrary number of crashes among readers and writers. In the *dynamic* system the set of nodes may be unbounded, and the set of participating nodes may completely change over time as the result of crashes, departures, and new nodes joining. The failure models considered in dynamic settings are much more complicated, given the systems may dramatically evolve over time.

Distributed shared memory and consistency. A distributed shared memory service emulates a shared memory space comprised of readable and writable objects, often called registers, over a networked platform that where distributed nodes communicate by message passing. Service implementations use replication to ensure survivability and availability of the objects, but the service makes this invisible to the clients. The contents of each object is replicated across several *servers* or *replica hosts*. Clients invoke read and write operations on the objects, where the clients that perform read operations are called *readers*, and those that perform write operations are called *writers* (a client may be both a reader and a writer).

In response to client requests the service invokes a *protocol* that involves communication with the replica hosts. This protocol implements and determines the consistency guarantees of the memory system. Atomic consistency definition involves "shrinking" the duration of each operation in any execution to a chosen *serialization point* between the operation's invocation and response, and requiring that the ordering of the operations according to the serialization points preserves their real-time ordering, and the resulting behavior of the object is consistent with its sequential specification. In particular, if a read is invoked after a write completes, then the read is guaranteed to return either the value of that write, or a value written by subsequent write that precedes the read. Additionally, if a read is invoked after another read completes, it returns the same or a "newer" value than the preceding read.

Whereas atomicity is often defined in terms of an equivalence with a serial memory, the definition given below implies this equivalence (as shown in in Lemma 13.16 in [39]), and is more convenient to use because it provides a usable recipe for proving atomic consistency.

▶ **Definition 1** (Atomicity, [39])**.** An implementation of an object is atomic, if for any execution if all the read and write operations that are invoked on an object complete, then the read and write operations for the object can be partially ordered by an ordering $\prec$, so that the following conditions are satisfied:

A1. The partial order is consistent with the external order of invocations and responses, that is, there do not exist read or write operations π_1 and π_2 such that π_1 completes before π_2 starts, yet $\pi_2 \prec \pi_1$.

A2. All write operations are totally ordered and every read operation is ordered with respect to all the writes.

A3. Every read operation ordered after any writes returns the value of the last write preceding it in the partial order; any read operation ordered before all writes returns the initial value.

Efficiency, Rounds and Message Exchanges. In assessing the efficiency of read and write operations of an implementation, we measure *communication latency*, *local computation time*, and *message complexity* of operations.

Communication latency of an operation is measured in terms of *communication rounds* or *communication exchanges*. The protocol implementing each operation involves a collection of sends of typed messages and the corresponding receives. A communication round is defined following [17].

▶ **Definition 2** (Communication Round, [17]). A process p performs a communication round during an operation π in an execution of an implementation A if all the following hold:

1. process p sends message(s) for operation π to a set of processes $Z \subseteq \mathcal{I}$,
2. upon the delivery of the message for π to process q, $q \in Z$, q sends a reply for π to p without waiting for any other messages, and
3. when p receives the collection of replies that is deemed sufficient by the implementation, it terminates the round. After this either p starts a new round or π completes.

A communication exchange is defined in [30].

▶ **Definition 3** (Communication Exchange, [30]). Within an execution of an implementation A, a communication exchange is the set of sends and corresponding matching receives for a specific type of message within the protocol.

We can observe that a round in Definition 2 is composed of two exchanges: the first is comprised of sends in item (1) and the corresponding receives in item (2), and the second is comprised of the reply sends in item (2) and the corresponding receives in item (3). Thus, in essence each exchange constitutes "one half" of a round. Traditional implementations in the style of ABD are structured in terms of communication rounds, cf. [5, 26], each consisting of two exchanges. The first is a broadcast from a reader or writer process to the servers, and the second is a convergecast in which the servers send corresponding responses to the initiating process.

Computation time accounts for all local computation within an operation; here time complexity of local computation may be significant. When local computation is not more than a constant time per each message send and receive, we consider this to be insignificant relative to the communication latency of an operation. Otherwise, computation time needs to be assessed in addition to communication latency.

Message complexity of an operation is determined by the worst case number of messages sent during the operation.

3 SWMR Implementations

Algorithms designed for single-writer static settings assume a fixed set known participants and accommodate some dynamic behaviors, such as asynchrony, transient failures, and permanent crashes within certain limits. A summary of the most relevant results for this setting is given in the first part of Table 1.

We commence by presenting the the seminal work of Attiya, Bar-Noy, and Dolev [5] that provides an algorithm, colloquially referred to as ABD, that implements SWMR atomic objects in message-passing crash-prone asynchronous environments. This work won the Dijkstra Prize in 2011. In ABD replication helps achieve fault-tolerance and availability, and the implementation replicates objects at nodes in the set $\mathcal{S}$, called servers, and it tolerates f replica servers crashes, provided a majority of replicas do not fail, i.e., $|\mathcal{S}| > 2f$. Read and write operations are ordered using logical *timestamps* associated with each written value. These timestamps totally order write operations, and therefore determine the values that read operations return. All operations terminate provided a majority of replicas do not crash.

A pseudocode for ABD is given in Algorithm 1; in referring to the numbered lines of code we use the prefix "L" to stand for "line". Write operations involve a single communication round-trip consisting of *two* communication exchanges. The writer broadcasts its request to all replica servers during the first exchange and terminates once it collects acknowledgments from some majority of servers in the second exchange (L19-23). Each read operation takes two rounds involving in *four* communication exchanges. The reader broadcasts a read request

Table 1 Model, Communication Exchanges, Message Complexities, Participation Bounds, and Predicate Computational Class.

Algorithm	Model	Write Exch.	Read Exch.	Wrt Msg Comp	Rd Msg Comp	Client Participation	Local Complexity
ABD [5]	SWMR	2	4	$2\|\mathcal{S}\|$	$4\|\mathcal{S}\|$	Unbounded	Constant
Fast [17]	SWMR	2	2	$2\|\mathcal{S}\|$	$2\|\mathcal{S}\|$	$\|\mathcal{R}\| < \frac{\|\mathcal{S}\|}{f} - 2$	NP-Hard
Sf [26]	SWMR	2	2 or 4	$2\|\mathcal{S}\|$	$4\|\mathcal{S}\|$	$\|\mathcal{V}\| < \frac{\|\mathcal{S}\|}{f} - 1$	NP-Hard
Sliq [25]	SWMR	2	2 or 4	$2\|\mathcal{S}\|$	$4\|\mathcal{S}\|$	Unbounded	Constant
ccFast [4]	SWMR	2	2	$2\|\mathcal{S}\|$	$2\|\mathcal{S}\|$	$\|\mathcal{R}\| < \frac{\|\mathcal{S}\|}{f} - 2$	Polynomial
OhSam [30]	SWMR	2	3	$2\|\mathcal{S}\|$	$2\|\mathcal{S}\| + \|\mathcal{S}\|^2$	Unbounded	Constant
OhSam′ [30]	SWMR	2	2 or 3	$2\|\mathcal{S}\|$	$3\|\mathcal{S}\| + \|\mathcal{S}\|^2$	Unbounded	Constant
ccHybrid [3]	SWMR	2	2 or 4	$2\|\mathcal{S}\|$	$4\|\mathcal{S}\|$	Unbounded	Polynomial
OhFast [3]	SWMR	2	2 or 3	$2\|\mathcal{S}\|$	$\|\mathcal{S}\|^2$	Unbounded	Polynomial
MR [41]	SWMR	2	2 or 3 or 4	$\|\mathcal{S}\|^2$	$4\|\mathcal{S}\|$	Unbounded	Constant
Erato [21]	SWMR	2	2 or 3	$2\|\mathcal{S}\|$	$3\|\mathcal{S}\| + \|\mathcal{S}\|^2$	Unbounded	Constant
ABD-mw [5, 40]	MWMR	4	4	$4\|\mathcal{S}\|$	$4\|\mathcal{S}\|$	Unbounded	Constant
Sfw [18]	MWMR	2 or 4	2 or 4	$4\|\mathcal{S}\|$	$4\|\mathcal{S}\|$	Unbounded	NP-Hard
CwFr [24]	MWMR	4	2 or 4	$4\|\mathcal{S}\|$	$4\|\mathcal{S}\|$	Unbounded	Constant
OhMam [30]	MWMR	4	3	$4\|\mathcal{S}\|$	$2\|\mathcal{S}\| + \|\mathcal{S}\|^2$	Unbounded	Constant
OhMam′ [30]	MWMR	4	2 or 3	$4\|\mathcal{S}\|$	$3\|\mathcal{S}\| + \|\mathcal{S}\|^2$	Unbounded	Constant
Erato-mw [21]	MWMR	4	2 or 3	$4\|\mathcal{S}\|$	$3\|\mathcal{S}\| + \|\mathcal{S}\|^2$	Unbounded	Constant

to all replica servers in the first exchange, collects acknowledgments from some majority of servers in the second exchange, and it discovers the maximum timestamp (L3-7). In order to ensure that any subsequent read will return a value associated with a timestamp at least as high as the discovered maximum, the reader propagates the value associated with the maximum timestamp to at least a majority of servers before completion (L8-11). The correctness of this implementation, that is, atomicity, relies on the fact that any two majorities have a non-empty intersection. The local computation at readers, writers and servers in ABD incurs insignificant computational overhead.

Following ABD, a folklore belief developed that in atomic memory implementations, "reads must write." The work by Dutta et al. [17] refuted this belief by presenting an algorithm, called Fast, in which all read and write operations involve only *two* communication exchanges. Such operations are called *fast*. To avoid the second round in read operations, Fast uses two mechanisms: (i) a recording mechanism at the servers, and (ii) a predicate that uses the server records at the readers. Here, each server records in a set all processes that witness its local timestamp and resets it whenever it learns a new timestamp. Each reader explores the sets from the different server replies to determine whether "enough" processes witnessed the maximum observed timestamp. If the predicate holds, the reader returns the value associated with the maximum timestamp. Otherwise it returns the value associated with the previous timestamp. The predicate takes in account *which* processes witnessed the latest timestamp as it examines the intersection of the received sets.

It was also shown in [17] that atomic memory implementations are only possible when the number of readers is constrained in with respect to the number of replicas servers and in inverse proportion to the number of crashes as stated in the following theorem.

▶ **Theorem 4** ([17]). *Let $f \geq 1$, $|\mathcal{W}| = 1$ and $|\mathcal{R}| \geq 2$. If $|\mathcal{R}| \geq \frac{|\mathcal{S}|}{f} - 2$, then there is no fast atomic register implementation.*

Algorithm 1 Reader, Writer, and Server Protocols for SWMR algorithm ABD

```
1: At each reader r
2: function READ(v: output)
3: Get: broadcast ⟨get, i⟩ to all replica servers
4:       await responses ⟨get-ack, v′, ts′⟩
5:             from some majority of servers
6:       Let v be the value associated with the
7:       maximum timestamp maxts received
8: Put: broadcast ⟨put, v, maxts, i⟩ to all servers
9:       await responses ⟨put-ack⟩
10:            from some majority of servers
11:      return(v)

12: At each server s
13: State v init ⊥, ts init 0
14: Upon receive ⟨get, j⟩
15:      send ⟨get-ack, v, ts⟩ to j

16: At each writer w
17: State ts init 0
18: function WRITE(v: input)
19: Put: ts ← ts + 1
20:      broadcast ⟨put, v, ts, i⟩ to all servers
21:      await responses ⟨put-ack⟩
22:            from some majority of servers
23:      return()

24: At each server s
25: Upon receive ⟨put, v′, ts′, j⟩
26:   if ts′ > ts then
27:         (ts, v) ← (ts′, v′)
28:   send ⟨put-ack⟩ to j
```

A recent work by Fernández Anta, Nicolaou, and Popa [4], has shown that, although the result in [17] is efficient in terms of communication, it requires reader processes to evaluate a computationally hard predicate. The authors abstracted the predicate used in FAST as a computational problem that they show to be NP-hard via a reduction from the decision version of the Maximum Edge Biclique Problem [43], which is NP-Complete. This suggest the existence of a trade-off between communication efficiency and computational overhead in atomic memory implementations.

Given the inherent limitation on the number of readers in fast single-writer implementations, Georgiou et al. [26] sought a solution that would remove the limit on the number of readers, in exchange for slowing down some operations, i.e., the goal is to enable fast operations, but allow slower operations, taking more than two communication exchanges, when this is unavoidable. They provided a SWMR algorithm, named SF, that adopts an approach to implementing readers similar to the one in [17], but uses a polynomial time predicate to determine whether it is safe for a read operation to terminate after two exchanges. In order not to place bounds on the number of readers, the authors group readers into abstract entities, called virtual nodes, serving as enclosures for multiple readers. This refinement has a non-trivial challenge of maintaining consistency among readers within the same virtual node. This solution trades communication for the scalability in the number of participating readers. In SF significant computational overheads incur in order to determine the speed of an operation (to evaluate the mentioned predicate). At most a single complete read operation performs *four* exchanges for each write operation. Writes and any read operation that precedes or succeeds a *four* exchange read, is *fast*. This development motivated creating a new class of implementations, called *semifast* implementations. Informally, an implementation is *semifast* if either all reads are *fast* or all write operations *fast*. Algorithm SF becomes fast (same as [17]) when each virtual node contains one reader.

Georgiou et al. [25] showed that fast and semifast quorum-based SWMR implementations are possible iff a common intersection exists among all quorums. Because a single point of failure exists in such solutions (i.e., any server in the common intersection), this renders such implementations not fault-tolerant. The same work introduced Quorum Views, client-side tools that examine the distribution of the latest value among the replicas in order to enable fast read operations (two exchanges) under read and write operation concurrency. The authors derived a a SWMR algorithm, called SLIQ, that requires at least one single *slow* read per any write operation, and where all writes are *fast*. No bound is placed on the number of readers. SLIQ trades communication for the scalability in the number of participating

readers. Here only insignificant computation effort is needed to examine the distribution of object values among the replicas that the reader receives during the read operation.

Another algorithm, called ccFast, with a new predicate, is given by Fernández Anta et al. [4], that allows the operations to be *fast* with only polynomial computation overhead. The idea of the new predicate is to examine the replies received in the first communication round of a read operation and determine *how many* (instead of *which* [17]) processes witnessed the maximum timestamp among those replies. With this modification, the predicate takes polynomial time to decide the value to be returned and it reduces the size of each message sent by the replica nodes. Algorithm ccFast is more practical than [17], but it has the same constraint on the number of readers.

Previous works dealt with algorithms that used only communication between clients (readers and writers) and the replica servers. Hadjistasi, Nicolaou and Schwarzmann [30] explored another approach that exploits server-to-server communication. In particular, they showed that atomic operations do not need to involve complete communication round trips between clients and servers. The authors focused on the gap between one-round and two-round algorithms, seeking implementations where operations can take "one and a half rounds," i.e., be able to complete in *three* exchanges. They presented a SWMR algorithm, called OhSam, which stands for *One and a Half Rounds Single-writer Atomic Memory*. Here reads take *three* exchanges: the first from the client to the servers, the second is server-to-server, and the third is from the servers to the client. Such implementations trade latency for message complexity: the latency is reduced to three exchanges, while server-to-server exchange has quadratic message complexity in the number of servers. A key idea of the algorithm is that the reader returns the value that is associated with the *minimum* timestamp that corresponds to the last complete write operation (cf. the observation in [17]).

In the same work [30], authors revised the protocol implementing read operations of algorithm OhSam to yield a protocol that implements read operations that terminate in either *two* or *three* communication exchanges, OhSam′. The idea here is to let the reader determine "quickly" that a majority of servers hold the same timestamp (or tag) and its associated value. This is done by having the servers send relay messages to each other as well as to the readers. While a reader collects the relays and the read acknowledgments, if it observes in the set of the received relay messages that a majority of servers holds the same timestamp, then it safely returns the associated value and the read operation terminates in *two* exchanges. If that is not the case, then the reader proceeds similarly to algorithm OhSam and terminates in *three* communication exchanges. Algorithms OhSam and OhSam′ do not impose constraints on reader participation and perform a modest amount of local computation, resulting in negligible computation overhead.

Fernández Anta et al. [3] introduced a "multi-speed" algorithm, named ccHybrid that allows operations to terminate in either *two* or *four* communication exchanges. Algorithm ccHybrid does not impose any bounds on the number of the participating readers. ccHybrid uses the polynomial predicate introduced in [4] to determine the speed of a read operation, and it requires at most one *complete* slow operation per written value. This is similar to the semifast algorithm Sf [26]. However, in contrast with Sf, in which processes have to decide NP-hard predicates, ccHybrid performs only linear computation.

The same work [3] explores the idea of combining the polynomial predicate with the *three* communication exchanges read protocol of algorithm OhSam [30]. The resulting "multi-speed" algorithm, called OhFast, allows *one* and *one-and-a-half* round-trip operations, equivalent *two* or *three* communication exchanges. In algorithm OhFast, the decision of whether the read operation must be slow is moved to the replica servers. When replica servers determine

that a slow read is necessary, they perform a *relay* phase to inform other servers before replying to the requesting reader. It is interesting that in OhFast not all servers uniformly perform a relay for a particular read operation. Some servers may be replying directly to the requesting reader, whereas others may perform a relay phase. Thus, it is possible for a read operation to terminate before receiving a reply from the server that initiates the relay.

A recent work by Mostefaoui and Raynal [41] defines what a *time-efficient* implementation of atomic registers is based on two different synchrony assumptions. The first assumes bounded message delays and is expressed in terms of delays, and the second assumes *round-based synchrony*. Authors then present a *time-efficient* implementation of atomic registers while trying to keep its design spirit as close as possible to ABD. We refer to this solution as algorithm MR. In the algorithm a write operation takes *two* communication exchanges and a read operation takes *two*, or *three*, or *four* exchanges. The heart of the given algorithm is the *wait predicate* that takes place on the servers side and it is associated with write operations. The wait predicate ensures both atomicity and the fact that the implementation is time-efficient. The trade-off between ABD and this implementation lies in the message complexity of write operations, which for ABD is linear and for MR is quadratic in to the number of replica servers. Algorithm MR is particularly interesting for registers used in read-dominated applications.

Lastly, Georgiou et al. [21] presented algorithm Erato, which stands for *Efficient Reads for ATomic Objects*. The algorithm improves the *three*-exchange read protocol of OhSam [30] to allow reads to terminate in either *two* or *three* exchanges using client-side tools, *Quorum Views*, introduced in algorithm Sliq [25]. The three exchanges of the new read protocol are as follows: (1) reader broadcasts a request to servers; (2) the servers share this information among themselves, including the reader, and (3) once this is "sufficiently" done, servers reply to the reader. During the second exchange, the reader uses Quorum Views [25] to categorize the distribution of timestamps, and determines whether it is able to complete the read. If not, it awaits "enough" messages from the third exchange before completion. Here, the idea of the algorithm is that when the reader is "slow" it returns the value associated with the *minimum* timestamp, i.e., the value of the previous write that is guaranteed to be complete (cf. [30] and [17]). Similarly to ABD, write operations take *two* exchanges.

4 MWMR Implementations

We now turn out attention to the multi-writer/multi-reader (MWMR) implementations of atomic memory. Whereas logical timestamps alone are sufficient to order write operations in the single-writer algorithms, the existence of multiple writers requires a somewhat different approach. The simplest approach is instead of a timestamp to use pairs consisting of a timestamp and processor id to order the written values. Such a pair is termed a *tag*. When a writer performs a write operation it associates the value with a tag $\langle ts, id \rangle$, where ts is a logical timestamp, and id is the writer's unique id that distinguishes the current write operation from all others. Tags are ordered lexicographically in establishing an order on the operations. A summary of the most relevant results for this setting is given in the second part of Table 1.

The work of Lynch and Schwarzmann [40] presented a multi-writer extension of algorithm ABD (and also introduced the notion of reconfigurable memory, where the set of replica servers can be dynamically reconfigured). The static version of their MWMR implementation, that we call ABD-MW, is given in Algorithm 2. In contrast with ABD, where the sole writer generates new timestamps without any communication, the writers in ABD-MW start a

Algorithm 2 Reader, Writer, and Server Protocols for MWMR algorithm ABD-mw

```
1: At each reader r
2: function READ(v: output)
3: Get: broadcast ⟨get, i⟩ to all replica servers
4:       await responses ⟨get-ack, v′, tag′⟩
5:             from some majority of servers
6:       Let v be the value associated with the
7:       maximum tag maxtag received
8: Put: broadcast ⟨put, v, maxtag, r⟩ to all servers
9:       await responses ⟨put-ack⟩
10:            from some majority of servers
11:      return(v)

12: At each server s
13: State v init ⊥, tag init ⟨0, ⊥⟩
14: Upon receive ⟨get, j⟩
15:      send ⟨get-ack, v, tag⟩ to j

16: At each writer w
17: function WRITE(v: input)
18: Get: broadcast ⟨get, i⟩ to all replica servers
19:      await responses ⟨get-ack, v′, tag′⟩
20:            from some majority of servers
21:      Let maxtag = ⟨ts, pid⟩ be the max tag
22:      Let newtag = ⟨ts + 1, w⟩
23: Put: broadcast ⟨put, v, newtag, w⟩ to all servers
24:      await responses ⟨put-ack⟩
25:            from some majority of servers
26:      return()

27: At each server s
28: Upon receive ⟨put, v′, tag′, j⟩
29:   if tag′ > tag then
30:         (tag, v) ← (tag′, v′)
31:   send ⟨put-ack⟩ to j
```

write operation by performing an additional round in which the replica servers are queried for their latest tags. Once tags are received from a majority of servers, the writer increments the timestamp of the highest detected timestamp to produce its new tag. The the second round is performed as in ABD.

In more detail, the writer performs the "Get" round, broadcasting its request to the servers in the first exchange (L18). Servers reply with their latest timestamps in the second exchange (L18-22 and L12-15). The writer determines the highest timestamp among the replies, increments it, produces a new tag that includes its id, and then performs the "Put" round in which it in the third exchange broadcasts the new tag and the new value to all servers (L23-26). On the server side, if the incoming message contains a higher tag, then the server update its local information and send an acknowledgment in the fourth communication exchange (L27-31). The write protocol completes once the writer collects acknowledgments from a majority of servers. The first two exchanges ensure that the writer produces a tag that is higher than that of any preceding write. Thus a write operation for ABD-MW takes *four* exchanges in comparison with the *two* exchanges in ABD. The read protocol is identical to the four-exchange protocol in ABD, the only difference being that tags are used instead of timestamps. The correctness (atomicity) of this implementation, relies on the fact that any two majorities have a non-empty intersection and that in each round, the read and write protocols await responses from at least a majority of servers.

This algorithm places no constrains on the number of readers and writers, and it performs a modest amount of local computation, resulting in negligible computation overhead. This algorithm can also be used used with quorum systems instead of majorities [40, 44], because the only property of majorities that is used is that any two majorities have a non-empty intersection, just like any two quorums. The failure model for the quorum based solution is that any pattern of crashes is tolerated, provided that the servers in at least one quorums do not crash.

Algorithm ABD-MW established that two rounds are sufficient to implement atomic read and write operations. The question of whether *fast* (single round) implementations are possible was answered in the negative in [17], where it was shown that fast reads are possible only in the single-writer model SWMR. In particular, *fast* MWMR implementations are impossible when the set of readers $\mathcal{R}$ and the set of writers $\mathcal{W}$ contain more than two nodes each.

▶ **Theorem 5** ([17]). *Let $|\mathcal{W}| \geq 2$, $|\mathcal{R}| \geq 2$, and $f \geq 1$. Any atomic register implementation has a run in which some complete read or write operation is not fast.*

Moreover, Georgiou et al. [26] showed that *semifast* implementations (recall from Section 3 that in a semifast implementation either all reads are fast or all the write operations fast) are impossible in the MWMR setting.

▶ **Theorem 6** ([26]). *If $|\mathcal{W}| \geq 2$, $|\mathcal{R}| \geq 2$, and the number of server crashes $f \geq 1$, then semifast atomic register implementation is impossible.*

These impossibility results motivated the development of algorithms that allow some operations to complete in less than two rounds or in less than four communication exchanges. The work from Englert et al. [18] proposed hybrid approaches where some operations complete in *two* and other in *four* communication exchanges. Their algorithm SFW uses quorum systems and enables some reads and writes to be fast. In order to decide whether an operation can terminate after its first round, the algorithm employs two specialized predicates. However, the predicates are computationally hard (NP-hard), and fast write operations are enabled only if the quorum system satisfies certain quorum intersection properties, rendering the algorithm impractical.

Georgiou et al. [24] presented a MWMR algorithm, called CWFR, that allows fast read operations. The algorithm uses a generalization of client-side decision tools, Quorum Views, developed for the SWMR setting [25], to analyze the distribution of a value within a quorum of replies from servers to determine whether fast termination is safe. Since multiple writes can occur concurrently, an iterative technique is used to discover the latest potentially complete write operation. Here read operations terminate in either *two* or *four* communication exchanges. The write protocol is essentially the same as in ABD-MW, taking *four* exchanges to complete. Algorithm CWFR does not impose constrains on participation and it performs a modest amount of local computation, resulting in negligible computation overhead.

Hadjistasi et al. [30] sought a MWMR solution that involves *three* or *four* communication exchanges per operation, and developed algorithm OHMAM, which stands for *One and a Half Rounds Multi-writer Atomic Memory*. The authors adopted the three-exchange protocol from the SWMR algorithm OHSAM to the MWMR setting. The read protocol of OHMAM differs in that it uses tags instead of timestamps. The write protocol is identical to the one that ABD-MW uses and completes in *four* communication exchanges.

The authors then revised the protocol implementing read operations of algorithm OHMAM to yield a protocol that implements read operations that terminate in either *two* or *three* communication exchanges, the resulting algorithm is called OHMAM′. The idea here is to expedite the reader's determination that a majority of servers hold the same tag and its associated value. This is achieved by having the servers send relay messages to each other as well as to the requesting reader. While a reader collects the relays and the read acknowledgments, if it observes in the set of the received relay messages that a majority of servers holds the same timestamp, then it safely returns the associated value, thus terminating in *two* exchanges. If that is not the case, then the reader proceeds similarly to algorithm OHMAM and terminates in *three* communication exchanges. The operations perform insignificant amount of local computation.

Lastly, Georgiou et al. [21], using the SWMR algorithm ERATO as the basis, developed a MWMR algorithm, ERATO-MW. In adopting the single-writer algorithm, the challenge is that the read protocol cannot be used directly because it relies on the fact that if a write is in progress, then the preceding write is complete. Instead the algorithm implements a *three*-exchange read protocol based on [30] in combination with the iterative technique using Quorum Views as in [24]. This technique determines the completion status of a write operation, and also detects the last potentially complete write operation. Read operations complete in either *two* or *three* exchanges. Writes are similar to ABD-MW and take *four* communication exchanges. This algorithm also has a negligible computation overhead.

5 Shared Memory in Dynamic Settings

Additional challenges arise when a shared memory system must be long-lived and must ensure data longevity. A storage system may be able to tolerate failures of some servers, but over a long period it is conceivable that all servers may need to be replaced, because no servers are infallible, and also due to unavoidable changes or planned upgrades. Additionally, in mobile settings, e.g., remote search-and-rescue or military operations, it may be necessary to provide migration of data from one collection of servers to another, so that the data can move as the needs dictate. Whether our concern is data longevity or mobility, the storage system must provide seamless runtime migration of data: one cannot stop the world and reconfigure the system in response to failures and changing environment.

We now survey several approaches for providing consistent shared memory in more dynamic systems, that is, where nodes may not only crash or depart voluntarily, but where new nodes may join the service, and where the entire collection of servers need to be replaced. In general, the set of object replicas can substantially evolve over time, ultimately migrating to a completely different set of replica hosts. Thus, an implementation designed for static settings, e.g., algorithm ABD, cannot be used directly in dynamic settings because it relies on the majority of original replica hosts to always be available. In order to use an ABD-like approach in dynamic settings, one must provide some means for managing the collections of replica hosts, and to ensure that readers and writers contact suitable such collections.

It is noteworthy that dealing with dynamic settings and managing collections of nodes does not directly address the provision of consistency in memory services. Instead, these issues are representative of the broader challenges present in the realm of dynamic distributed computing. It is illustrative that implementations of consistent shared memory services can sometimes be constructed using distributed building blocks, such as those designed for managing collections of participating nodes, for providing suitable communication primitives, and for reaching agreement (consensus) in dynamic distributed settings. A tutorial covering several of these topics is presented by Aguilera et al. [2].

We start by presenting the consensus problem because it provides a natural basis for implementing an atomic memory service by establishing an agreed-upon order of operations, and because consensus is used in other ways in atomic memory implementations. Next we present group communication services (GCS) solutions that use strong communication primitives, such as totally ordered broadcast, to order operations. Finally we focus on approaches that extend the ideas of algorithm ABD to dynamic settings with explicit management of the evolving collections of replica hosts.

Consensus. Reaching agreement in distributed settings is a fundamental problem of computer science. The agreement problem in distributed settings is called *consensus*. Here a collection of processes need to agree on a value, where each process may propose a value for consideration. Any solution must satisfy the following properties: *Agreement:* no two processes decide on different values; *Validity:* the value decided was proposed by some process; *Termination:* all correct processes reach a decision. Consensus is a powerful tool in designing distributed services [39], however, consensus is a notoriously difficult problem to solve in asynchronous systems, where termination cannot be guaranteed in the presence of even a single process crash [20] (this is the seminal FLP impossibility result of Fischer, Lynch, and Paterson); thus consensus must be used with care.

Consensus algorithms can be used directly to implement an atomic data service by enabling the participants to agree on a global total ordering of all operations [36]. The

correctness (atomicity) here is guaranteed regardless of the choice of a specific consensus implementation, but the understanding of the underlying platform characteristics can guide the choice of the implementation for the benefit of system performance (for a *tour de force* of implementations see [39]). Nevertheless, using consensus for each operation is a heavy-handed approach, especially given that perturbations may delay or even prevent termination. Thus, when using consensus, one must avoid invoking it in conjunction with individual memory operations, and make operations independent of the termination of consensus.

We note that achieving consensus is a more difficult problem than implementing atomic read/write objects. In particular, consensus cannot be solved for two or more processes by using atomic read/write registers [31, 38].

Group communication services. Among the most important building blocks for distributed systems are *group communication services* (GCS) [8]. GCSs enable processes at different nodes of a network to operate collectively as a group by means of multicast services that deliver messages to the members of the group, and offer various guarantees about the order and reliability of delivery. The basis of a GCS is a *group membership service*. Each process, at any time, has a unique *view* of the group that includes a list of the processes in the group. Views can change over time, and may become different at different processes. Another important concept introduced by the GCS approach is *virtual synchrony*, where an essential requirement is that processes that proceed together through two consecutive views deliver the same set of messages between these views. This allows the recipients to take coordinated action based on the message, the membership set, and the rules prescribed by the application [8].

GCSs offer one approach for implementing shared memory. For example, one can implement a global totally ordered multicast service on top of a view-synchronous GCS [19]. The ordered multicast is used to impose an order on the memory access operations, yielding atomic memory. The main disadvantage in such solutions is that in GCS implementations, forming a new view takes time, and client memory operations are delayed (or aborted) during the view-formation period.

Another approach is to integrate a GCS with algorithm ABD as done in the dynamic primary configuration GCS of [14] that implements atomic memory by using techniques of [6] within each configuration, where configurations include a group view and a quorum system.

A general methodology for dynamic service replication is presented in [9]. This reconfiguration model unifies the virtual synchrony approach with state machine replication, as used in consensus solutions, in particular, Paxos [36].

DynaStore Algorithm. DynaStore [1] is an implementation of a dynamic atomic memory service for multi-writer/multi-reader objects. The participants start with a default local configuration, that is, some common set of replica hosts. The algorithm supports three kinds of operations: *read*, *write*, and *reconfig*. The read and write operations involve two phases, and in the absence of reconfigurations, the protocol is similar to ABD. If a participant wishes to change its current configuration, it uses the *reconfig* operation and supplies with it a set of incremental changes.

The implementation of *reconfig* involves traversals of DAG's representing possible sequences of changed configurations. In each traversal the DAG may be revised to reflect multiple changes to the same configuration. The assumption that a majority of the involved hosts are not removed and do not crash ensures that there is a path through the DAG that is guaranteed to be common among all hosts. The traversal terminates when a sink node is reached. The

reconfig protocol involves two phases. The goal of the first phase is similar to the Get phase of ABD: discover the latest value-tag pair for the object. The goal of the second phase is similar to the Put phase of ABD: convey the latest value-tag pair to a suitable majority of replica hosts. The main difference is that these two phases are performed in the context of applying the incremental changes to the configuration, while at the same time discovering the changes submitted by other participants. This "bootstraps" possible new configurations. Given that all of this is done by traversing all possible paths—and thus configurations—in the DAG's ensures that the common path is also traversed.

The *read* follows the implementation of *reconfig*, with the differences being: (a) the set of configuration changes is empty, and (b) the discovered value is returned to the client. The *write* also follows the implementation of *reconfig*, with the differences being: (a) the set of changes is empty, (b) a new, higher tag is produced upon the completion of the first phase, and (c) the new value-tag pair is propagated in the second phase.

We note that DynaStore implementation does not incorporate consensus for reconfiguration. On the other hand, reconfigurations are accomplished by additions and removals of individual nodes and this may lead to larger overheads as compared to approaches that evolve the system by replacing a complete configuration with another. Thus the latency of read and write operations are more dependent on the rate of reconfigurations. Finally, in order to guarantee termination, DynaStore assumes that reconfigurations eventually subside.

Rambo Framework. RAMBO is a dynamic memory service supporting MWMR objects [28]; RAMBO stands for Reconfigurable Atomic Memory for Basic Objects. This algorithm uses *configurations*, each consisting of a set of replica hosts plus a quorum system defined over these hosts, and supports *reconfiguration*, by which configurations can be replaced. Notably, any quorum configuration may be installed at any time, and quorums from distinct configurations are not required to have non-empty intersections. The algorithm ensures atomicity in all executions. During quiescent periods when there are no reconfigurations, the algorithm operates similarly to algorithm ABD [6, 40]. To enable long-term operation of the service, quorum configurations can be reconfigured. Reconfigurations are performed concurrently with any ongoing read and write operations, and do not directly affect such operations. Additionally, multiple reconfigurations may be in progress concurrently. Reconfiguration involves two decoupled protocols: (1) introduction of a new configuration by the component called *Recon*, and (2) upgrade to the new configuration and garbage collection of obsolete configuration(s). *Recon* always emits a unique new configuration. Different reconfiguration proposals are reconciled by executing consensus among the members of an existing configuration. Note that termination of read and write operations does not depend on termination of reconfiguration. It is the duty of a decoupled *upgrade* protocol to garbage collect old configurations and propagate the information about the object to the latest locally-known configuration. The main algorithm performs read and write operations using a two-phase strategy. The first phase gathers information from the quorums of active configurations, then the second phase propagates information to the quorums of active configurations. Note that during each phase new configurations may be discovered. To handle this each phase is terminated by a *fixed point* condition that involves a quorum from each active configuration.

Lastly, RAMBO is used as a framework for refinements and optimizations, and several subsequent works focused on practical considerations [29, 12, 22, 23, 33]. *GeoQuorums* [16] is an approach to implementing atomic shared memory on top of a physical platform that is based on mobile nodes moving in arbitrary patterns. The algorithm simplifies reconfiguration of RAMBO by using a finite set of possible configurations, and as the result it avoids the use of consensus. Here it is sufficient for a mobile node to discover the latest configuration, and contact and propagate the latest register information to all configurations.

6 Discussion

We presented several approaches for implement atomically consistent memory services in distributed message-passing systems. Our focus is on atomic consistency because it is an intuitive notion that hides the complexities of underlying implementations, presenting a convenient abstraction to the software builders. This is particularly valuable because of the common perception that the shared-memory paradigm is easier to deal with than the message-passing paradigm in designing distributed algorithms. The solutions presented in this work are representative of the different design choices available for implementing distributed memory services, and we emphasized the trade-offs present in different approaches.

We discussed in detail the issues of resilience and efficiency in the single-writer and multi-writer models in static settings. We then surveyed several approaches that implement consistent shared memory in these settings. In the established implementations frameworks some open problems remain regarding out ability to further improve the efficiency of services that use variable number of exchanges (two, three, and four) in implementing read and write operations. We also anticipate that additional lower bounds will be established to help better understand limitations on efficient implementations. For the static setting, it is also interesting to investigate the possibility of devising consistent implementations with *zero-delay* operations. That is, where operations are able to complete without additional communication, perhaps only relying on the knowledge obtain through prior communications. If the answer is in the negative, then it will still be interesting to understand the possibility of obtaining such implementations for notions of consistency weaker than atomicity, e.g., eventual consistency, or by weakening the power of adversity. As an example of this direction, the work of Chandra et al. [11] assumes a partially synchronous system using synchronized local clocks. Such solutions are particularly interesting for applications that are either read or write dominated.

The algorithms that we surveyed for the static settings may have difference fault-tolerance guarantees and be subject to efficiency trade-offs. This prompted researchers to perform empirical studies of their proposed algorithms [25, 24, 4, 3, 30, 21]. Here a goal is to understand how the analytical results are are reflected in practical efficiency. In addition to simulations, full scale cloud-based experimental evaluations will be certain to yield valuable observations in realistic settings.

In this paper we also surveyed several approaches for providing consistent shared memory in more dynamic systems, that is, where nodes may not only crash or depart voluntarily, but where new nodes may join, and where servers in one configurations can be replaced with entirely new configurations. Providing efficient atomic implementations remains challenging for dynamic settings. Here the expectation is that solutions are found by integrating static algorithms with a reconfiguration framework so that during periods of relative stability one benefits from the efficiency of static algorithms, and where during the more turbulent times performance degrades gracefully when reconfigurations are needed. One of the open questions here is whether consensus is truly necessary for implementing consistent memory services for long-lived dynamic systems.

The technical challenges and performance overheads in the dynamic setting may be the reasons why the existing distributed storage solutions shy away from atomic consistency guarantees. Commercial solutions, such as Google's File System (GFS) [27], Amazon's Dynamo [15], and Facebook's Cassandra [34], provide less-than-intuitive, unproved guarantees. The concepts discussed in section 5 are echoed in the design decisions of production systems. For instance, consensus is used in GFS [27] to ensure agreement on system configuration as it

is done in RAMBO; global time is used in Spanner [13] as it is done in *GeoQuorums*; replica access protocols in Dynamo [15] use quorums as in some approaches surveyed here. These examples provide motivation for pursuing rigorous algorithmic approaches in the study of consistent data services for dynamic networked systems. For a more detailed discussion, we direct the interested reader to related work that surveys atomic shared implementations for dynamic settings [42].

Consistent storage systems continues to be an area of active research and advanced development, and there are good reasons to believe that as high performance memory systems with superior fault-tolerance become available, they will play a significant role in the construction of sophisticated distributed applications. The demand for implementations providing atomic read/write memory will ultimately be driven by the needs of distributed applications that require provable consistency and performance guarantees.

References

1 Marcos K. Aguilera, Idit Keidar, Dahlia Malkhi, and Alexander Shraer. Dynamic atomic storage without consensus. *J. ACM*, 58:7:1–7:32, April 2011. doi:10.1145/1944345.1944348.

2 Marcos K. Aguilera, Idit Keidary, Dahlia Malkhi, Jean-Philippe Martin, and Alexander Shraery. Reconfiguring replicated atomic storage: A tutorial. *Bulletin of the EATCS*, 102:84–081, 2010.

3 Antonio Fernández Anta, Theophanis Hadjistasi, and Nicolas C. Nicolaou. Computationally light "multi-speed" atomic memory. In *20th International Conference on Principles of Distributed Systems, OPODIS 2016, December 13-16, 2016, Madrid, Spain*, pages 29:1–29:17, 2016. doi:10.4230/LIPIcs.OPODIS.2016.29.

4 Antonio Fernández Anta, Nicolas C. Nicolaou, and Alexandru Popa. Making "fast" atomic operations computationally tractable. In *19th International Conference on Principles of Distributed Systems, OPODIS 2015, December 14-17, 2015, Rennes, France*, pages 19:1–19:16, 2015. doi:10.4230/LIPIcs.OPODIS.2015.19.

5 H. Attiya, A. Bar-Noy, and D. Dolev. Sharing memory robustly in message passing systems. *Journal of the ACM*, 42(1):124–142, 1996.

6 Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *J. ACM*, 42(1):124–142, 1995. doi:10.1145/200836.200869.

7 Hagit Attiya and Jennifer L. Welch. Sequential consistency versus linearizability. *ACM Trans. Comput. Syst.*, 12(2):91–122, 1994. doi:10.1145/176575.176576.

8 Ken Birman. A history of the virtual synchrony replication model. In *Replication: Theory and Practice*, volume 5959 of *Lecture Notes in Computer Science*, pages 91–120, 2010.

9 Ken Birman, Dahlia Malkhi, and Robbert Van Renesse. Virtually synchronous methodology for dynamic service replication. Technical report, MSR-TR-2010-151, Microsoft Research, 2010.

10 Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, and Leonidas Rigas. Windows azure storage: a highly available cloud storage service with strong consistency. In Ted Wobber and Peter Druschel, editors, *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*, pages 143–157. ACM, 2011. doi:10.1145/2043556.2043571.

11 Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. An algorithm for replicated objects with efficient reads. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 325–334, 2016. doi:10.1145/2933057.2933111.

12 Gregory Chockler, Seth Gilbert, Vincent Gramoli, Peter M. Musial, and Alexander A. Shvartsman. Reconfigurable distributed storage for dynamic networks. *Journal of Parallel and Distributed Computing*, 69(1):100–116, 2009.

13 James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson C. Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally-distributed database. In *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, pages 261–264, 2012. URL: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/corbett.

14 Roberto De Prisco, Alan Fekete, Nancy A. Lynch, and Alexander A. Shvartsman. A dynamic primary configuration group communication service. In *Proc. of the 13th Int-l Symposium on Distributed Computing*, pages 64–78. Springer-Verlag, 1999. URL: http://dl.acm.org/citation.cfm?id=645956.675955.

15 Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, 2007. doi:10.1145/1323293.1294281.

16 S. Dolev, S. Gilbert, N. Lynch, A. Shvartsman, and J. Welch. Geoquorums: Implementing atomic memory in mobile ad hoc networks. In *Proceedings of 17th International Symposium on Distributed Computing (DISC)*, 2003.

17 Partha Dutta, Rachid Guerraoui, Ron R. Levy, and Arindam Chakraborty. How fast can a distributed atomic read be? In *Proceedings of the 23rd ACM symposium on Principles of Distributed Computing (PODC)*, pages 236–245, 2004.

18 Burkhard Englert, Chryssis Georgiou, Peter M. Musial, Nicolas Nicolaou, and Alexander A. Shvartsman. On the efficiency of atomic multi-reader, multi-writer distributed memory. In *Proceedings 13th International Conference On Principle Of DIstributed Systems (OPODIS 09)*, pages 240–254, 2009.

19 Alan Fekete, Nancy A. Lynch, and Alexander A. Shvartsman. Specifying and using a partitionable group communication service. *ACM Trans. Comput. Syst.*, 19(2):171–216, 2001. doi:10.1145/377769.377776.

20 Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

21 Chryssis Georgiou, Theophanis Hadjistasi, Nicolas C. Nicolaou, and Alexander A. Schwarzmann. Unleeshing and speeding up readers in atomic object implementations. In *Networked Systems - 6th International Conference, NETYS 2018, Essaouria, Morocco, May 9-11, 2018, Proceedings*, 2018.

22 Chryssis Georgiou, Peter M. Musial, and Alex A. Shvartsman. Long-lived RAMBO: Trading knowledge for communication. *Theoretical Computer Science*, 383(1):59–85, 2007.

23 Chryssis Georgiou, Peter M. Musial, and Alexander A. Shvartsman. Developing a consistent domain-oriented distributed object service. *IEEE Transactions of Parallel and Distributed Systems (TPDS)*, 20(11):1567–1585, 2009. A preliminary version of this work appeared

in the proceedings of the 4th IEEE International Symposium on Network Computing and Applications (NCA'05).

24 Chryssis Georgiou, Nicolas Nicolaou, Alexander Russel, and Alexander A. Shvartsman. Towards feasible implementations of low-latency multi-writer atomic registers. In *10th Annual IEEE International Symposium on Network Computing and Applications*, 2011.

25 Chryssis Georgiou, Nicolas C. Nicolaou, and Alexander A. Shvartsman. On the robustness of (semi) fast quorum-based implementations of atomic shared memory. In *DISC '08: Proceedings of the 22nd international symposium on Distributed Computing*, pages 289–304, Berlin, Heidelberg, 2008. Springer-Verlag. doi:10.1007/978-3-540-87779-0_20.

26 Chryssis Georgiou, Nicolas C. Nicolaou, and Alexander A. Shvartsman. Fault-tolerant semifast implementations of atomic read/write registers. *Journal of Parallel and Distributed Computing*, 69(1):62–79, 2009. doi:10.1016/j.jpdc.2008.05.004.

27 Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM. doi:10.1145/945445.945450.

28 S. Gilbert, N. Lynch, and A. Shvartsman. RAMBO: A robust, reconfigurable atomic memory service for dynamic networks. *Distributed Computing*, 23(4):225–272, December 2010.

29 Vincent Gramoli, Peter M. Musial, and Alexander A. Shvartsman. Operation liveness and gossip management in a dynamic distributed atomic data service. In *Proceedings of the ISCA 18th International Conference on Parallel and Distributed Computing Systems, September 12-14, 2005 Imperial Palace Hotel, Las Vegas, Nevada, US*, pages 206–211, 2005.

30 Theophanis Hadjistasi, Nicolas C. Nicolaou, and Alexander A. Schwarzmann. Oh-ram! one and a half round atomic memory. In *Networked Systems - 5th International Conference, NETYS 2017, Marrakech, Morocco, May 17-19, 2017, Proceedings*, pages 117–132, 2017. doi:10.1007/978-3-319-59647-1_10.

31 Maurice Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1):124–149, 1991. doi:10.1145/114005.102808.

32 Maurice P. Herlihy and Jeannette M. Wing. Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(3):463–492, 1990. doi:10.1145/78969.78972.

33 Kishori M. Konwar, Peter M. Musial, Nicolas C. Nicolaou, and Alexander A. Shvartsman. Implementing atomic data through indirect learning in dynamic networks. In *Sixth IEEE International Symposium on Network Computing and Applications (NCA 2007), 12 - 14 July 2007, Cambridge, MA, USA*, pages 223–230, 2007. doi:10.1109/NCA.2007.30.

34 Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, 2010. doi:10.1145/1773912.1773922.

35 L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programm. *IEEE Trans. Comput.*, 28(9):690–691, 1979. doi:10.1109/TC.1979.1675439.

36 Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998. doi:10.1145/279227.279229.

37 Barbara Liskov. The power of abstraction. In Nancy A. Lynch and Alexander A. Shvartsman, editors, *Distributed Computing, 24th Int-l Symposium, DISC 2010 Proc.*, volume 6343 of *LNCS*. Springer, 2010. doi:10.1007/978-3-642-15763-9.

38 Michael C. Loui and Hosame H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. In Franco P. Preparata, editor, *Parallel and Distributed Computing*, volume 4 of *Advances in Computing Research*, pages 163–183. JAI Press, Greenwich, Conn., 1987.

39 N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

40 Nancy A. Lynch and Alexander A. Shvartsman. Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts. In *Proceedings of Symposium on Fault-Tolerant Computing*, pages 272–281, 1997.

41 Achour Mostéfaoui and Michel Raynal. Time-efficient read/write register in crash-prone asynchronous message-passing systems. In *Networked Systems - 4th International Conference, NETYS 2016, Marrakech, Morocco, May 18-20, 2016, Revised Selected Papers*, pages 250–265, 2016. doi:10.1007/978-3-319-46140-3_21.

42 Peter M. Musial, Nicolas C. Nicolaou, and Alexander A. Shvartsman. Implementing distributed shared memory for dynamic networks. *Commun. ACM*, 57(6):88–98, 2014. doi:10.1145/2500874.

43 René Peeters. The maximum edge biclique problem is np-complete. *Discrete Applied Mathematics*, 131(3):651–654, 2003. doi:10.1016/S0166-218X(03)00333-0.

44 Marko Vukolic. *Quorum Systems: With Applications to Storage and Consensus*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012. doi:10.2200/S00402ED1V01Y201202DCT009.

Sparsity – an Algorithmic Perspective

Jaroslav Nešetřil
Computer Science Institute, Charles University, Prague, Czech Republic
nesetril@iuuk.mff.cuni.cz

Abstract

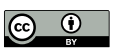
It is a well known experience that for sparse structures one can find fast algorithm for some problems which seem to be otherwise complex. The recently developed theory of sparse classes of graphs (and structures) formalizes this. Particularly the dichotomy Nowhere vs Somewhere Dense presents a very robust tool to study and design algorithms and algorithmic metatheorems. This dichotomy can be characterized in many different ways leading tp broad applications. We survey some of the recent highlights. This is a joint work with Patrice Ossona de Mendez (EHESS Paris and Charles University Prague).

2012 ACM Subject Classification Theory of computation → Theory and algorithms for application domains

Keywords and phrases sparse structures, algorithm design

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.2

Category Invited Paper



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 2; pp. 2:1–2:1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Probability Theory from a Programming Perspective

Sam Staton
Department of Computer Science, University of Oxford, Oxford OX1 3QD UK
sam.staton@cs.ox.ac.uk

Abstract

A leading idea is to apply techniques from verification and programming theory to machine learning and statistics, to deal with things like compositionality and various notions of correctness and complexity. Probabilistic programming is an example of this. Moreover, this approach leads to new foundational methods in probability theory. This is particularly true in the "non-parametric" aspects, for example in higher-order functions and infinite random graph models.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases correctness, complexity, statistics

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.3

Category Invited Paper

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 3; pp. 3:1–3:1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Lower Bounds by Algorithm Design: A Progress Report

Richard Ryan Williams[1]
MIT EECS and CSAIL, 32 Vassar St, Cambridge, MA 02139 USA
rrw@mit.edu
https://orcid.org/0000-0003-2326-2233

Abstract

In 2010, the author proposed a program for proving lower bounds in circuit complexity, via faster algorithms for circuit satisfiability and related problems. This talk will give an overview of how the program works, report on the successes of this program so far, and outline open frontiers that have yet to be resolved.

2012 ACM Subject Classification Theory of computation → Circuit complexity

Keywords and phrases circuit complexity, satisfiability, derandomization

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.4

Category Invited Paper

[1] Supported by NSF Grant CCF-1552651

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 4; pp. 4:1–4:1

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Power of d Choices with Simple Tabulation

Anders Aamand[1]
BARC, University of Copenhagen, Universitetsparken 1, Copenhagen, Denmark.
aa@di.ku.dk
https://orcid.org/0000-0002-0402-0514

Mathias Bæk Tejs Knudsen
University of Copenhagen and Supwiz, Copenhagen, Denmark.
mathias@tejs.dk
https://orcid.org/0000-0001-5308-9609

Mikkel Thorup[1]
BARC, University of Copenhagen, Universitetsparken 1, Copenhagen, Denmark.
mikkel2thorup@gmail.com
https://orcid.org/0000-0001-5237-1709

Abstract

We consider the classic d-choice paradigm of Azar et al. [STOC'94] in which m balls are put into n bins sequentially as follows: For each ball we are given a choice of d bins chosen according to d hash functions and the ball is placed in the least loaded of these bins, breaking ties arbitrarily. The interest is in the number of balls in the fullest bin after all balls have been placed.

In this paper we suppose that the d hash functions are simple tabulation hash functions which are easy to implement and can be evaluated in constant time. Generalising a result by Dahlgaard et al. [SODA'16] we show that for an arbitrary constant $d \geq 2$ the expected maximum load is at most $\frac{\lg \lg n}{\lg d} + O(1)$. We further show that by using a simple tie-breaking algorithm introduced by Vöcking [J.ACM'03] the expected maximum load is reduced to $\frac{\lg \lg n}{d \lg \varphi_d} + O(1)$ where φ_d is the rate of growth of the d-ary Fibonacci numbers. Both of these expected bounds match those known from the fully random setting.

The analysis by Dahlgaard et al. relies on a proof by Pătraşcu and Thorup [J.ACM'11] concerning the use of simple tabulation for cuckoo hashing. We require a generalisation to $d > 2$ hash functions, but the original proof is an 8-page tour de force of ad-hoc arguments that do not appear to generalise. Our main technical contribution is a shorter, simpler and more accessible proof of the result by Pătraşcu and Thorup, where the relevant parts generalise nicely to the analysis of d choices.

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization, Mathematics of computing → Random graphs, Mathematics of computing → Probabilistic algorithms, Theory of computation → Online algorithms, Theory of computation → Data structures design and analysis, Theory of computation → Bloom filters and hashing

Keywords and phrases Hashing, Load Balancing, Balls and Bins, Simple Tabulation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.5

Related Version A full version of the paper is available at [1], `https://arxiv.org/abs/1804.09684`.

[1] Research supported by Thorup's Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research and by his Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

1 Introduction

Suppose that we are to place $m = O(n)$ balls sequentially into n bins. If the positions of the balls are chosen independently and uniformly at random it is well-known that the maximum load of any bin is[2] $\Theta(\log n/\log\log n)$ whp (i.e. with probability $1 - O(n^{-\gamma})$ for arbitrarily large fixed γ). See for example [10] for a precise analysis.

Another allocation scheme is the ***d*-choice paradigm** (also called the d-choice balanced allocation scheme) first studied by Azar et al. [2]: The balls are inserted sequentially by for each ball choosing d bins, according to d hash functions $h_1, \ldots, h_d$ and placing the ball in the one of these d bins with the least load, breaking ties arbitrarily. Azar et al. [2] showed that using independent and fully random hash functions the maximum load surprisingly drops to at most $\frac{\log\log n}{\log d} + O(1)$ whp. This result triggered an extensive study of this and related types of load balancing schemes. Currently the paper by Azar et al. has more than 700 citations by theoreticians and practitioners alike. The reader is referred to the text book [13] or the recent survey [21] for thorough discussions. Applications are numerous and are surveyed in [11, 12].

An interesting variant was introduced by Vöcking [20]. Here the bins are divided into d groups each of size $g = n/d$ and for each ball we choose a single bin from each group. The balls are inserted using the d-choice paradigm but in case of ties we always choose the leftmost of the relevant bins i.e. the one in the group of the smalles index. Vöcking proved that in this case the maximum load drops further to $\frac{\log\log n}{d\log\varphi_d} + O(1)$ whp.

In this paper we study the use of simple tabulation hashing in the load balancing schemes by Azar et al. and by Vöcking.

1.1 Simple tabulation hashing

Recall that a hash function h is a map from a key universe U to a range R chosen with respect to some probability distribution on R^U. If the distribution is uniform we say that h is fully random but we may impose any probability distribution on R^U.

Simple tabulation hashing was first introduced by Zobrist [23]. In simple tabulation hashing $U = [u] = \{0, 1, \ldots, u-1\}$ and $R = [2^r]$ for some r. We identify R with the $\mathbb{Z}_2$-vector space $(\mathbb{Z}_2)^r$. The keys $x \in U$ are viewed as vectors consisting of $c > 1$ characters $x = (x[0], \ldots, x[c-1])$ with each $x[i] \in \Sigma \stackrel{def}{=} [u^{1/c}]$. We always assume that $c = O(1)$. The simple tabulation hash function h is defined by

$$h(x) = \bigoplus_{i=0}^{c-1} h_i(x[i])$$

where $h_0, \ldots, h_{c-1} : \Sigma \to R$ are chosen independently and uniformly at random from R^Σ. Here $\oplus$ denotes the addition in R which can in turn be interpreted as the bit-wise XOR of the elements $h_i(x[i])$ when viewed as bit-strings of length r.

Simple tabulation is trivial to implement, and very efficient as the character tables $h_0, \ldots, h_{c-1}$ fit in fast cache. Pătraşcu and Thorup [15] considered the hashing of 32-bit keys divided into 4 8-bit characters, and found it to be as fast as two 64-bit multiplications. On computers with larger cache, it may be faster to use 16-bit characters. We note that the c character table lookups can be done in parallel and that character tables are never changed once initialised.

[2] All logarithms in this paper are binary.

In the d-choice paradigm, it is very convenient that all the output bits of simple tabulation are completely independent (the jth bit of $h(x)$ is the XOR of the jth bit of each $h_i(x[i])$). Using (dr)-bit hash values, can therefore be viewed as using d independent r-bit hash values, and the d choices can thus be computed using a single simple tabulation hash function and therefore only c lookups.

1.2 Main results

We will study the maximum load when the elements of a fixed set $X \subset U$ with $|X| = m$ are distributed into d groups of bins $G_1, \dots, G_d$ each of size $g = n/d$ using the d-choice paradigm with independent simple tabulation hash functions $h_1, \dots, h_d : U \to [n/d]$. The d choices thus consist of a single bin from each group as in the scheme by Vöcking but for $x \in X$ we may identify $h_i(x)$ with $(h_i(x), i) \in [n/d] \times [d]$ and thus think of all h_i as mapping to the same set of bins like in the scheme by Azar et al.

Dahlgaard et al. [7] analysed the case $d = 2$ proving that if $m = O(n)$ balls are distributed into two tables each consisting of $n/2$ bins according to the two choice paradigm using two independently chosen simple tabulation hash functions, the maximum load of any bin is $O(\log \log n)$ whp. For $k = O(1)$ they further provided an example where the maximum load is at least $\lfloor k^{c-1}/2 \rfloor \log \log n - O(1)$ with probability $\Omega(n^{-2(k-1)(c-1)})$. Their example generalises to arbitrary fixed $d \geq 2$ so we cannot hope for a maximum load of $(1+o(1))\frac{\log \log n}{\log d}$ or even $100 \times \log \log n$ whp when d is constant. However, as we show in the full version of this paper [1], their result implies that even with $d = O(1)$ choices the maximum load is $O(\log \log n)$ whp.

Dahlgaard et al. also proved that the expected maximum load is at most $\log \log n + O(1)$ when $d = 2$. We prove the following result which generalises this to arbitrary $d = O(1)$.

▶ **Theorem 1.** *Let $d > 1$ be a fixed constant. Assume $m = O(n)$ balls are distributed into d tables each of size n/d according to the d-choice paradigm using d independent simple tabulation hash functions $h_1, \dots, h_d : U \to [n/d]$. Then the expected maximum load is at most $\frac{\log \log n}{\log d} + O(1)$.*

When in the d-choice paradigm we sometimes encounter ties when placing a ball — several bins among the d choices may have the same minimum load. As observed by Vöcking [20] the choice of tie breaking algorithm is of subtle importance to the maximum load. In the fully random setting, he showed that if we use the **Always-Go-Left algorithm** which in case of ties places the ball in the leftmost of the relevant bins, i.e. in the bin in the group of the smallest index, the maximum load drops to $\frac{\log \log n}{d \log \varphi_d} + O(1)$ whp. Here φ_d is the *unique* positive real solution to the equation $x^d = x^{d-1} + \cdots + x + 1$. We prove that his result holds in expectation when using simple tabulation hashing.

▶ **Theorem 2.** *Suppose that we in the setting of Theorem 1 use the Always-Go-Left algorithm for tie-breaking. Then the expected maximum load of any bin is at most $\frac{\log \log n}{d \log \varphi_d} + O(1)$.*

Note that φ_d is the rate of growth of the so called d-ary Fibonacci numbers for example defined by $F_d(k) = 0$ for $k \leq 0$, $F_d(1) = 1$ and finally $F_d(k) = F_d(k-1) + \cdots + F_d(k-d)$ when $k > 1$. With this definition we can write $\varphi_d = \lim_{k\to\infty} \sqrt[k]{F_d(k)}$. It is easy to check that $(\varphi_d)_{d>1}$ is an increasing sequence converging to 2.

1.3 Technical contributions

In proving Theorem 1 we would ideally like to follow the approach by Dahlgaard et al. [7] for the case $d = 2$ as close as possible. They show that if some bin gets load $k + 1$ then

either the hash graph (informally, the d-uniform hypergraph with an edge $\{h_1(x), \ldots, h_d(x)\}$ for each $x \in X$) contains a subgraph of size $O(k)$ with more edges than nodes or a certain kind of "witness tree" T_k. They then bound the probability that either of these events occur when $k = \log\log n + r$ for some sufficiently large constant r. Putting $k = \frac{\log\log n}{\log d} + r$ for a sufficiently large constant r we similarly have three tasks:

(1) Define the d-ary witness trees and argue that if some bin gets load $k+1$ then either **(A)**: the hash graph contains a such, or **(B)**: it contains a subgraph $G = (V, E)$ of size $O(k)$ with $|V| \leq (d-1)|E| - 1$.
(2) Bound the probability of **(A)**.
(3) Bound the probability of **(B)**.

Step (1) and (2) require intricate arguments but the techniques are reminiscent to those used by Dahlgaard et al. in [7] and it is not surprising that their arguments generalise to our setting. Due to space limitations this part of our analysis can be found in the full version of this paper [1].

Our main technical contribution is our work on step (3) as we now describe. Dealing with step (3) in the case $d = 2$ Dahlgaard et al. used the proof by Pătraşcu and Thorup [15] of the result below concerning the use of simple tabulation for cuckoo hashing[3].

▶ **Theorem 3** (Pătraşcu and Thorup [15]). *Fix $\varepsilon > 0$. Let $X \subset U$ be any set of m keys. Let n be such that $n > 2(1+\varepsilon)m$. With probability $1 - O(n^{-1/3})$ the keys of X can be placed in two tables of size $n/2$ with cuckoo hashing using two independent simple tabulation hash functions h_0 and h_1.*

Unfortunately for us, the original proof of Theorem 3 consists of 8 pages of intricate ad-hoc arguments that do not seem to generalise to the d-choice setting. Thus we have had to develop an alternative technique for dealing with step (3) As an extra reward this technique gives a new proof of Theorem 3 which is shorter, simpler and more readable and we believe it to be our main contribution and of independent interest[4].

1.4 Alternatives

We have shown that balanced allocation with d choices with simple tabulation gives the same expected maximum load as with fully-random hashing. Simple tabulation uses c lookups in tables of size $u^{1/c}$ and $c-1$ bit-wise XOR. The experiments from [15], with $u = 2^{32}$ and $c = 4$, indicate this to be about as fast as two multiplications.

Before comparing with alternative hash functions, we note that we may assume that $u \leq n^2$. If u is larger, we can first apply a universal hash function [3] from $[u]$ to $[n^2]$. This yields an expected number of $\binom{n}{2}/n^2 < 1/2$ collisions. We can now apply *any* hash function, e.g., simple tabulation, to the reduced keys in $[n^2]$. Each of the duplicate keys can increase the maximum load by at most one, so the expected maximum load increases by at most $1/2$. If $u = 2^w$, we can use the extremely simple universal hash function from [8], multiplying the key by a random odd w-bit number and performing a right-shift.

Looking for alternative hash functions, it can be checked that $O(\log n)$-independence suffices to get the same maximum load bounds as with full randomness even with high

[3] Recall that in cuckoo hashing, as introduced by Pagh and Rodler [14], we are in the 2-choice paradigm but we require that no two balls collide. However, we are allowed to rearrange the balls at any point and so the feasibility does only depend on the choices of the balls.
[4] We mention in passing that Theorem 3 is best possible: There exists a set X of m keys such that with probability $\Omega(n^{-1/3})$ cuckoo hashing is forced to rehash (see [15]).

probability. High independence hash functions were pioneered by Siegel [17] and the most efficient construction is the double tabulation of Thorup [18]. It gives independence $u^{\Omega(1/c^2)}$ using space $O(cu^{1/c})$ in time $O(c)$. With c a constant this would suffice for our purposes. However, looking into the constants suggested in [18], with 16-bit characters for 32-bit keys, we have 11 times as many character table lookups with double tabulation as with simple tabulation and we loose the same factor in space, so this is not nearly as efficient.

Another approach was given by Woelfel [22] using the hash functions he earlier developed with Dietzfelbinger [9]. He analysed Vöcking's Always-Go-Left algorithm, bounding the error probability that the maximum load exceeded $\frac{\log\log n}{d\log\varphi_d}+O(1)$. Slightly simplified and translated to match our notation, using $d+1$ k-independent hash functions and d lookups in tables of size $n^{2/c}$, the error probability is $n^{1+o(1)-k/c}$. Recall that we may assume $n^{2/c}\geq u^{1/c}$, so this matches the space of simple tabulation with c characters. With, say, $c=4$, he needs 5-independent hashing to get any non-trivial bound, but the fastest 5-independent hashing is the tabulation scheme of Thorup and Zhang [19], which according to the experiments in [15] is at least twice as slow as simple tabulation, and much more complicated to implement.

A final alternative is to compromise with the constant evaluation time. Reingold et al. [16] have shown that using the hash functions from [4] yields a maximum load of $O(\log\log n)$ whp. The functions use $O(\log n\log\log n)$ random bits and can be evaluated in time $O((\log\log n)^2)$. Very recently Chen [5] used a refinement of the hash family from [4] giving a maximum load of at most $\frac{\log\log n}{\log d}+O(1)$ whp and $\frac{\log\log n}{d\log\varphi_d}+O(1)$ whp using the Always-Go-Left algorithm. His functions require $O(\log n\log\log n)$ random bits and can be evaluated in time $O((\log\log n)^4)$. We are not so concerned with the number of random bits. Our main interest in simple tabulation is in the constant evaluation time with a very low constant.

1.5 Structure of the paper

In Section 2 we provide a few preliminaries for the proofs of our main results. In Section 3 we deal with step (3) described under *Technical contributions*. To provide some intuition we first provide the new proof of Theorem 3. Finally, we show how to proceed for general d. For step (1) and (2) as well as the final deduction of Theorem 1 and Theorem 2 the reader is referred to the full version of this paper [1].

2 Preliminaries

First, recall the definition of a hypergraph:

▶ **Definition 4.** A **hypergraph** is a pair $G=(V,E)$ where V is a set and E is a multiset consisting of elements from $\mathcal{P}(V)$. The elements of V are called **vertices** and the elements of E are called **edges**. We say that G is ***d*-uniform** if $|e|=d$ for all $e\in E$.

When using the d-choice paradigm to distribute a set of keys X there is a natural d-uniform hypergraph associated with the keys of X.

▶ **Definition 5.** Given a set of keys $X\subset U$ the **hash graph** is the d-uniform hypergraph on $[n/d]\times[d]$ with an edge $\{(h_1(x),1),\dots,(h_d(x),d)\}$ for each $x\in X$.

When working with the hash graph we will hardly ever distinguish between a key x and the corresponding edge, since it is tedious to write $\{(h_i(x),i)\}_{1\leq i\leq d}$. Statements such as "$P=(x_1,\dots,x_t)$ is a path" or "The keys x_1 and x_2 are adjacent in the hash graph" are examples of this abuse of notation.

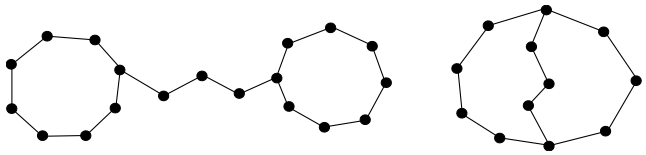


Figure 1 Double cycles - the minimal obstructions for cuckoo hashing.

Now we discuss the independence of simple tabulation. First recall that a **position character** is an element $(j, \alpha) \in [c] \times \Sigma$. With this definition a key $x \in U$ can be viewed as the set of position characters $\{(i, x[i])\}_{i=0}^{c-1}$ but it is sensible to define $h(S) = \bigoplus_{i=1}^{k} h_{j_i}(\alpha_i)$ for any set $S = \{(j_1, \alpha_1), \ldots, (j_k, \alpha_k)\}$ of position characters.

In the classical notion of independence of Carter and Wegman [3] simple tabulation is not even 4-independent. In fact, the keys $(a_0, b_0), (a_0, b_1), (a_1, b_0)$ and (a_1, b_1) are dependent, the issue being that each position character appears an even number of times and so the bitwise XOR of the hash values will be the zero string. As proved by Thorup and Zhang [19] this property in a sense characterises dependence of keys.

▶ **Lemma 6** (Thorup and Zhang [19]). *The keys $x_1, \ldots, x_k \in U$ are dependent if and only if there exists a non-empty subset $I \subset \{1, \ldots, k\}$ such that each position character in $(x_i)_{i \in I}$ appears an even number of times. In this case we have that $\bigoplus_{i \in I} h(x_i) = 0$.*

When each position character appears an even number of times in $(x_i)_{i \in I}$ we will write $\bigoplus_{i \in I} x_i = \emptyset$ which is natural when we think of a key as a set of position characters and $\oplus$ as the symmetric difference. As shown by Dahlgaard et al. [6] the characterisation in Lemma 6 can be used to bound the independence of simple tabulation.

▶ **Lemma 7** (Dahlgaard et al. [6]). *Let $A_1, \ldots, A_{2t} \subset U$. The number of $2t$-tuples $(x_1, \ldots, x_{2t}) \in A_1 \times \cdots \times A_{2t}$ such that $x_1 \oplus \cdots \oplus x_{2t} = \emptyset$ is at most*[5] $((2t-1)!!)^c \prod_{i=1}^{2t} \sqrt{|A_i|}$.

This lemma will be of extreme importance to us. In the full version of this paper [1] proofs of both Lemma 6 and Lemma 7 can be found.

3 Cuckoo hashing and generalisations

The following result is a key ingredient in the proofs of Theorem 1 and Theorem 2.

▶ **Theorem 8.** *Suppose that we are in the setting of Theorem 1 i.e. $d > 1$ is a fixed constant, $X \subset U$ with $|X| = m = O(n)$ and $h_1, \ldots, h_d : U \to [n/d]$ are independent simple tabulation hash functions. The probability that the hash graph contains a subgraph $G = (V, E)$ of size $|E| = O(\log \log n)$ with $|V| \leq (d-1)|E| - 1$ is at most $n^{-1/3+o(1)}$.*

Before giving the full proof however we provide the new proof of Theorem 3 which is more readable and illustrates nearly all the main ideas.

Proof of Theorem 3. It is well known that cuckoo hashing is possible if and only if the hash graph contains no subgraph with more edges than nodes. A minimal such graph is called a **double cycle** and consists of two cycles connected by a path or two vertices connected by

[5] Recall the double factorial notation: If a is a positive integer we write $a!!$ for the product of all the positive integers between 1 and a that have the same parity as a.

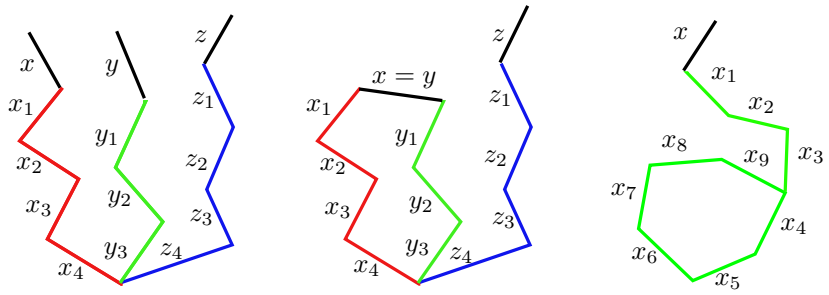


Figure 2 Non-black edges: Two tridents and a lasso. Black edges: Keys that are each dependent on the set of coloured keys.

three disjoint paths (see Figure 1). Hence, it suffices to bound the probability that the hash graph contains a double cycle by $O(n^{-1/3})$.

We denote by g the number of bins in each of the two groups. Thus in this setting $g = n/2 \geq (1+\varepsilon)m$. First of all, we argue that we may assume that the hash graph contains no trail of length at least $\ell = \frac{4}{3}\frac{\log n}{\log(1+\varepsilon)}$ consisting of *independent*. Indeed, the keys of a such can be chosen in at most m^ℓ ways and since we require $\ell - 1$ equations of the form $h_i(x) = h_i(y)$, $i \in \{1,2\}$ to be satisfied and since these events are independent the probability that the hash graph contains such a trail is by a union bound at most

$$\frac{2m^\ell}{g^{\ell-1}} \leq \frac{n}{(1+\varepsilon)^\ell} = n^{-1/3}.$$

Now we return to the double cycles. Let A_ℓ denote the event that the hash graph contains a double cycle of size ℓ consisting of *independent* keys. The graph structure of a such can be chosen in $O(\ell^2)$ ways and the keys (including their positions) in at most m^ℓ ways. Since there are $\ell + 1$ equations of the form $h_i(x) = h_i(y)$, $i \in \{1,2\}$ to be satisfied the probability that the hash graph contains a double cycle consisting of independent keys is at most

$$\sum_{\ell=3}^{m} \mathbb{P}(A_\ell) = O\left(\sum_{\ell=3}^{m} \ell^2 \frac{m^\ell}{g^{\ell+1}}\right) = O\left(\frac{1}{n}\sum_{\ell=3}^{m} \frac{\ell^2}{(1+\varepsilon)^\ell}\right) = O(n^{-1}).$$

The argument above is the same as in the fully random setting. We now turn to the issue of dependencies in the double cycle starting with the following definition.

▶ **Definition 9.** We say that a graph is a **trident** if it consists of three paths P_1, P_2, P_3 of non-zero lengths meeting at a single vertex v. (see the non-black part of Figure 2).

We say that a graph is a **lasso** if it consists of a path that has one end attached to a cycle (see the non-black part of Figure 2).

We claim that in any double cycle D consisting of *dependent* keys we can find one of the following structures (see Figure 2):

- **S1:** A lasso L consisting of independent keys together with a key x not on L and incident to the degree 1 vertex of L such that x is dependent on the keys of L.
- **S2:** A trident T consisting of independent keys together with 3 (not necessarily distinct) keys x, y, z not on T but each dependent on the keys of T and incident to the 3 vertices of degree 1 on T

To see this suppose first that one of the cycles C of D consists of independent keys. In this case any maximal lasso of independent keys in D containing the edges of C is an S_1.

On the other hand if all cycles contained in D consist of dependent keys we pick a vertex of D of degree at least 3 and 3 incident edges. These 3 edges form an independent trident (simple tabulation is 3-independent) and any maximal independent trident contained in D and containing these edges forms an S_2.

Our final step is thus to show that the probability that these structures appear in the hash graph is $O(n^{-1/3})$

The lasso (S_1):

Since the edges of the lasso form an independent trail it by the initial observation suffices to bound the probability that the hash graph contains an S_1 of size ℓ for any $\ell = O(\log n)$.

Fix the size ℓ of the lasso. The number of ways to choose the graph structure of the lasso is $\ell - 2 < \ell$. Denote the set of independent keys of the lasso by $S = \{x_1, \ldots, x_\ell\}$ and let x be the dependent key in S_1. By Lemma 6 we may write $x = \bigoplus_{i \in I} x_i$ for some $I \subset \{1, \ldots, \ell\}$. Fix the size $|I| = t \geq 3$ (which is necessarily odd). By Lemma 7 the number of ways to choose the keys of $(x_i)_{i \in I}$ (including their order) is at most $(t!!)^c m^{(t+1)/2}$ and the number of ways to choose their positions in the lasso is $\binom{\ell}{t}$. The number of ways to choose the remaining keys of S is trivially bounded by $m^{\ell - t}$ and the probability that the choice of independent keys hash to the correct positions in the lasso is at most $2/g^\ell$. By a union bound the probability that the hash graph contains an S_1 for fixed values of ℓ and t is at most

$$\ell (t!!)^c m^{(t+1)/2} m^{\ell - t} \binom{\ell}{t} \frac{2}{g^\ell}.$$

This is maximised for $t = 3$. In fact, when $\ell \leq m^{1/(c+2)}$ and $t \leq \ell - 2$ we have that

$$\frac{((t+2)!!)^c m^{(t+3)/2} m^{\ell - t - 2} \binom{\ell}{t+2}}{(t!!)^c m^{(t+1)/2} m^{\ell - t} \binom{\ell}{t}} = \frac{(t+2)^c}{m} \frac{\binom{\ell - t}{2}}{\binom{t+2}{2}} \leq \frac{\ell^{c+2}}{m} \leq 1.$$

Thus the probability that the hash graph contains an S_1 of size $O(\log n)$ is at most

$$\sum_{\ell=3}^{O(\log n)} \sum_{t=3}^{\ell} \ell 3^c \binom{\ell}{3} \frac{2 m^{\ell - 1}}{g^\ell} = O\left(\sum_{\ell=3}^{O(\log n)} \frac{\ell^5}{n(1+\varepsilon)^{\ell-1}} \right) = O(n^{-1}).$$

The trident (S_2):

Fix the size ℓ of the trident. The number of ways to choose the structure of the trident is bounded by ℓ^2 (once we choose the lengths of two of the paths the length of the third becomes fixed). Let $P_1 = (x_1, \ldots, x_{t_1})$, $P_2 = (y_1, \ldots, y_{t_2})$ and $P_3 = (z_1, \ldots, z_{t_3})$ be the three paths of the trident meeting in $x_{t_1} \cap y_{t_2} \cap z_{t_3}$. As before we may assume that each has length $O(\log n)$. Let S denote the keys of the trident and enumerate $S = \{w_1, \ldots, w_\ell\}$ in some order. Write $x = \bigoplus_{i \in I} w_i$, $y = \bigoplus_{j \in J} w_j$ and $z = \bigoplus_{k \in K} w_k$ for some $I, J, K \subset \{1, \ldots, \ell\}$. By a proof almost identical to that given for the lasso we may assume that $|I| = |J| = |K| = 3$. Indeed, if for example $|I| \geq 5$ we by Lemma 7 save a factor of nearly m^2 when choosing the keys of S and this makes up for the fact that the trident contains no cycles and hence that the probability of a fixed set of independent keys hashing to it is a factor of g larger.

The next observation is that we may assume that $|I \cap J|, |J \cap K|, |K \cap I| \geq 2$. Again the argument is of the same flavour as the one given above. If for example $|I \cap J| = 1$ we by an application of Lemma 7 obtain that the number of ways to choose the keys of $(w_i)_{i \in I}$ is $O(m^2)$. Conditioned on this, the number of ways to choose the keys $(w_j)_{j \in J}$ is $O(m^{3/2})$ by

another application of Lemma 7 with one of the A_i's a singleton. Thus we save a factor of $m^{3/2}$ when choosing the keys of S which will again suffice. The bound gets even better when $|I \cap J| = 0$ where we save a factor of m^2.

Suppose now that x_1 is not a summand of $\bigoplus_{i\in I} w_i$. Write $x = w_a \oplus w_b \oplus w_c$ and let A be the event that the independent keys of S hash to the trident (with the equation involving x_1 and x_2 being $h_2(x_1) = h_2(x_2)$ without loss of generality). Then $\mathbb{P}(A) = \frac{1}{g^{\ell-1}}$. We observe that

$$\mathbb{P}(h_1(x) = h_1(x_1) \,|\, A) = \mathbb{P}(h_1(x_1) = h_1(w_a) \oplus h_1(w_b) \oplus h_1(w_c) \,|\, A) = g^{-1}$$

since A is a conjunction of events of the form $\{h_i(w) = h_i(w')\}$ none of them involving $h_1(x_1)$[6]. A union bound then gives that the probability that this can happen is at most

$$\sum_{\ell=3}^{O(\log n)} \ell^2 \binom{\ell}{3} (3!!)^c m^2 m^{\ell-3} \left(\frac{1}{g}\right)^{\ell} = O\left(\frac{1}{n} \sum_{\ell=3}^{\infty} \frac{\ell^5}{(1+\varepsilon)^{\ell-1}}\right) = O(n^{-1}).$$

Thus we may assume that x_1 is a summand of $\bigoplus_{i\in I} w_i$ and by similar arguments that y_1 is a summand of $\bigoplus_{j\in J} w_j$ and that z_1 is a summand of $\bigoplus_{k\in K} w_k$.

To complete the proof we need one final observation. We can define an equivalence relation on $X \times X$ by $(a, b) \sim (c, d)$ if $a \oplus b = c \oplus d$. Denote by $\mathcal{C} = \{C_1, \ldots, C_r\}$ the set of equivalence classes. One of them, say C_1, consists of the elements $(x, x)_{x\in X}$. We will say that the equivalence class C_i is **large** if $|C_i| \geq m^{2/3}$ and **small** otherwise. Note that

$$\sum_{i=1}^{r} |C_i|^2 = |\{(a, b, c, d) \in X^4 : a \oplus b \oplus c \oplus d = \emptyset\}| \leq 3^c m^2$$

by Lemma 7. In particular the number of large equivalence classes is $O(m^{2/3})$.

If h is a simple tabulation hash function we can well-define a map $\tilde{h} : \mathcal{C} \to R$ by $\tilde{h}(a, b) = h(a) \oplus h(b)$. Since the number of large equivalence classes is $O(m^{2/3})$ the probability that $\tilde{h}_i(C) = 0$ for some large $C \in \mathcal{C}\backslash\{C_1\}$ and some $i \in \{1, 2\}$ is $O(m^{2/3}/n) = O(n^{-1/3})$ and we may thus assume this does not happen.

In particular, we may assume that (x, x_1), (y, y_1) and (z, z_1) each represent small equivalence classes as they are adjacent in the hash graph. Now suppose that y_1 is not a summand in $x = \bigoplus_{i\in I} w_i$. The number of ways to pick $(x_i)_{i\in I}$ is at most $3^c m^2$ by Lemma 7. By doing so we fix the equivalence class of (y, y_1) but not y_1 so conditioned on this the number of ways to pick $(y_j)_{j\in J}$ is at most $m^{2/3}$. The number of ways to choose the remaining keys is bounded by $m^{\ell-4}$ and a union bound gives that the probability of having such a trident is at most

$$\sum_{\ell=3}^{O(\log n)} \ell^2 3 \binom{\ell}{2} 3^c m^2 m^{2/3} m^{\ell-4} \left(\frac{1}{g}\right)^{\ell-1} = O\left(n^{-1/3} \sum_{\ell=3}^{\infty} \frac{\ell^4}{(1+\varepsilon)^{\ell-4/3}}\right) = O(n^{-1/3}),$$

which suffices.

We may thus assume that y_1 is a summand in $\bigoplus_{i\in I} w_i$ and by an identical argument that z_1 is a summand in $\bigoplus_{i\in I} w_i$ and hence $x = x_1 \oplus y_1 \oplus z_1$. But the same arguments apply to y and z reducing to the case when $x = y = z = x_1 \oplus y_1 \oplus z_1$ which is clearly impossible. ◀

[6] If $x_1 = w_a$, say, we don't necessarily get the probability g^{-1}. In this case the probability is $\mathbb{P}(h_1(w_b) = h_1(w_c) \,|\, A)$ and the event $\{h(w_b) = h(w_c)\}$ might actually be included in A in which case the probability is 1. This can of course only happen if the keys w_b and w_c are *adjacent* in the trident so we could impose even further restrictions on the dependencies in S_2.

3.1 Proving Theorem 8

Now we will explain how to prove Theorem 8 proceeding much like we did for Theorem 3. Let us say that a d-uniform hypergraph $G = (V, E)$ is **tight** if $|V| \leq (d-1)|E| - 1$. With this terminology Theorem 8 states that the probability that the hash graph contains a tight subgraph of size $O(\log \log n)$ is at most $n^{-1/3+o(1)}$. It clearly suffices to bound the probability of the existence of a *connected* tight subgraph of size $O(\log \log n)$.

We start with the following two lemmas. The counterparts in the proof of Theorem 3 are the bounds on the probability of respectively an independent double cycle and an independent lasso with a dependent key attached.

▶ **Lemma 10.** *Let A_1 denote the event that the hash graph contains a tight subgraph $G = (V, E)$ of size $O(\log \log n)$ consisting of independent keys. Then $\mathbb{P}(A_1) \leq n^{-1+o(1)}$.*

Proof. Let $\ell = |E|$ be fixed. The number of ways to choose the keys of E is trivially bounded by m^ℓ and the number of ways to choose the set of nodes V in the hash graph is $\binom{n}{(d-1)\ell-1}$. For such a choice of nodes let a_i denote the number of nodes of V in the i'th group. The probability that one of the keys hash to V is then

$$\prod_{i=1}^{d} \frac{da_i}{n} \leq \left(\frac{a_1 + \cdots + a_d}{n}\right)^d \leq \left(\frac{d\ell}{n}\right)^d.$$

By the independence of the keys and a union bound we thus have that

$$\mathbb{P}(A_1) \leq \sum_{\ell=2}^{O(\log \log n)} m^\ell \binom{n}{(d-1)\ell - 1} \left(\frac{d\ell}{n}\right)^{d\ell} \leq \sum_{\ell=2}^{O(\log \log n)} \frac{1}{n} \left(\frac{m}{n}\right)^\ell (d\ell)^{d\ell} = n^{-1+o(1)},$$

as desired. ◀

▶ **Lemma 11.** *Let A_2 be the event that the hash graph contains a subgraph $G = (V, E)$ with $|V| \leq (d-1)|E|$ and $|E| = O(\log \log n)$ such that the keys of E are independent but such that there exists a key $y \notin E$ dependent on the keys of E. Then $\mathbb{P}(A_2) \leq n^{-1+o(1)}$.*

Proof. Let $|E| = \ell$ be fixed and write $E = \{x_1, \ldots, x_\ell\}$. We want to bound the number of ways to choose the keys of E. By Lemma 6, $y = \bigoplus_{i \in I} x_i$ for some $I \subset \{1, \ldots, \ell\}$ with $|I| = r$ for some odd $r \geq 3$. Let r be fixed for now. Using Lemma 7, we see that the number of ways to choose the keys of E is no more than $(r!!)^c m^{\frac{r+1}{2}} m^{\ell - r}$. For fixed ℓ and r the probability is thus bounded by

$$(r!!)^c m^{\ell - \frac{r-1}{2}} \binom{n}{\ell(d-1)} \left(\frac{d\ell}{n}\right)^{d\ell} = n^{-1+o(1)}$$

and a union bound over all $\ell = O(\log \log n)$ and $r \leq \ell$ suffices. ◀

We now generalise the notion of a double cycle starting with the following definition.

▶ **Definition 12.** Let $G = (V, E)$ be a d-uniform hypergraph. We say that a sequence of edges $P = (e_1, \ldots, e_t)$ of G is a **path** if $|e_i \cap e_{i+1}| = 1$ for $1 \leq i \leq t-1$ and $e_i \cap e_j = \emptyset$ when $i < j - 1$.

We say that $C = (e_1, \ldots, e_t)$ is a **cycle** if $t \geq 3$, $|e_i \cap e_{i+1}| = 1$ for all $i \pmod t$ and $e_i \cap e_j = \emptyset$ when $i \neq j \pm 1 \pmod t$.

Next comes the natural extension of the definition of double cycles to d-uniform hypergraphs.

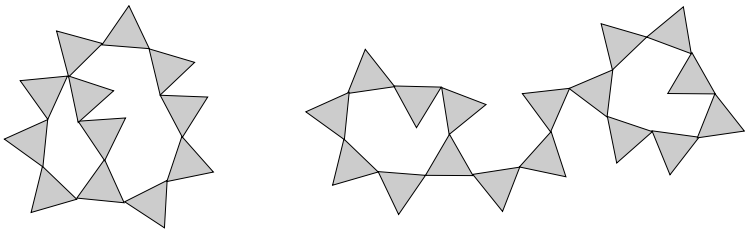


Figure 3 Double cycles in the case $d = 3$. The triangles represent edges of the graph and the corners represent the vertices.

▶ **Definition 13.** A d-uniform hypergraph G is called a **double cycle** if it has either of the following forms (see Figure 3).

- **D1:** It consists of of two vertex disjoint cycles C_1 and C_2 connected by a path $P = (x_1, \ldots, x_t)$ such that $|x_1 \cap V(C_1)| = |x_t \cap V(C_2)| = 1$ and $x_{i+1} \cap V(C_1) = x_i \cap V(C_2) = \emptyset$ for $1 \leq i \leq t-1$. We also allow P to have zero length and $|V(C_1) \cap V(C_2)| = 1$.
- **D2:** It consist of a cycle C and a path $P = (x_1, \ldots, x_t)$ of length $t \geq 2$ such that $|x_1 \cap V(C)| = |x_t \cap V(C)| = 1$ and $x_i \cap V(C) = \emptyset$ for $2 \leq i \leq t-1$. We also allow $t = 1$ and $|x_1 \cap C| = 2$.

Note that a double cycle always has $|V| = (d-1)|E| - 1$.

Now assume that the hash graph contains a connected tight subgraph $G = (V, E)$ of size $O(\log \log n)$ but that neither of the events of Lemma 10 and 11 has occurred. In particular no two edges e_1, e_2 of G has $|e_1 \cap e_2| \geq 2$ and no cycle consists of independent keys.

It is easy to check that under this assumption G contains at least two cycles. Now pick a cycle C_1 of least possible length. Since simple tabulation is 3-independent the cycle consists of at least 4 edges. If there exists an edge x not part of C_1 with $|x \cap V(C_1)| = 2$ we get a double cycle of type D_2. If $|x \cap V(C_1)| \geq 3$ we can use x to obtain a shorter cycle than C_1 which is a contradiction[7]. Using this observation we see that if there is a cycle $C_2 \neq C_1$ such that $|V(C_1) \cap V(C_2)| \geq 2$ then we can find a D_2 in the hash graph. Thus we may assume that any cycle $C_2 \neq C_1$ satisfies $|V(C_2) \cap V(C_1)| \leq 1$.

Now pick a cycle C_2 different from C_1 of least possible length. As before we may argue that any edge x not part of C_2 satisfies that $|x \cap V(C_2)| \leq 1$. Picking a shortest path connecting C_1 and C_2 (possibly the length is zero) gives a double cycle of type D_1.

Next we define tridents (see the non-grey part of Figure 4).

▶ **Definition 14.** We call a d-uniform hypergraph T a **trident** if it consists of paths $P_1 = (x_1, \ldots, x_{t_1})$, $P_2 = (y_1, \ldots, y_{t_2})$ and $P_3 = (z_1, \ldots, z_{t_3})$ of non-zero length such that either:

- There is a vertex v such that $x_{t_1} \cap y_{t_2} \cap z_{t_3} = \{v\}$, v is contained in no other edge of T and no vertex different from v is contained in more than one of the three paths.
- P_1, P_2 and $P_3 \backslash \{z_{t_3}\} = (z_2, \ldots, z_{t_3})$ are vertex disjoint and $(x_1, \ldots, x_{t_1}, z_{t_3}, y_{t_2}, \ldots, y_1)$ is a path.

Like in the proof of of Theorem 3 the existence of a double cycle not containing a cycle of independent keys implies the existence of the following structure (see Figure 4):

[7] Here we use that the length of C_1 is at least 4. If C_1 has length t the fact that x contains three nodes of C_1 only guarantees a cycle of length at most $3 + \lfloor \frac{t-3}{3} \rfloor$.

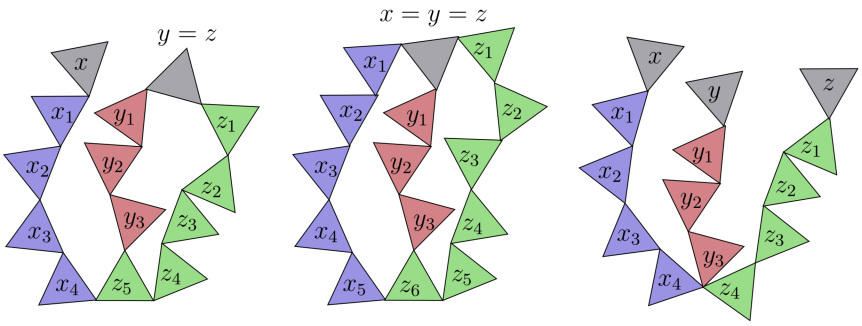


Figure 4 The case $d = 3$. Non-grey edges: Tridents. Grey edges: Keys that are each dependent on the set of non-black keys.

- **S1**: A trident consisting of three paths $P_1 = (x_1, \ldots, x_{t_1})$, $P_2 = (y_1, \ldots, y_{t_2})$ and $P_3 = (z_1, \ldots, z_{t_3})$ such that the keys of the trident are independent and such that there are, not necessarily distinct, keys x, y, z not in the trident extending the paths P_1, P_2 and P_3 away from their common meeting point such that x, y and z are each dependent on the keys in the trident.

We can bound the probability of this event almost identically to how we proceeded in the proof of Theorem 3. The only difference is that when making the ultimate reduction to the case where $x = y = z = x_1 \oplus y_1 \oplus z_1$ this event is in fact possible (see Figure 4). In this case however, there are three *different* hash function h_x, h_y and h_z such that $h_x(x_1) = h_x(x)$, $h_y(y_1) = h_y(x)$ and $h_z(z_1) = h_z(x)$. However, it is easy to bound the probability that this occur: The number of ways to choose the keys (x, x_1, y_1, z_1) is at most $3^c m^2$ by Lemma 7. The number of ways to choose the hash functions is upper bounded by d^3. Since the hash functions $h_1, \ldots, h_d$ are independent the probability that this can happen in the hash graph is by a union bound at most

$$d^3 3^c m^2 \left(\frac{d}{n}\right)^3 = O(n^{-1})$$

which suffices to complete the proof of Theorem 8.

Summing up

For now we have spent most of our energy proving Theorem 8. At this point it is perhaps not clear to the reader why it is important so let us again highlight the steps to Theorem 1. First of all let $k = \frac{\log\log n}{\log d} + r$ for r a sufficiently large constant. The steps are:

(1) Show that if some bin has load k then either the hash graph contains a tight subgraph of size $O(k)$ or a certain kind of witness tree T_k.
(2) Bound the probability that the hash graph contains a T_k by $O((\log\log n)^{-1})$.
(3) Bound the probability that the hash graph contains a tight subgraph of size $O(k)$ by $O((\log\log n)^{-1})$.

We can now cross (3) of the list. In fact, we have a much stronger bound than we require. The remaining steps as well as the final proofs of Theorem 1 and Theorem 2 are dealt with in the full version of this paper [1]. As already mentioned the proofs of all the above steps (except step (3)) are intricate but straightforward generalisations of the methods in [7].

References

1 Anders Aamand, Mathias Bæk Tejs Knudsen, and Mikkel Thorup. Power of d choices with simple tabulation. *CoRR*, abs/1804.09684, 2018. URL: `https://arxiv.org/abs/1804.09684`.

2 Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal of Computation*, 29(1):180–200, 1999. See also STOC'94.

3 Larry Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. See also STOC'77.

4 L. Elisa Celis, Omer Reingold, Gil Segev, and Udi Wieder. Balls and bins: Smaller hash families and faster evaluation. In *IEEE 52nd Symposium on Foundations of Computer Science*, FOCS, pages 599–608, 2011.

5 Xue Chen. Derandomized balanced allocation. *CoRR*, abs/1702.03375, 2017. Preprint. URL: `http://arxiv.org/abs/1702.03375`, `arXiv:1702.03375`.

6 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. Hashing for statistics over k-partitions. In *Proc. 56th Symposium on Foundations of Computer Science*, FOCS, pages 1292–1310, 2015.

7 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. The power of two choices with simple tabulation. In *Proc. 27. ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 1631–1642, 2016.

8 Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19–51, 1997. `doi:10.1006/jagm.1997.0873`.

9 Martin Dietzfelbinger and Philipp Woelfel. Almost random graphs with simple hash functions. In *Proc. 35th ACM Symposium on Theory of Computing*, STOC, pages 629–638, 2003. `doi:10.1145/780542.780634`.

10 Gaston H. Gonnet. Expected length of the longest probe sequence in hash code searching. *Journal of the ACM*, 28(2):289–304, 1981.

11 Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distribed Systems*, 12(10):1094–1104, 2001.

12 Michael Mitzenmacher, Andrea W. Richa, and Ramesh Sitaraman. The power of two random choices: A survey of techniques and results. *Handbook of Randomized Computing*, 1:255–312, 2001.

13 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

14 Rasmus Pagh and Flemming F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004. See also ESA'01.

15 Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation hashing. *Journal of the ACM*, 59(3):14:1–14:50, 2012. Announced at STOC'11.

16 Omer Reingold, Ron D. Rothblum, and Udi Wieder. Pseudorandom graphs in data structures. In *Proc. 41st International Colloquium on Automata, Languages and Programming*, ICALP, pages 943–954, 2014. `doi:10.1007/978-3-662-43948-7_78`.

17 Alan Siegel. On universal classes of extremely random constant-time hash functions. *SIAM Journal of Computing*, 33(3):505–543, 2004. See also FOCS'89.

18 Mikkel Thorup. Simple tabulation, fast expanders, double tabulation, and high independence. In *Proc. 54th Symposium on Foundations of Computer Science*, FOCS, pages 90–99, 2013.

19 Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM Journal of Computing*, 41(2):293–331, apr 2012. Announced at SODA'04 and ALENEX'10.

20 Berthold Vöcking. How asymmetry helps load balancing. *Journal of the ACM*, 50(4):568–589, 2003. See also FOCS'99.

21 Udi Wieder. Hashing, load balancing and multiple choice. *Foundations and Trends in Theoretical Computer Science*, 12(3-4):275–379, 2017. doi:10.1561/0400000070.

22 Philipp Woelfel. Asymmetric balanced allocation with simple hash functions. In *Proc. 17th ACM-SIAM Symposium on Discrete Algorithm*, SODA, pages 424–433, 2006.

23 Albert L. Zobrist. A new hashing method with application for game playing. Technical report, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1970.

One-Way Trail Orientations

Anders Aamand[1]
University of Copenhagen, Copenhagen, Denmark
andersaamand@hotmail.com
https://orcid.org/0000-0002-0402-0514

Niklas Hjuler[2]
University of Copenhagen, Copenhagen, Denmark
niklashjuler@gmail.com
https://orcid.org/0000-0002-0815-670X

Jacob Holm[3]
University of Copenhagen, Copenhagen, Denmark
jaho@di.ku.dk
https://orcid.org/0000-0001-6997-9251

Eva Rotenberg[4]
Technical University of Denmark, Lyngby, Denmark
erot@dtu.dk
https://orcid.org/0000-0001-5853-7909

Abstract

Given a graph, does there exist an orientation of the edges such that the resulting directed graph is strongly connected? Robbins' theorem [Robbins, Am. Math. Monthly, 1939] asserts that such an orientation exists if and only if the graph is 2-edge connected. A natural extension of this problem is the following: Suppose that the edges of the graph are partitioned into trails. Can the trails be oriented consistently such that the resulting directed graph is strongly connected?

We show that 2-edge connectivity is again a sufficient condition and we provide a linear time algorithm for finding such an orientation.

The generalised Robbins' theorem [Boesch, Am. Math. Monthly, 1980] for mixed multigraphs asserts that the undirected edges of a mixed multigraph can be oriented to make the resulting directed graph strongly connected exactly when the mixed graph is strongly connected and the underlying graph is bridgeless.

We consider the natural extension where the undirected edges of a mixed multigraph are partitioned into trails. It turns out that in this case the condition of the generalised Robbin's Theorem is not sufficient. However, we show that as long as each cut either contains at least 2 undirected edges or directed edges in both directions, there exists an orientation of the trails such that the resulting directed graph is strongly connected. Moreover, if the condition is satisfied, we may start by orienting an arbitrary trail in an arbitrary direction. Using this result one obtains a very simple polynomial time algorithm for finding a strong trail orientation if it exists, both in the undirected and the mixed setting.

[1] Research supported by Thorup's Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research and by his Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.
[2] This work is supported by the Innovation Fund Denmark through the DABAI project.
[3] Research supported by Thorup's Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research and by his Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.
[4] This research was conducted during the fourth author's time as a PhD student at University of Copenhagen.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 6; pp. 6:1–6:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Mathematics of computing → Paths and connectivity problems

Keywords and phrases Graph algorithms, Robbins' theorem, Graph orientation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.6

1 Introduction and motivation

Suppose that the mayor of a small town decides to make all the streets one-way such that it is possible to get from any place to any other place without violating the orientations of the streets[5]. If all the streets are initially two-way then Robbins' theorem [10] asserts that this can be done exactly when the corresponding graph is 2-edge connected. If, on the other hand some of the streets were already one-way in the beginning then the generalised Robbins' theorem by Boesch [1] states that it can be done exactly when the corresponding "mixed" graph is strongly connected and the underlying graph is bridgeless.

However, the proofs of both of these results assume that every street of the city corresponds to exactly one edge in the graph. This assumption hardly holds in any city in the world and therefore a more natural assumption is that every street corresponds to a trail (informally, a potentially self-crossing path) in the graph and that the edges of each trail must be oriented consistently[6].

In this paper we consider such graphs having their edges partitioned into trails. We prove that the trails can be oriented to make the resulting directed graph strongly connected exactly if the initial graph is 2-edge connected (note that this is precisely the condition of Robbins' theorem).

Not only do we show that the strong trail orientation problem in undirected 2-edge connected graphs always has a solution, we also provide a linear time algorithm for finding such an orientation. In doing so, we use an interesting combination of techniques that allow us to reduce to a graph with a number of 3-edge connected components that is linear in the number of edges. Using that the average size of these components is constant and that we can piece together solutions for the individual components we obtain an efficient algorithm.

Finally, we will consider the generalised Robbins' theorem in this new setting by allowing some edges to be oriented initially and supposing that the remaining edges are partitioned into trails. We will show that if each cut (V_1, V_2) in the graph has either at least 2 undirected edges going between V_1 and V_2 or at least 1 directed edge in each direction then it is possible to orient the trails making the resulting graph strongly connected. In fact, we show that if this condition is satisfied we may start by orienting an *arbitrary* trail in an *arbitrary* direction. Although this condition is not necessary it does give a simple algorithm for finding a trail orientation if it exists. Indeed, initially the graph may contain undirected edges that are *forced* in one direction by some cut. For finding a trail orientation if it exists we can thus orient forced trails in the forced direction. If there are no forced trails we orient any trail arbitrarily.

[5] The motivation for doing so is that the streets of the town are very narrow and thus it is a great hassle when two cars unexpectedly meet.

[6] This version of the problem was given to us through personal communication with Professor Robert E. Tarjan.

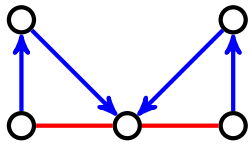


Figure 1 The graph is strongly connected and the underlying graph is 2-edge connected, but irrespective of the choice of orientation of the red trail, the graph will no longer be strongly connected.

Note that in the mixed setting the feasibility depends on the trail decomposition which is not the case for the other results. That the condition from the generalised Robbins' theorem is not sufficient can be seen from Figure 1.

Earlier methods

Several methods have already been applied for solving orientation problems in graphs where the goal is to make the resulting graph strongly connected.

One approach used by Robbins [10] is to use that a 2-edge connected graph has an *ear-decomposition*. An ear decomposition of a graph is a partition of the set of edges into a cycle C and paths $P_1, \ldots, P_t$ such that P_i has its two endpoints but none of its internal vertices on $C \cup \left(\bigcup_{j=1}^{i-1} P_j\right)$. Given the existence of an ear decomposition of a 2-edge connected graph it is easy to prove Robbins' theorem. Indeed, any choice of consistent orientations of the paths and the cycle gives a strongly connected graph.

A second approach introduced by Tarjan [4] gives another simple proof of Robbins' theorem. One can create a DFS tree in the graph rooted at a vertex v and orient all edges in the DFS tree away from v. The remaining edges are all back edges (see [4]) and are oriented towards v. It is easily verified that this gives a strong orientation if the graph is 2-edge connected. A similar approach was used by Chung et al. [2] in the context of the generalized Robbins' theorem for mixed multigraphs.

The above methods not only prove Robbins' theorem, they also provide linear time algorithms for finding strong orientations of undirected or mixed multigraphs.

However, none of the above methods have proven fruitful in our case. In case of the ear decomposition we would need one that is somehow compatible with the partitioning into trails, and this seems hard to guarantee. Similar problems appear when trying a DFS-approach. Neither does the proof by Boesch [1] of Robbins' theorem for mixed multigraphs generalise to prove our result. Most importantly, the corresponding theorem would no longer be true for trail orientations as is shown by the example in Figure 1.

Since the classical linear time algorithms rely on ear-decompositions and DFS searches, and since these approaches do not immediately work for trail partitions, our linear time algorithm will be a completely new approach to solving orientation problems.

Structure of the paper

The structure of this paper is as follows. In section 3 we prove our generalisation of Robbins' theorem for undirected graphs partitioned into trails. In section 4 we study the case of mixed graphs. Finally in section 5 we provide our linear time algorithm for trail orientation in an undirected graph.

2 Preliminaries

Let us briefly review the concepts from graph theory that we will need.

A graph having some subset of its edges oriented is said to be a *mixed* graph. We will write $\{u, v\}$ for an undirected edge between u and v and (u, v) for an edge directed from u to v.

A *walk* in a graph is an alternating sequence of vertices and edges $v_0, e_1, v_1, e_2, \ldots, v_k$, such that for $1 \leq i \leq k$ the edge e_i has v_{i-1} and v_i as its two endpoints. In a directed or mixed graph we further require that either e_i be undirected or directed from v_{i-1} to v_i

A *trail* is a walk without repeated edges. A *path* is a trail without repeated vertices (except possibly $v_0 = v_k$). Finally, a *cycle* is a path for which $v_0 = v_k$.

Next, a mixed multigraph $G = (V, E)$ is called *strongly connected* if for each pair of vertices $u, v \in V$ there exists a walk from u to v. In case the graph is undirected this is equivalent to saying that it consists of exactly one connected component. If $A \subseteq V$ we will say that A is *strongly connected in* G if for each pair of vertices $u, v \in A$ there is a walk in G from u to v.

A *cut* or *edge-cut* (V_1, V_2) in a graph is a partition of its vertices into two non-empty subsets V_1, V_2. We recall the definition of k-edge connectivity. A graph $G = (V, E)$ is said to be *k-edge connected* if and only if $G' = (V, E - X)$ is connected for all $X \subseteq E$ where $|X| < k$. A trivially equivalent condition is that each cut (V_1, V_2) in the graph has at least k edges going between V_1 and V_2.

Finally, if $G = (V, E)$ is a mixed multigraph and $A \subseteq V$ we define G/A to be the graph obtained by contracting A to a single vertex (maintaining duplicate edges and self-loops) and $G[A]$ to be the subgraph of G induced by A. The following simple observation will be used repeatedly in this paper.

▶ **Observation 2.1.** *If $G = (V, E)$ is k-edge connected and $A \subseteq V$ then G/A is k-edge connected. Also, if G is a strongly connected mixed multigraph then G/A is too.*

3 Robbins Theorem Revisited

We are now ready to state and prove our generalisation of Robbins' theorem.

▶ **Theorem 3.1.** *Let $G = (V, E)$ be an undirected multigraph with E partitioned into trails. An orientation of each trail such that the resulting directed graph is strongly connected exists if and only if G is 2-edge connected.*

We note that this theorem could also be proven using a general result from Király and Szigeti [6] which relies on theorems by Nash-Willams [9]. However, our proof is significantly simpler (in fact we believe that restating the theorem by Király and Szigeti and explaining the reduction would be more cumbersome) and more suitable for constructing algorithms.

Proof. If G is not 2-edge connected, such an orientation obviously doesn't exist, so we only need to prove the converse. Suppose therefore that G is 2-edge connected.

Our proof is by induction on the number of edges in G. If there are no edges, the graph consists of a single vertex, and the statement is obviously true. Assume now the statement holds for all graphs with strictly fewer edges than G. Pick an arbitrary edge e that sits at the end of its corresponding trail.

If $G - e$ is 2-edge connected, then by the induction hypothesis there is a strong orientation of $G - e$ that respects the trails of G. Such an orientation clearly extends to the required orientation of G.

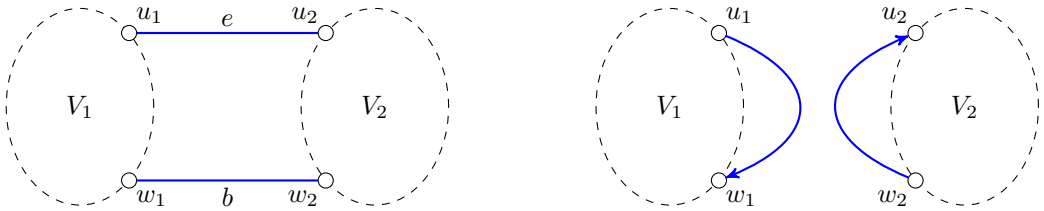


Figure 2 A 2-edge cut and the two graphs $G_1 = G[V_1] \cup \{\{u_1, w_1\}\}$ and $G_2 = G[V_2] \cup \{\{u_2, w_2\}\}$. The orientations of the two new edges are obtained from the strong trail orientations of G_1 and G_2.

If $G - e$ is not 2-edge connected, there exists a bridge b in $G - e$ (see Figure 2). Let V_1, V_2 be the two connected components of $G - \{e, b\}$, and let $e = \{u_1, u_2\}$ and $b = \{w_1, w_2\}$ such that for $i \in \{1, 2\}$, $u_i, w_i \in V_i$ (note that we don't necessarily have that u_i and w_i are distinct for $i \in \{1, 2\}$).

Now for $i \in \{1, 2\}$ construct the graph $G_i = G[V_i] \cup \{\{u_i, w_i\}\}$ (note that $\{u_i, w_i\}$ might be a self-loop but this causes no problems for the argument), and define the trails in G_i to be the trails of G that are completely contained in G_i, together with a single trail combined from the (possibly empty) partial trail of e contained in G_i and ending at u_i, followed by the edge $\{u_i, w_i\}$, followed by the (possibly empty) partial trail of b contained in G_i starting at w_i. Both G_1 and G_2 are 2-edge connected since they can each be obtained as contractions of G with some self-loops deleted. Furthermore, they each have strictly fewer edges than G, so inductively each has a strong orientation that respects the given trails. Further, we can assume that the orientations are such that the new edges are oriented (u_1, w_1) and (w_2, u_2) by flipping the orientation of all edges in either graph if necessary. We claim that this orientation, together with e oriented as (u_1, u_2) and b oriented as (w_2, w_1), is the required orientation of G. To see this first note that (by our choice of flips) this orientation respects the trails. Secondly, suppose $v_1 \in V_1$ and $v_2 \in V_2$ are arbitrary. Since G_1 is strongly connected $G[V_1]$ contains a directed path from v_1 to u_1. Similarly, $G[V_2]$ contains a directed path from u_2 to v_2. Thus G contains a directed path from v_1 to v_2. A similar argument gives a directed path from v_2 to v_1 and since v_1 and v_2 were arbitrary this proves that G is strongly connected and our induction is complete. ◀

The construction in the proof can be interpreted as a naive algorithm for finding the required orientation when it exists.

▶ **Corollary 3.2.** *The one-way trail orientation problem on a graph with n vertices and m edges can be solved in $\mathcal{O}(n + m \cdot f(m, n))$ time, where $f(m, n)$ is the time per operation for fully dynamic bridge finding (a.k.a. 2-edge connectivity).*

At the time of this writing [3], this is $\mathcal{O}(n + m(\log n \log\log n)^2)$. In Section 5 we will provide a less naive algorithm which runs in linear time.

4 Extension to Mixed graphs

Now we will extend our result to the case of mixed graphs. We are going to prove the following.

▶ **Theorem 4.1.** *Let $G = (V, E)$ be a strongly connected mixed multigraph. Then $G - e$ is strongly connected for all undirected $e \in E$ if and only if for any partition $\mathcal{P}$ of the undirected edges of G into trails, and any $T \in \mathcal{P}$, any orientation of T can be extended to a strong trail orientation of $(G, \mathcal{P})$.*

Algorithm 1: Algorithm for mixed graphs.

Input: A mixed multigraph G and a partition $\mathcal{P}$ of the undirected edges of G into trails.

Output: True if $(G, \mathcal{P})$ has a strong trail orientation, otherwise false. If G has a bridge or is not strongly connected, G is left unmodified. Otherwise G is modified, either to have such a strong trail orientation, or to a forced graph that is not strongly connected.

```
if G has a bridge or is not strongly connected then
    return false
end
while |P| > 0 do
    if for some undirected edge e, G − e is not strongly connected then
        Let T ∈ P be the trail containing e.
        if some orientation of T leaves G strongly connected then
            Apply such an orientation of T to G
        else
            return false
        end
    else
        Let T ∈ P be arbitrary.
        Update G by orienting T in an arbitrary direction.
    end
    Remove T from P.
end
return true
```

Suppose $G = (V, E)$ is as in the theorem. We will say that $e \in E$ is *forced* if it is undirected and satisfies that $G - e$ is not strongly connected. This terminology is natural as it is equivalent to saying that there exists a cut (V_1, V_2) in G such that e is the only undirected edge in this cut and such that all the directed edges go from V_1 to V_2. If we want an orientation of the trails making the graph strongly connected we are clearly forced to orient e from V_2 to V_1.

Theorem 4.1 is a proper extension of Theorem 3.1 since if G is undirected and 2-edge connected then no $e \in E$ is forced. Furthermore, the theorem suggests a very simple polynomial time algorithm (see Algorithm 1) for finding a strong orientation of the trails if it exists. Indeed, if the mixed graph contains forced edges we direct the corresponding trails in the forced direction. If there are no forced edges then either the graph is no longer strongly connected in which case we know that a strong trail orientation doesn't exist. Otherwise, we may by Theorem 4.1 orient any trail in an arbitrary direction.

For proving Theorem 4.1 we will need the following lemma.

▶ **Lemma 4.2.** *Let G be a directed graph, and let (A, B) be a cut with exactly one edge crossing from A to B. Then G is strongly connected if and only if G/A and G/B are.*

Proof. Strong connectivity is preserved by contractions, so if G is strongly connected then G/A and G/B both are. For the other direction, let (a_1, b_1) be the edge going from A to B. As G/A is strongly connected and (a_1, b_1) is the only edge going from A to B we can for any edge (b_2, a_2) going from B to A find a path from b_1 to b_2 that stays in B. Since G/B is

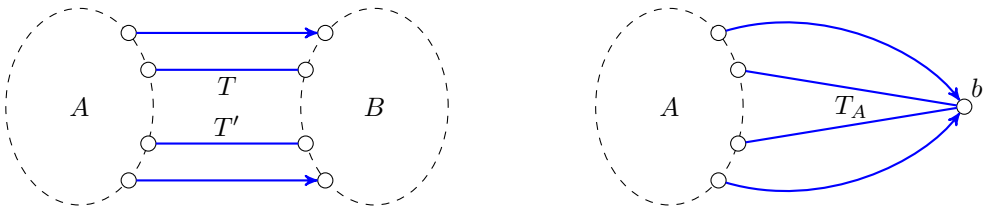


Figure 3 A cut with two undirected edges and all directed edges going from A to B followed by a contraction of B.

strongly connected, it follows that A is strongly connected in G. By a symmetric argument, B is also strongly connected in G and since the cut has edges in both directions (as e.g. G/A is strongly connected), G must be strongly connected. ◀

Now we provide the proof of Theorem 4.1.

Proof of Theorem 4.1. If there exists an undirected edge e such that $G - e$ is not strongly connected, then the trail T containing e can at most be directed one way since e is forced, so there is an orientation of T that does not extend to a strong trail orientation of $(G, \mathcal{P})$. To prove the converse suppose $G - e$ is strongly connected for all undirected $e \in E$.

The proof is by induction on $|\mathcal{P}|$. If $|\mathcal{P}| = 0$ the result is trivial. So suppose $|\mathcal{P}| \geq 1$ and that the theorem holds for all $(G', \mathcal{P}')$ with $|\mathcal{P}'| < |\mathcal{P}|$.

Consider the chosen trail $T \in \mathcal{P}$. If both orientations of T leave a graph where the condition in the theorem is still satisfied we are home by induction. Otherwise, there must exist a cut (A, B) of the following form: (1) T crosses the cut exactly once, (2) exactly one undirected edge from a different undirected trail $T' \in \mathcal{P}$ crosses the cut and (3) every directed edge crossing the cut goes from A to B.

Now suppose there is such a cut (A, B) (see Figure 3). Consider the graph G/B and let b be the node corresponding to B in G/B. Let $\mathcal{P}_A$ consist of all trails in $\mathcal{P}$ that are completely contained in A, together with a single trail T_A combined from the (possibly empty) fragments of T and T', joined at b. Since any cut in G/B corresponds to a cut in G, G/B is strongly connected and remains so after deletion of any single undirected edge. By construction $|\mathcal{P}_A| \leq |\mathcal{P}| - 1$, so by induction any orientation of T_A in G/B extends to a strong orientation of $(G/B, \mathcal{P}_A)$. Let G/A, a, $\mathcal{P}_\mathcal{B}$ and T_B be defined symmetrically, then by the same argument any orientation of T_B in G/A extends to a strong orientation of $(G/A, \mathcal{P}_B)$. Now for any orientation of T, we can choose orientations of T_A and T_B that are compatible. The result then follows by Lemma 4.2. ◀

Theorem 4.1 gives a sufficient condition for the existence of a strong orientation and we deal with the other cases by first orienting all forced edges. However, the generalised Robbins' theorem provides a simple equivalent condition, which we lack. Finding such an equivalent condition in our setting is an essential open problem for strong trail orientations. As seen by the example of Figure 1 such a condition will necessarily have to depend on the structure of the trail partition.

5 Linear time algorithm

In this section we provide our linear time algorithm for solving the trail orientation problem in undirected graphs. For this, we make two crucial observations. First, we show that there is an easy linear time reduction from general graphs or multigraphs to cubic multigraphs.

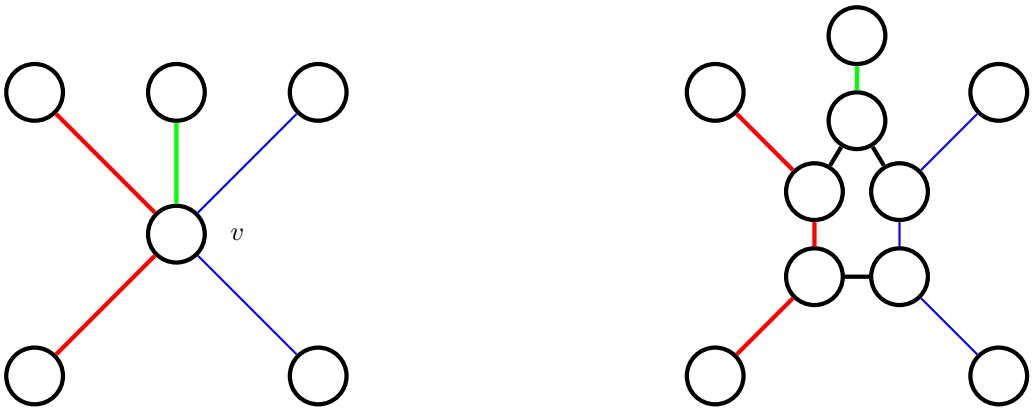


Figure 4 A node of degree 5 turns into a cycle of length 5.

Second, we show that in a cubic multigraph with n vertices, we can in linear time find and delete a set of edges that are at the end of their trails, such that the resulting graph has $\Omega(n)$ 3-edge connected components. We further show that we can compute the required orientation recursively from an orientation of each 3-edge connected component together with the cactus graph of 3-edge connected components. Since the average size of these components is constant, we can compute the orientations of most of them in constant time individually and thus in linear time taken together. The rest contains at most a constant fraction of the vertices, and so a simple geometric sum argument tells us that the total time is also linear.

We start out by making the following reduction.

▶ **Lemma 5.1.** *The one-way trail problem on a 2-edge connected graph or multigraph with n vertices and m edges, reduces in $\mathcal{O}(m+n)$ time to the same problem on a 2-edge connected cubic multigraph with $2m$ vertices and $3m$ edges.*

Proof. Cyclically order the edges adjacent to each vertex such that two edges that are adjacent on the same trail are consecutive in the order. Replace each single vertex v with a cycle of length $\deg(v)$, with each vertex of the new cycle inheriting a corresponding neighbour of v such that the order of the vertices on the cycle corresponds to the cyclic ordering (see Figure 4). Note that for a vertex of degree 2, this creates a pair of parallel edges, so the result may be a multigraph. By the choice of cyclic ordering, we can make the cycle-edge between the two vertices on the same trail belong to that trail. The rest of the cycle edges form new length 1 trails. Clearly the new graph is also 2-edge connected so by Theorem 3.1 it has a strong trail orientation, and any strong trail orientation on this graph translates to a strong trail orientation of the original graph. The new graph has exactly $2m$ vertices and $3m$ edges, and is constructed in $\mathcal{O}(m+n)$ time. ◀

Recall now that a multigraph C is called a *cactus* if it is connected and each edge is contained in at most one cycle. If G is any connected graph we let $C_1, \ldots, C_k$ be its 3-edge connected components. It is well known that if we contract each of these we obtain a cactus graph. For a proof of this result see section 2.3.5 of [8]. As the cuts in a contracted graph are also cuts in the original graph we have that if G is 2-edge connected then the cactus graph is 2-edge connected. The edges of the cactus are exactly the edges of G which are part of a 2-edge cut. We will call these edges *2-edge critical*.

It is easy to check that if a cactus has m edges and n vertices then $m \leq 2(n-1)$. We will be using this result in the proof of the following lemma.

▶ **Lemma 5.2.** *Let $G = (V, E)$ be a cubic 2-edge connected multigraph, let $X \subseteq E$, and let $F \subseteq E$ with $F \supseteq E \setminus X$ minimal (w.r.t inclusion) such that $H = (V, F)$ is 2-edge connected. Then H has at least $\frac{2}{5}|X|$ distinct 3-edge connected components.*

Proof. Let $X_{\text{del}} = X \setminus F$ be the set of edges deleted from G to obtain H, and let $X_{\text{keep}} = X \setminus X_{\text{del}}$ be the remaining edges in X.

By minimality of H there are at least $|X_{\text{keep}}|$ 2-edge-critical edges in H i.e. edges of the corresponding cactus, and thus, if $|X_{\text{keep}}| \geq \frac{4}{5}|X|$, there are at least $\frac{1}{2}|X_{\text{keep}}| + 1 \geq \frac{2}{5}|X| + 1$ distinct 3-edge connected components.

If $|X_{\text{keep}}| \leq \frac{4}{5}|X|$ then $|X_{\text{del}}| \geq \frac{1}{5}|X|$, and since G is cubic and the removal of each edge creates two vertices of degree 2 we must have that H has at least $2|X_{\text{del}}| \geq \frac{2}{5}|X|$ distinct 3-edge connected components. ◀

▶ **Lemma 5.3.** *Let $G = (V, E)$ be a connected cubic multigraph with E partitioned into trails. Then G has a spanning tree that contains all edges that are not at the end of their trail.*

Proof. Let F be the set of edges that are not at the end of their trail. Since G is cubic, the graph (V, F) is a collection of vertex-disjoint paths, and in particular it is acyclic. Since G is connected, F can be extended to a spanning tree. ◀

Note that we can find this spanning tree in linear time e.g. by contracting all edges internal to a trail, finding a spanning tree of the resulting graph, and adding the internal trail edges to the edges of this spanning tree.

▶ **Lemma 5.4.** *Let $G = (V, E)$ be a cubic 2-edge connected multigraph with E partitioned into trails. Let T be a spanning tree of G containing all edges that are not at the end of their trail. Let H be a minimal subgraph of G (w.r.t inclusion) that contains T and is 2-edge connected. Then for any $k \geq 5$, less than $\frac{4}{5}\frac{k}{k-1}|V|$ of the vertices in H are in a 3-edge connected component with at least k vertices.*

Proof. Let X be the set of edges that are not in T. Since G is cubic, $|X| = \frac{1}{2}|V| + 1$. By Lemma 5.2 H has at least $\frac{2}{5}|X| > \frac{1}{5}|V|$ 3-edge connected components. Each such component contains at least one vertex, so the total number of vertices in components of size at least k is less than $\frac{k}{k-1}\left(|V| - \frac{1}{5}|V|\right) = \frac{4}{5}\frac{k}{k-1}|V|$. ◀

▶ **Definition 5.5.** Let C be a 3-edge connected component of some 2-edge connected graph H, whose edges are partitioned into trails. Define $\Gamma_H(C)$ to be the 3-edge connected graph obtained from C by inserting a new edge $\{e_1, f_1\}$ for each min-cut $\{e, f\}$ where $e = \{e_1, e_2\}$ and $f = \{f_1, f_2\}$ and $e_1, f_1 \in C$. Define the corresponding partition of the edges of $\Gamma_H(C)$ into trails by taking every trail that is completely contained in C, together with new trails combined from the fragments of the trails that were broken by the min-cuts together with the new edges that replaced them. See Figure 5.

At this point the idea of the algorithm can be explained. We remove as many of the edges that sit at the end of their trails as possible, while maintaining that the graph is 2-edge connected. Lemma 5.4 guarantees that we obtain a graph H with $\Omega(|V|)$ many 3-edge connected components of size $O(1)$. We solve the problem for each $\Gamma_H(C)$ for every 3-edge connected component. Finally, we combine the solutions for the different components like in the proof of Theorem 3.1.

▶ **Theorem 5.6.** *The one-way trail orientation problem can be solved in $\mathcal{O}(m + n)$ time on any 2-edge connected undirected graph or multigraph with n vertices and m edges.*

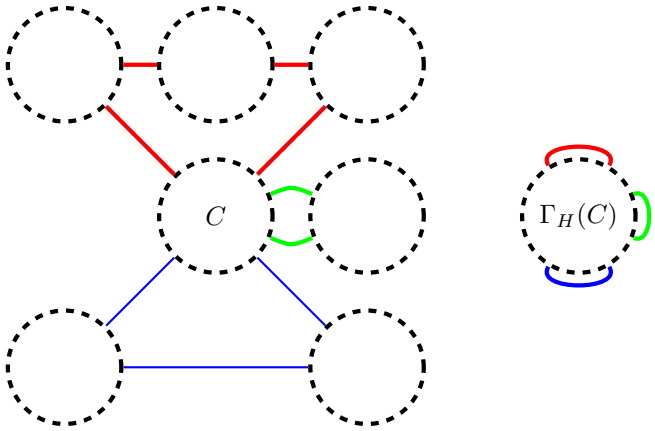


Figure 5 The 3-edge connected components of a 2-edge connected graph. Notice that every edge leaving a 3-edge connected component C becomes part of a cycle if all 3-edge components are contracted. The right hand side shows $\Gamma_H(C)$ where C is the component in the middle.

Proof. By Lemma 5.1, we can assume the graph is cubic. For the algorithm we will use two subroutines. First of all, when we have found a minimum spanning tree T containing the edges that are not on the end of their trail we can use the algorithm of Kelsen et al. [5] to, in linear time, find a minimal (w.r.t. inclusion) subgraph H of G that contains T and is 2-edge connected. Secondly, we will use the algorithm by Mehlhorn et al. [7] to, in linear time, build the cactus graph of 3-edge connected components. The algorithm runs as follows:

1. Construct a spanning tree T of G that contains all edges that are not at the end of their trail.
2. Construct a minimal subgraph H of G that contains T and is 2-edge connected[7].
3. Find the cactus of 3-edge connected components of[8] H.
4. For each 3-edge connected component C_i, construct $\Gamma_H(C_i)$.
5. Recursively compute an orientation for each[9] $\Gamma_H(C_i)$.
6. Combine the orientations from each component to a strong trail orientation of H. A such is also a strong trail orientation of G.

First we will show correctness and then we will determine the running time.

Recall that we can flip the orientation in each $\Gamma_H(C_i)$ and still obtain a strongly connected graph respecting the trails in $\Gamma_H(C_i)$. The way we construct the orientation of the edges of G is by flipping the orientation of each $\Gamma_H(C_i)$ in such a way that each cycle in the cactus graph becomes a directed cycle[10]. This can be done exactly because no edge of the cactus is contained in two cycles. By construction this orientation respects the trails so we need to argue that it gives a strongly connected graph.

For showing that the resulting graph is strongly connected, consider the graph in which every 3-edge connected component is contracted to a single point. This is exactly the cactus of 3-edge connected component of G which is strongly connected as the cycles of the cactus

[7] See Kelsen [5].
[8] See Mehlhorn [7].
[9] Note that $\Gamma_H(C_i)$ is cubic unless it consists of exactly one node. In this case however we don't need to do anything.
[10] In practice this is done by making a DFS (or any other search tree one likes) of the cactus and repeatedly orienting each component in a way consistent with the previous ones.

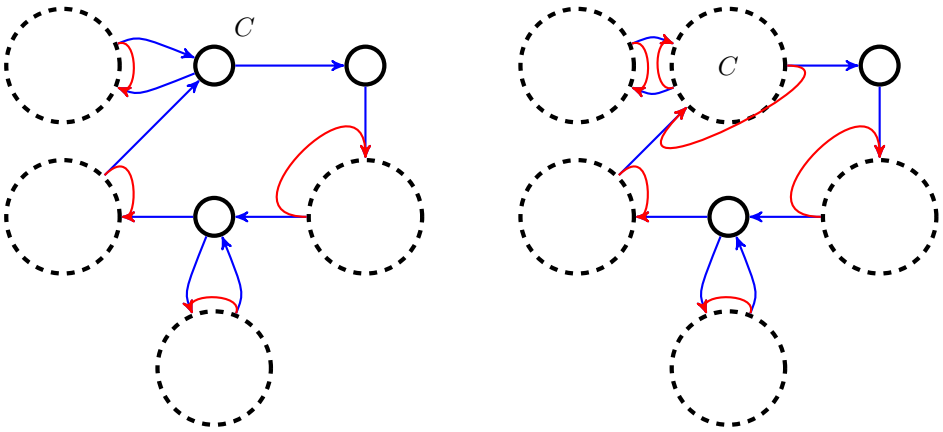


Figure 6 Before and after uncontracting component C. The blue edges are the edges of the cactus i.e. the 2-edge critical edges of H. The red edges are the ones obtained from the 2-edge cuts of H as described in the construction of the $\Gamma_H(C_i)$.

graph have become directed cycles. Now assume inductively that we have uncontracted some of the 3-edge connected components obtaining a graph G_1 which is strongly connected. We then uncontract another component C (see Figure 6) and obtain a new graph G_2 which we will show is strongly connected. If $u, v \in C$, then since $\Gamma_H(C)$ is strongly connected there is a path from u to v in $\Gamma_H(C)$. If this path only contains edges which are edges of C it will also exist in G_2. If the path uses one of the added (now oriented) edges (e_1, f_1), it is because there are edges (e_1, e_2) and (f_2, f_1) forming a cut and thus being part of a cycle in the cactus. In this case we use edge (e_1, e_2) to leave component C and then go from e_2 back to component C which is possible since G_1 was strongly connected. When we get back to the component C we must arrive at f_1 since otherwise there would be two cycles in the cactus containing the edge (e_1, e_2). Hence we succeeded in disposing of the edge (e_1, f_1) with a directed path in G_2. This argument can be used for any of the edges of $\Gamma_H(C)$ that are not in C and thus C is strongly connected in G_2. Since G_1 was strongly connected this suffices to show that G_2 is strongly connected. By induction this implies that after uncontracting all components the resulting graph is strongly connected.

Now for the running time. By Lemma 5.4 each level of recursion reduces the number of vertices in "large" components by a constant fraction, for instance for $k = 10$ we reduce the number of vertices in components of size at least 10 by a factor of $\frac{8}{9}$. Let $f(n)$ be the worst case running time with n nodes for a cubic graph, and pick c large enough such that cn is larger than the time it takes to go through steps 1-4 and 6 as well as computing the orientations in the "small" components. This includes the linear time needed to construct the new set of trails (in 4), and the linear time to reassemble the directed trails (in 6). Let $a_1, \ldots, a_k$ be the number of vertices in the "large" 3-edge connected components. Then $\sum_i a_i \leq \frac{8n}{9}$ and

$$f(n) \leq cn + \sum_i f(a_i).$$

Inductively, we may assume that $f(a_i) \leq 9ca_i$ and thus obtain

$$f(n) \leq cn + \sum_i f(a_i) \leq cn + \sum_i 9ca_i = cn + 8cn = 9cn$$

proving that $f(n) \leq 9cn$ for all n. ◀

Algorithm 2: Linear time algorithm for cubic graphs.

Input: A 2-edge connected undirected cubic multigraph G and a partition $\mathcal{P}$ of the edges of G into trails.

Output: G is modified to a strong trail orientation of $(G, \mathcal{P})$.

```
Construct a spanning tree T of G that contains all edges that are not at the end of
 their trail.
Construct a minimal subgraph H of G that contains T and is 2-edge connected.
Find the cactus C of 3-edge connected components of H.
for each 3-edge connected component C_i in C in DFS preorder do
    Construct G_i = Γ_H(C_i).
    Recursively compute an orientation for G_i.
    if the orientation of G_i is not compatible with its DFS parent then
        Flip orientation of G_i
    end
end
for each edge e deleted from G to create H do
    if no edge on the trail of e has been oriented yet then
        Pick an arbitrary orientation for e.
    else
        Set the orientation of e to follow the trail.
    end
end
```

6 Open problems

We here mention two problems concerning trail orientations which remain open.

First of all, our linear time algorithm for finding trail orientations only works for undirected graphs and it doesn't seem to generalise to the trail orientation problem for mixed graphs. It would be interesting to know whether there also exists a linear time algorithm working for mixed graphs. If so it would complete the picture of how fast an algorithm we can obtain for any variant of the trail orientation problem.

Secondly, our sufficient condition for when it is possible to solve the trail orientation problem for mixed multigraphs is clearly not necessary. It would be interesting to know whether there is a simple necessary and sufficient condition like there is in the undirected case. Since in the mixed case the answer to the problem actually depends on the given trail decomposition and not just on the structure of the mixed graph it is harder to provide such a condition. One can however give the following condition. It is possible to orient the trails making the resulting graph strongly connected if and only if when we repeatedly direct the forced trails end up with a graph satisfying our condition in Theorem 4.1. This condition is not simple and is not easy to check directly. Is there a more natural condition?

References

1 Frank Boesch and Ralph Tindell. Robbins's theorem for mixed multigraphs. *The American Mathematical Monthly*, 87(9):716–719, 1980. URL: `http://www.jstor.org/stable/2321858`.

2 Fan R. K. Chung, Michael R. Garey, and Robert E. Tarjan. Strongly connected orientations of mixed multigraphs. *Networks*, 15(4):477–484, 1985. `doi:10.1002/net.3230150409`.

3 Jacob Holm, Eva Rotenberg, and Mikkel Thorup. Dynamic bridge-finding in $\tilde{O}(\log 2\ n)$ amortized time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 35–52, 2018. URL: http://dl.acm.org/citation.cfm?id=3174304.3175269.

4 John Hopcroft and Robert Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, 1973. doi:10.1145/362248.362272.

5 Pierre Kelsen and Vijaya Ramachandran. On finding minimal 2-connected subgraphs. In *Proceedings of the Second Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 28-30 January 1991, San Francisco, California.*, pages 178–187, 1991. URL: http://dl.acm.org/citation.cfm?id=127787.127824.

6 Zoltán Király and Zoltán Szigeti. Simultaneous well-balanced orientations of graphs. *J. Comb. Theory, Ser. B*, 96(5):684–692, 2006.

7 Kurt Mehlhorn, Adrian Neumann, and Jens M. Schmidt. Certifying 3-edge-connectivity. *Algorithmica*, 77(2):309–335, 2017. doi:10.1007/s00453-015-0075-x.

8 Hiroshi Nagamochi and Toshihide Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.

9 J. A. Nash-Willams. On orientations, connectivity and odd-vertex-pairings in finite graphs. *Canad. J. Math.*, 12(5):555–567, 1960.

10 H. E. Robbins. A theorem on graphs, with an application to a problem of traffic control. *The American Mathematical Monthly*, 46(5):281–283, 1939. URL: http://www.jstor.org/stable/2303897.

Dynamic Matching: Reducing Integral Algorithms to Approximately-Maximal Fractional Algorithms

Moab Arar
Tel Aviv University, Tel Aviv, Israel

Shiri Chechik*
Tel Aviv University, Tel Aviv, Israel

Sarel Cohen
Tel Aviv University, Tel Aviv, Israel

Cliff Stein*[1]
Columbia University, New York, USA

David Wajc*[2]
Carnegie Mellon University, Pittsburgh, USA

Abstract

We present a simple randomized reduction from fully-dynamic integral matching algorithms to fully-dynamic "approximately-maximal" fractional matching algorithms. Applying this reduction to the recent fractional matching algorithm of Bhattacharya, Henzinger, and Nanongkai (SODA 2017), we obtain a novel result for the integral problem. Specifically, our main result is a randomized fully-dynamic $(2+\epsilon)$-approximate integral matching algorithm with small polylog worst-case update time. For the $(2+\epsilon)$-approximation regime only a *fractional* fully-dynamic $(2+\epsilon)$-matching algorithm with worst-case polylog update time was previously known, due to Bhattacharya et al. (SODA 2017). Our algorithm is the first algorithm that maintains approximate matchings with worst-case update time better than polynomial, for any constant approximation ratio. As a consequence, we also obtain the first constant-approximate worst-case polylogarithmic update time maximum weight matching algorithm.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic, Worst-case, Maximum Matching, Maximum Weight Matching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.7

Related Version A full version of this paper is available at [6], `https://arxiv.org/abs/1711.06625`.

Acknowledgements We thank Monika Henzinger for sharing a preprint of [15] with us, and Virginia Vassilevska Williams for sharing a preprint of [50] with us. The fifth author wishes to thank Seffi Naor for a remark which inspired Definition 1.1.

* Work done in part while the authors were at the Simons Institute for the Theory of Computing.
[1] Supported in part by NSF grant CCF-1421161.
[2] Supported in part by NSF grants CCF-1527110, CCF-1618280 and NSF CAREER award CCF-1750808.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 7; pp. 7:1–7:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The maximum matching problem is one of the most widely-studied problems in computer science and operations research, with a long history and theory [5, 38]. On n-vertex and m-edge graphs, the state-of-the art maximum matching algorithms require $O(m\sqrt{n})$ and $O(n^\omega)$ time [40, 41] (here $\omega < 2.37$ is the matrix multiplication exponent [54]). For bipartite graphs, simpler algorithms with the same asymptotic running times are known [34, 41], as well as a faster, $O(m^{10/7} \cdot \text{poly}(\log n))$-time algorithm, due to the recent breakthrough of Mądry[39] for the maximum flow problem. For approximate matchings, it is long- and well-known that a matching admitting no augmenting paths of length $O(1/\epsilon)$ forms a $(1+\epsilon)$-approximate maximum matching (see [34]). The linear-time "blocking flow" subroutines of [34, 40] therefore result in an $O(m/\epsilon)$-time $(1+\epsilon)$-approximate maximum matching.

The maximum weight matching (MWM) problem has also garnered much interest over the years. For general weights, the seminal work of Edmonds [22] shows how to reduce the problem on bipartite graphs to the solution of n non-negative single-source shortest path instances. Relying on Fibonacci Heaps of Fredman and Tarjan [23], this approach yields the current fastest strongly-polynomial running time for the problem, $O(n(m + n\log n))$. Gabow [24] later showed how to obtain the same running time for general graphs. For integer weights $w : E \to \{0, 1, 2, \ldots, N\}$, algorithms nearly matching the state-of-of-the-art for the unweighted problem, with either logarithmic or linear dependence on N, are known.[3] These include an $O(m\sqrt{n}\log(nN))$-time algorithm [25], an $O(Nn^\omega)$-time algorithm [48] and a recent $O(m^{10/7} \cdot \text{poly}(\log n) \cdot \log N)$-time algorithm for bipartite graphs [17]. For approximation algorithms, an algorithm nearly matching the unweighted problem's guarantees is known, yielding a $(1+\epsilon)$-approximate maximum weight matching in $O((m/\epsilon)\log(1/\epsilon))$ time, [21].

All of the above results pertain to the static problem; i.e., where the input is given and we only need to compute a maximum matching on this given input. However, in many applications the graphs considered are inherently *dynamic*, with edges removed or added over time. One could of course address such changes by recomputing a solution from scratch, but this could be wasteful and time-consuming, and such applications may require immediately updating the solution given, as having users wait on a solution to be recomputed may likely be unsatisfactory. Consider for example point to point shortest path computation, a problem routinely solved by navigation systems: for such an application, the temporary closure of some road due to construction should not result in unresponsive GPS applications, busy re-computing the relevant data structures (see e.g.,[7, 36, 33, 19, 20, 51, 52, 10, 26, 28, 2, 31, 29, 30, 32, 3, 4]). Therefore, for such applications we want to update our solution quickly for *every update*, using fast *worst-case* (rather than amortized) update time.

Returning to the maximum matching problem, we note that a maximum matching can be trivially updated in $O(m)$ time. Sankowski [47] showed how to maintain the *value* of the maximum matching in $O(n^{1.495})$ update time.[4] On the other hand, Abboud and Vassilevska Williams [1] and Kopelowitz et al. [37] presented lower bounds based on long-standing conjectures, showing that even maintaining the maximum matching value likely requires $\Omega(m^c)$ update time for some constant $c \geq \frac{1}{3}$.

Given these hardness results for exact solutions, one is naturally inclined to consider fast *approximate* solutions. Trivially updating a maximal matching (and therefore a 2-

[3] Indeed, a black-box reduction of Pettie [46] from maximum weight matching to the maximum matching problem shows that a linear dependence in N is the largest possible gap between these two problems.

[4] We emphasize that this algorithm does not maintain an actual matching, but only the optimal value, and it seems unlikely to obtain such update times for maintaining a matching of this value.

approximate maximum matching) can be done using $O(n)$ worst-case update time. The goal is to obtain sublinear update times – ideally polylogarithmic (or even constant) – with as low an approximation ratio as possible.

The first non-trivial result for fully-dynamic maximum matching is due to Ivkovic and Lloyd [35], who presented a maximal matching algorithm with $O((m+n)^{1/\sqrt{2}})$ amortized update time. Note that this bound is sublinear only for sufficiently sparse graphs. The problem of approximate maximum matchings remained largely overlooked until 2010, when Onak and Rubinfeld [44] presented a fully-dynamic constant-approximate $O(\log^2 n)$ (amortized) update time algorithm. Additional results followed in quick succession.

Baswana et al. [8] showed how to maintain a maximal matching in $O(\log n)$ expected update time, and $O(\log^2 n)$ update time w.h.p. This was recently improved by Solomon [49] who presented a maximal matching algorithm using $O(1)$ update time w.h.p. For deterministic algorithms, Neiman and Solomon [42] showed how to maintain 3/2-approximate matchings deterministically in $O(\sqrt{m})$ update time, a result later improved by Gupta and Peng [27] to obtain $(1+\epsilon)$-approximate matchings in $O(\sqrt{m}/\epsilon^2)$. This result was in turn refined by Peleg and Solomon [45], who obtained the same approximation ratio and update time as [27] with $\sqrt{m}$ replaced by the maximum arboricity of the graph α (which is always at most $\alpha = O(\sqrt{m})$). Bernstein and Stein [11, 12] and Bhattacharya et al. [13] presented faster polynomial update time algorithms (with higher approximation ratios), and Bhattacharya et al. [14] presented a $(2+\epsilon)$-approximate algorithm with $\text{poly}(\log n, \epsilon^{-1})$ update time. See Table 1 for an in-depth tabular exposition of previous work and our results.[5] In §5 we discuss our results for MWM, also widely studied in the dynamic setting (see, e.g. [8, 27, 49, 50]).

Note that in the previous paragraph we did not state whether the update times of the discussed algorithms were worst case or amortized. We now address this point. As evidenced by Table 1, previous fully-dynamic matching algorithms can be broadly divided into two classes according to their update times: polynomial update time algorithms and polylogarithmic *amortized* update time algorithms. The only related polylogarithmic worst-case update time algorithms known to date were *fractional* matching algorithms, due to Bhattacharya et al. [15]. We bridge this gap by presenting the first fully-dynamic integral matching (and weighted matching) algorithm with polylogarithmic worst-case update times and constant approximation ratio. In particular, our approach yields a $(2+\epsilon)$-approximate algorithm, within the $O_\epsilon(\log^3 n)$ time bound of [15], but for *integral* matching.[6]

1.1 Our Contribution

Our main technical result requires the following natural definition of (c,d)-approximately-maximal fractional matchings.

▶ **Definition 1.1** (Approximately-Maximal Fractional Matching)**.** We say that a fractional matching $w : E \to \mathbb{R}^+$ is (c,d)-*approximately-maximal* if every edge $e \in E$ either has fractional value $w_e \geq 1/d$ or has one endpoint v with sum of incident edges' weights at least $W_v \triangleq \sum_{e \ni v} w_e \geq 1/c$ and moreover all edges e' incident on this v have $w_{e'} \leq 1/d$.

[5] For the sake of simplicity we only list bounds here given in terms of n and m. In particular, we do not state the results for arboricity-bounded graphs, which in the worst case (when the arboricity of a graph is $\alpha = \Theta(\sqrt{m})$) are all outperformed by algorithms in this table, with the aforementioned algorithm of Peleg and Solomon [45] being the lone exception to this rule.

[6] Independently of our work, and using a different approach, Charikar and Solomon [16] obtained a $(2+\epsilon)$-approximate dynamic matching algorithm with $O_\epsilon(\log^7 n)$ worst-case update time. For fixed ϵ their algorithm is slower than ours, and is arguably more complicated than our approach.

Table 1 Our Results and Previous Results for Fully-Dynamic Matching.
(All references are to the latest publication, with the first publication venue in parentheses.)

Approx.	Update Time	det.	w.c.	notes	reference
$O(1)$	$O(\log^2 n)$	✗	✗		Onak and Rubinfeld (STOC '10) [44]
$4+\epsilon$	$O(m^{1/3}/\epsilon^2)$	✓	✓		Bhattacharya et al. (SODA '15) [13]
$3+\epsilon$	$O(\sqrt{n}/\epsilon)$	✓	✗		Bhattacharya et al. (SODA '15) [13]
$2+\epsilon$	$\text{poly}(\log n, 1/\epsilon)$	✓	✗		Bhattacharya et al. (STOC '16) [14]
$2+\epsilon$	$\text{poly}(\log n, 1/\epsilon)$	✗	✓	w.h.p	**This work**[6]
2	$O((m+n)^{1/\sqrt{2}})$	✓	✗		Ivković and Lloyd (WG '93) [35]
2	$O(\log n)$	✗	✗	$O(\log^2 n)$ w.h.p	Baswana et al. (FOCS '11) [9]
2	$O(1)$	✗	✗	w.h.p	Solomon (FOCS '16) [49]
$3/2+\epsilon$	$O(\sqrt[4]{m}/\epsilon^{2.5})$	✓	✓	bipartite only	Bernstein and Stein (ICALP '15) [11]
$3/2+\epsilon$	$O(\sqrt[4]{m}/\epsilon^{2.5})$	✓	✗		Bernstein and Stein (SODA '16) [12]
$3/2$	$O(\sqrt{m})$	✓	✓		Neiman and Solomon (STOC '13) [43]
$1+\epsilon$	$O(\sqrt{m}/\epsilon^2)$	✓	✓		Gupta and Peng (FOCS '13) [27]

Note that this definition generalizes maximal fractional matchings (for which $c = d = 1$). The second condition required of v above (i.e., having no incident edges e' with $w_{e'} > 1/d$) may seem a little puzzling, but will prove important later; it can be safely ignored until §2.1 and §3.

Our main qualitative result, underlying our quantitative result, is the following black-box reduction from integral matching algorithms to approximately-maximal fractional matching algorithms, as stated in the following theorem.

▶ **Theorem 1.2.** *Let $\mathcal{A}$ be a fully-dynamic (c,d)-approximately-maximal fractional matching algorithm whose update time $T(n,m)$ and which changes at most $C(n,m)$ edge weights per update, for some $c \geq 1$, $d \geq \frac{6c\cdot\ln(\max\{n,1/\epsilon\})}{\epsilon^2}$, with $\epsilon \leq \frac{1}{2}$. Then, there exists a randomized fully-dynamic* integral *$2c(1+O(\epsilon))$-approximate matching algorithm $\mathcal{A}'$ (with this bound holding both w.h.p and in expectation) with update time $T(n,m)+O(C(n,m)\cdot d/\epsilon^2)$. Moreover, if $T(n,m)$ and $C(n,m)$ are worst-case bounds, so is the update time of Algorithm $\mathcal{A}'$.*

Now, one may wonder whether fully-dynamic (c,d)-approximately-maximal fractional matching algorithms with low worst-case update time and few edge weight changes exist for any non-trivial values of c and d. Indeed, the recent algorithm of Bhattacharya et al. [15] is such an algorithm, as the following lemma asserts.

▶ **Lemma 1.3** ([15])**.** *For all $\epsilon \leq \frac{1}{2}$, there is a fully-dynamic $(1+2\epsilon, \max\{54\log n/\epsilon^3, (3/\epsilon)^{21}\})$-approximately-maximal fractional matching algorithm with $T(n,m) = O(\log^3 n/\epsilon^7)$ worst-case update time, using at most $C(n,m) = O(\log n/\epsilon^2)$ edge weight changes per update in the worst case.*

We highlight the general approach of the algorithm of Bhattacharya et al. [15] in §2.1 to substantiate the bounds given in Lemma 1.3. Plugging the values of c, $T(n,m)$ and $C(n,m)$ of Lemma 1.3 into Theorem 1.2 immediately yields our result, given in the following theorem.

▶ **Theorem 1.4.** *For all $\epsilon \leq \frac{1}{2}$, there exists a randomized fully-dynamic $(2 + O(\epsilon))$-approximate integral matching algorithm (w.h.p and in expectation) with worst-case update time $O(\log^3 n/\epsilon^7 + \log(\max\{n, 1/\epsilon\})/\epsilon^2 \cdot \max\{\log n/\epsilon^3, (3/\epsilon)^{21}\}) = O_\epsilon(\log^3 n)$.*

We recall that until now (barring the aforementioned independent result of Charikar and Solomon [16]), for worst-case polylog update times only *fractional* dynamic algorithms – algorithms which only approximate the *value* of the maximum matching – were known for this problem.

Finally, combined with the recent black-box reduction of Stubbs and Vassilevska Williams [50] from the weighted to the unweighted matching problem, our algorithm also yields the first fully-dynamic constant-approximate maximum weight matching algorithm with polylogarithmic worst-case update time.

▶ **Theorem 1.5.** *For all $\epsilon \leq \frac{1}{2}$, there exists a randomized fully-dynamic $(4+O(\epsilon))$-approximate maximum weight matching algorithm with poly$(\log n, 1/\epsilon)$ worst-case update time. The approximation guarantee holds with high probability and in expectation.*

1.2 Our Techniques

Our framework yielding our main result combines three ingredients: approximately-maximal fractional matchings, kernels and fast $(1+\epsilon)$ matching algorithms for bounded-degree graphs. We give a short exposition of these ingredients and conclude with how we combine all three.

Approximately-Maximal Fractional Matchings. The first ingredient we rely on is (c, d)-approximately-maximal fractional matchings, introduced in the previous section. Recalling that for such solutions, each edge has value at least $1/d$ or one of its endpoints has sum of incident edge values at least $1/c$. This approximate maximality condition implies this fractional matching has high value compared to the maximum matching size; specifically, this fractional matching's size is at least a $1/2 \max\{c, d\}$ fraction of this value (easily verifiable using LP duality). As we shall show, approximate maximality also allows one to use these fractional values to sample a subgraph in the support of this fractional matching which contains a large *integral* matching compared to G, with high probability. We discuss the dynamic fractional matching algorithm of Bhattacharya et al. [15] and show that it maintains an approximately-maximal fractional matching in §2.1.

Kernels. The second ingredient we rely on is the notion of *kernels*, introduced by [13]. Roughly speaking, a kernel is a low-degree subgraph H of G such that each edge of G not taken into H has at least one endpoint whose degree in H is at least $1/c$ times the maximum degree in H. Relying on Vizing's Theorem [53], we show in §2.2 that such a graph has maximum matching size $\mu(H)$ at least $1/(2c + \epsilon)$ of the matching size of G, previously only known for kernels of bipartite graphs, where this is easily verifiable via LP duality.[7] Efficiently maintaining a large matching can therefore be reduced to maintaining a low-degree kernel, given the last ingredient of our approach.

[7] As a byproduct of our proof, we show how the algorithms of Bhattacharya et al. [13] can be made $(2 + \epsilon)$-approximate within the same time bounds. As this is tangential to our main result, we do not elaborate on this.

Bounded-Degree $(1+\epsilon)$-matching. The final ingredient we rely on for our framework is $(1+\epsilon)$ matching algorithms with worst-case update time bounded by the graph's maximum degree, such as the algorithms of Gupta and Peng [27] and Peleg and Solomon [45].

Our approach in a nutshell

Given the above ingredients, our framework is a simple and natural one. Throughout our algorithm's run, we run a fully-dynamic (c, d)-approximately-maximal fractional matching algorithm with efficient worst-case update. Sampling edges independently according to this fractional value (times some logarithmic term in n, to guarantee concentration) allows us to sample a kernel of logarithmic maximum degree, with each non-sampled edge having at least one endpoint with degree at least $1/c$ times the maximum subgraph degree, with high probability. As the obtained subgraph H therefore has a maximum matching of size at least $\approx 1/2c$ times the maximum matching in G, a $(1+\epsilon)$-matching algorithm in H yields a $\approx 2c + O(\epsilon)$ matching in G. We then maintain a $(1+\epsilon)$-matching in H (which by virtue of H's bounded degree we can do in logarithmic worst-case time) following each update to H incurred by a change of some edge's fractional value by the dynamic fractional matching algorithm. The obtained integral algorithm's update time is dominated by two terms: the running time of the fractional algorithm, and the number of edge weight changes per update, times $O(\log n)$. This concludes the high-level analysis of the obtained approximation ratio and update time of our approach, as given in Theorem 1.2.

Wider applicability

We stress that our framework is general, and can use any approximately-maximal fractional matching algorithm. Consequently, any improvement on the running time and number of edge value changes for maintaining approximately-maximal fractional matchings would yield a faster worst-case update time.

2 Preliminaries

In this section we introduce some previous results which we will rely on in our algorithm and its analysis. We start by reviewing the approach of Bhattacharya et al. [15] to obtain efficient fractional algorithms in §2.1. We then discuss the bounded-degree subgraphs we will consider, also known as *kernels*, in §2.2. Finally, we briefly outline the $(1+\epsilon)$-approximate $O(\Delta/\epsilon^2)$ worst case update time algorithms we will rely on for our algorithm, in §2.3.

2.1 Hierarchical Partitions

In this section we review the approximately-maximal fractional matchings maintained by Bhattacharya et al. [15]. At a high level, this algorithm relies on the notion *hierarchical partitions*, in which vertices are assigned some *level* (the partition here is given by the level sets), and edges are assigned a fractional value based on their endpoints' levels. Specifically, an edge is assigned a value exponentially small in its vertices' maximum level. The levels (and therefore the edge weights) are updated in a way as to guarantee feasibility, as well as guaranteeing that a vertex v of high level has high sum of incident edge weights, W_v. These conditions are sufficient to guarantee approximate maximality, as we shall soon show.

The hierarchical partitions considered by Bhattacharya et al. [15], termed simply *nice partitions*, is described as follows. In the definition constants β, K, L and a function $f(\beta) = 1 - 3/\beta$ are used, satisfying the following.

$$\beta \geq 5,\ K = 20,\ f(\beta) = 1 - 3/\beta,\ L = \lceil \log_\beta n \rceil. \tag{1}$$

In our case, for some $\epsilon \leq \frac{1}{2}$, we will let $\beta = \frac{3}{\epsilon} (\geq 5)$. As we will be shooting for $O(1)$-approximation algorithms with polylogarithmic update time and our reduction's update time has polynomial dependence on ϵ^{-1}, we will assume without loss of generality that $\epsilon = \Omega(\frac{3}{\sqrt[20]{n}})$, and so for n large enough, we have $K \leq L$.

▶ **Definition 2.1** (A nice partition [15]). In a *nice partition* of a graph $G = (V, E)$, each vertex v is assigned an integral *level* $\ell(v)$ in the set $\{K, K+1, \ldots, L\}$. In addition, for each vertex $v \in V$ and edge $e \ni v$ the *shadow-level of v with respect to e*, denoted by $\ell_v(e)$, is a (positive) integer satisfying $\ell(v) - 1 \leq \ell_v(e) \leq \ell(v) + 1$. Moreover, for each vertex v, we have

$$\max_{e \ni v} \ell_v(e) - \min_{e \ni v} \ell_v(e) \leq 1. \tag{2}$$

The level of an edge $e = (u, v)$ is taken to be the maximum shadow-level of an endpoint of e with respect to e; i.e., $\ell(u, v) = \max\{\ell_u(e), \ell_v(e)\}$. Let $W_v = \sum_{e \in v} w_e$ be the sum of weights of edges incident on a vertex v. Then,

1. For every edge $e \in E$, it holds that $w_e = \beta^{-\ell(e)}$.
2. For every node $v \in V$, it holds that $W_v < 1$.
3. For every node $v \in V$ with level $\ell(v) > K$, it holds that $W_v \geq f(\beta)$.

The intuition behind this definition in Bhattacharya et al. [15] is to mimic the hierarchical partition of Bhattacharya et al. [13], termed (α, β)-*decompositions* there. (α, β)-decompositions are the special case of nice partitions where the shadow-level of a vertex v with respect to each edge $e \ni v$ is precisely equal to the vertex's level; i.e, $\ell_v(e) = \ell(v)$ (with α denoting $f(\beta)/\beta$). The advantage of this more relaxed notion of shadow-level is to allow a vertex to move between levels "slowly", only notifying *part* of its incident edges of its level change between updates, and therefore only updating some of its edges' weights. This allows for maintaining this partition with fast worst-case update time, as shown in Bhattacharya et al. [15] (more on this below).

This above intuition concerning nice partitions will not prove important for our analysis. The crucial property we will rely on is given by the following lemma, which asserts that the fractional matching associated with a nice partition is approximately-maximal.

▶ **Lemma 2.2.** *Let $\epsilon \leq \frac{1}{2}$. Consider a nice partition with parameter $\beta = 3/\epsilon \geq 6 \geq 5$, and so $f(\beta) = 1 - \epsilon$. Then, the fractional matching associated with this nice partition is $(1 + 2\epsilon, \max\{54 \log n/\epsilon^3, (3/\epsilon)^{21}\})$-approximately-maximal.*

Proof. Let $K' = \max\{\lceil \log_\beta(18c \log n/\epsilon^2) \rceil, K+1\}$ and $d = \beta^{K'} \leq \max\{54 \log n/\epsilon^3, (3/\epsilon)^{21}\}$. For any edge e, if $\ell(e) = \max_{v \in e}\{\ell_v(e)\} \leq K'$, then by definition $w_e = \beta^{-\ell(e)} \geq \beta^{-K'} = \frac{1}{d}$. Alternatively, if $w_e < \frac{1}{d}$ then $\ell(e) > K'$ and therefore by integrality of $\ell(e)$, we have $\ell(e) \geq K' + 1$. Now, let v be $\arg\max_{v \in e}\{\ell_v(e)\} \geq K' + 1$. Then, by definition of shadow-levels and $K' > K$, we have $\ell(v) \geq \ell_v(e) - 1 \geq K' > K$ and so by Property 3 of a nice partition we have $W_v > f(\beta) = 1 - \epsilon \geq \frac{1}{1+2\epsilon}$ (as $\epsilon \leq \frac{1}{2}$). But on the other hand, by Equation (2), we also know that for every edge $e' \ni v$,

$$\ell_v(e') \geq \min_{e' \ni v} \ell_v(e') \geq \max_{e' \ni v} \ell_v(e') - 1 \geq \ell_v(e) - 1 \geq K' > K.$$

Therefore, by definition of the edge weights, each edge $e' \ni v$ satisfies $w_{e'} \leq \beta^{-K'} \leq \frac{1}{d}$. ◀

The recent result of Bhattacharya et al. [15] for maintaining nice partitions in polylogarithmic worst-case update time together with Lemma 2.2 immediately implies Lemma 1.3, restated below. We substantiate these bounds with the dependence on ϵ stated explicitly in the full version of this paper [6], as Bhattacharya et al. [15] had $\epsilon = O(1)$ and so their results do not state these dependencies explicitly.

▶ **Lemma 1.3** ([15]). *For all $\epsilon \leq \frac{1}{2}$, there is a fully-dynamic $(1+2\epsilon, \max\{54 \log n/\epsilon^3, (3/\epsilon)^{21}\})$-approximately-maximal fractional matching algorithm with $T(n,m) = O(\log^3 n/\epsilon^7)$ worst-case update time, using at most $C(n,m) = O(\log n/\epsilon^2)$ edge weight changes per update in the worst case.*

As we shall show, approximately-maximal fractional matchings allow us to sample a bounded-degree subgraph H of G containing a large matching compared to the maximum matching size in G, $\mu(G)$. For this we will require the notion of *kernels*, defined in §2.2.

2.2 Kernels

In this section we review the concept of kernels, first introduced by Bhattacharya et al. [13].

▶ **Definition 2.3** (Kernels [13]). A (c,d)-*kernel* of a graph G is a subgraph H of G satisfying:
1. For each vertex $v \in V$, the degree of v in H is at most $d_H(v) \leq d$.
2. For each edge $(u,v) \in E \setminus H$, it holds that $\max\{d_H(u), d_H(v)\} \geq d/c$.

The interest in finding a bounded-degree subgraph H of G may seem natural, as one may expect to be able to compute a matching quickly in H due to its sparsity (we elaborate more on this point in §2.3). The interest in satisfying the second property, on the other hand, may seem a little cryptic. However, combining both properties implies that the matching number of H, $\mu(H)$, is large in comparison with the matching number of G, $\mu(G)$.

▶ **Lemma 2.4.** *Let H be a (c,d)-kernel of G for some $c \geq 1$. Then $\mu(H) \geq \frac{1}{2c(1+1/d)} \cdot \mu(G)$.*

Proof. Let M^* be some maximum matching in G (i.e., $|M^*| = \mu(G)$). Consider the following fractional matching solution:

$$f_{u,v} = \begin{cases} \frac{1}{d} & (u,v) \in H \setminus M^* \\ \max\{1 - \frac{d_h(u)+d_H(v)-2}{d}, 0\} & (u,v) \in H \cap M^*. \end{cases}$$

This is a feasible fractional matching due to the degree bound of H and the fractional values assigned to edges of a vertex v incident on an edge $e \in H \cap M^*$ being at most $\frac{d_H(v)-1}{d} + \frac{d-d_H(v)+1}{d} = 1$. To show that this fractional matching has high value, consider the variables $y_v = \sum_u f_{u,v}$. On the one hand, by the handshake lemma, $\sum_{u,v} f_{u,v} = \frac{1}{2}\sum_v y_v$. On the other hand, each edge (u,v) of $M^* \cap H$ has $y_u + y_v \geq 1 \geq \frac{1}{c}$ by construction and each edge of $M^* \setminus H$ has at least one endpoint v of degree $d_H(v) \geq \frac{d}{c}$, implying that $y_u + y_v \geq \frac{d}{c} \cdot \frac{1}{d} = \frac{1}{c}$ for each $(u,v) \in M^* \setminus H$. As each vertex v neighbors at most one edge of the (optimal) matching M^*, we obtain

$$\sum_e f_e = \frac{1}{2} \cdot \sum_v y_v \geq \frac{1}{2c} \cdot |M^*| = \frac{1}{2c} \cdot \mu(G).$$

Now, to show that H contains a large *integral* matching, we rely on Vizing's Theorem [53], which asserts that every multigraph of maximum degree Δ and maximum edge multiplicity μ has a proper $\Delta + \mu$ edge-coloring; i.e., a partition of the edge multiset into $\Delta + \mu$ edge-disjoint

matchings. To use this theorem, we construct a multigraph on the same vertex set V with each edge e replaced by $f_e \cdot d$ parallel copies (note that $f_e \cdot d$ is integral). By construction, the number of edges in this multigraph is $\sum_e f_e \cdot d$. By feasibility of f, we have that this multigraph has maximum degree d. By Vizing's Theorem, the simple subgraph obtained by ignoring parallel edges corresponding to edges in $H \cap M^*$ can be edge colored using $d+1$ colors. But for each edge $e = (u, v) \in H \cap M^*$, such a coloring uses at most $d_H(u) - 1 + d_H(v) - 1$ distinct colors on edges other than (u, v) incident on u or v. To extend this $d+1$ edge coloring to a proper coloring of the multigraph, we color the $\max\{d - (d_H(u) - 1 + d_H(v) - 1), 0\}$ multiple edges (u, v) in this multigraph using some $\max\{d - (d_H(u) - 1 + d_H(v) - 1), 0\}$ colors of the palette of size $d + 1$ not used on the other edges incident on u and v. We conclude that this multigraph, which is contained in H and has $\sum_e f_e \cdot d$ edges, is $d+1$ edge colorable and therefore H contains an integral matching of size at least

$$\frac{1}{d+1} \cdot \sum_e f_e \cdot d = \frac{1}{1+1/d} \cdot \sum_e f_e \geq \frac{1}{2c(1+1/d)} \cdot \mu(G). \qquad \blacktriangleleft$$

Lemma 2.4 and the algorithm of §2.3 immediately imply that the algorithms of Bhattacharya et al. [13] can be made $(2+\epsilon)$-approximate within the same time bounds (up to poly$(1/\epsilon)$ terms). As this was previously also observed in Bhattacharya et al. [14], we do not elaborate on this point here.

2.3 Nearly-Maximum Matchings in Degree-Bounded Graphs

In this short subsection we highlight one final component we will rely on for our reduction: fast nearly-optimal matching algorithms with worst-case update time bounded by G's maximum degree. Such algorithms were given by Peng and Gupta [27] and Peleg and Solomon [45]. More precisely, we have the following lemma.

▶ **Lemma 2.5** ([27, 45]). *There exists a dynamic $(1+\epsilon)$-approximate matching algorithm with worst-case $O(\Delta/\epsilon^2)$ update time in dynamic graphs of maximum degree at most Δ.*

The bound for the algorithm of [45] follows as $\alpha \leq \Delta$ always, while the bound for the algorithm of [27] is immediate by inspecting this algorithm, as observed in [45].

3 Sampling Using Approximately-Maximal Matchings

In what follows we will show that sampling edges independently with probability roughly proportional to their assigned value according to an approximately-maximal fractional matching yields a kernel of logarithmic maximum degree with high probability.

▶ **Lemma 3.1.** *Let $\epsilon \leq \frac{1}{2}$. Let $w : E \to \mathbb{R}^+$ be a (c, d)-approximately-maximal fractional matching with $c \geq 1$ and $d \geq \frac{6c \cdot \ln(\max\{n, 1/\epsilon\})}{\epsilon^2}$. Then, a subgraph H obtained by sampling each edge e independently with probability*

$$\min\{1, w_e \cdot d\} \tag{3}$$

is a $(c(1+O(\epsilon)), (1+\epsilon)d)$-kernel of G with probability at least $1 - \frac{2}{\max\{n, 1/\epsilon\}}$.

Proof. For any vertex $v \in V$, denote by D_v the random variable which corresponds to v's degree in H. As before, denote by $W_v = \sum_{e \ni v} w_e \leq 1$ the sum of edge weights of edges incident on v.

First, we prove the degree upper bound; i.e., Property 1 of a kernel. As w is a fractional matching, we know that $W_v \leq 1$. Therefore, by Equation (3), we have that $\mathbb{E}[D_v] \leq d$. By standard Chernoff bounds, as $d \geq \frac{6c \cdot \ln(\max\{n,1/\epsilon\})}{\epsilon^2}$ and $c \geq 1$, we find that

$$\Pr[D_v \geq (1+\epsilon) \cdot d] \leq \exp\left(\frac{-\epsilon^2 \cdot d}{3}\right) \leq \frac{1}{\max\{n,1/\epsilon\}^{2c}} \leq \frac{1}{\max\{n,1/\epsilon\}^2}.$$

Next, we prove that any edge not sampled into H will, with high probability, be incident on some high-degree vertex in H; i.e., we show that H satisfies Property 2 of a kernel. First, note that an edge e with $w_e \geq 1/d$ will be sampled with probability one, given our sampling probability given in Equation (3), therefore trivially satisfying Property 2 of a kernel. Conversely, an edge e with $w_e < 1/d$ has some endpoint v with $W_v \geq 1/c$ and all edges e' incident on v have $w_{e'} \leq 1/d$, since w is (c,d)-approximately maximal. Therefore, by Equation (3) each edge e' incident on v is sampled with probability precisely $w_{e'} \cdot d$. Consequently, we have that $\mu_v = \mathbb{E}[D_v] = W_v \cdot d \geq d/c$. By standard Chernoff bounds, as $\mu_v \geq d/c \geq \frac{6\ln(\max\{n,1/\epsilon\})}{\epsilon^2}$, we find that

$$\Pr[D_v \leq (1-\epsilon) \cdot d/c] \leq \Pr[D_v \leq (1-\epsilon) \cdot \mu_v] \leq \exp\left(\frac{-\epsilon^2 \cdot \mu_v}{2}\right) \leq \frac{1}{\max\{n,1/\epsilon\}^3}.$$

Taking a union bound over the $O(n^2)$ possible bad events corresponding to violating a property of a $(c(1+\epsilon)/(1-\epsilon)), d(1+\epsilon))$-kernel, we find that probability at least $1 - \frac{2}{\max\{n,1/\epsilon\}}$:

1. For each vertex $v \in V$, it holds that $d_H(v) \leq (1+\epsilon) \cdot d$.
2. For each edge $(u,v) \in E \setminus H$, it holds that $\max\{d_H(u), d_H(v)\} \geq (1-\epsilon) \cdot d/c$.

In other words, H is a $(c(1+\epsilon)/(1-\epsilon), d(1+\epsilon))$-kernel of G with high probability. ◀

4 Our Reduction

Given the previous sections, we are now ready to describe our reduction from fully-dynamic integral matching to approximately-maximal fractional matching and analyzing its performance, given by Theorem 1.2, restated here.

▶ **Theorem 1.2.** *Let $\mathcal{A}$ be a fully-dynamic (c,d)-approximately-maximal fractional matching algorithm whose update time $T(n,m)$ and which changes at most $C(n,m)$ edge weights per update, for some $c \geq 1$, $d \geq \frac{6c \cdot \ln(\max\{n,1/\epsilon\})}{\epsilon^2}$, with $\epsilon \leq \frac{1}{2}$. Then, there exists a randomized fully-dynamic* integral *$2c(1+O(\epsilon))$-approximate matching algorithm $\mathcal{A}'$ (with this bound holding both w.h.p and in expectation) with update time $T(n,m)+O(C(n,m)\cdot d/\epsilon^2)$. Moreover, if $T(n,m)$ and $C(n,m)$ are worst-case bounds, so is the update time of Algorithm $\mathcal{A}'$.*

Proof. Our reduction works as follows. Whenever an edge e is added/removed from G, we update the (c,d)-approximately-maximal fractional matching, using algorithm $\mathcal{A}$, in time $T(n,m)$. We then sample each of the at most $C(n,m)$ edges e whose value is changed, independently, with probability given by Equation (3). To control the maximum degree in the sampled subgraph H', every vertex v maintains a list of at most $(1+\epsilon) \cdot d$ sampled edges "allowable" for use in H. (This list can be maintained dynamically in $O(1)$ time per update in H' in a straightforward manner.) We let H be the graph induced by the sampled edges "allowed" by both their endpoints. Finally, we use a $(1+\epsilon)$-matching algorithm as in Lemma 2.5 to maintain a matching in the sampled subgraph H.

By Lemma 3.1 the subgraph H is a $(c(1+O(\epsilon)), (1+\epsilon)d)$-kernel of G with high probability (note that by the same lemma, all sampled edges of H' will appear in our H). By Lemma 2.4,

this means that with high probability this kernel H has matching number at least

$$\mu(H) \geq \frac{1}{2c(1+O(\epsilon))(1+1/d))} \cdot \mu(G) \geq \frac{1}{2c(1+O(\epsilon))} \cdot \mu(G),$$

where the second inequality follows from $d \geq \frac{6c \cdot \ln(\max\{n,1/\epsilon\})}{\epsilon^2} \geq \frac{1}{\epsilon}$. As the above lower bound on $\mu(H)$ holds with probability at least $1-2\epsilon$ by Lemma 3.1 and Lemma 2.4, we have that $\mathbb{E}[\mu(H)] \geq (1-2\epsilon) \cdot \frac{1}{2c(1+O(\epsilon))} \cdot \mu(G) \geq \frac{1}{2c(1+O(\epsilon))} \cdot \mu(G)$. Therefore, a $(1+\epsilon)$-approximate matching in H is itself a $2c(1+O(\epsilon))$-approximate matching in G w.h.p and in expectation. Now, each of the $C(n,m)$ changes to edge weights of the fractional matching incurs at most three updates to the kernel H: for every edge e whose weight changes, this edge can be added/removed to/from H if it is sampled in/out; in the latter case both of e's endpoints can have a new edge added to their "allowable" edge list in place of e, and therefore possibly added to H, in case the endpoints had less than $(1+\epsilon)d$ sampled edges. In either case, the number of edges added to H is at most a constant per edge weight update. But on the other hand, the $(1+\epsilon)$-approximate matching algorithm implied by Lemma 2.5 requires $O(d/\epsilon^2)$ worst-case time per update in H, by H's worst-case degree bound. Consequently, our algorithm maintains a $2c(1+O(\epsilon))$-approximate integral matching (w.h.p and in expectation) in $T(n,m) + O(C(n,m) \cdot d/\epsilon^2)$ update time; moreover, this update time is worst case if the bounds on $T(n,m)$ and $C(n,m)$ are themselves worst case. ◀

5 Applications to Maximum Weight Matching

In this section we highlight the consequences of our results for fully-dynamic maximum weight matching. First, we discuss a new reduction of Stubbs and Vassilevska Williams [50].

▶ **Lemma 5.1** ([50]). *Let $\mathcal{A}$ be an fully-dynamic α-approximate maximum cardinality matching algorithm with update time $T(n,m)$. Then, there exists a fully-dynamic $2\alpha(1+\epsilon)$-approximate maximum cardinality matching algorithm with update time $O(T(n,m) \cdot \frac{\log(n/\epsilon)}{\epsilon})$. Furthermore, if Algorithm $\mathcal{A}$ is deterministic, so is the new one, and if Algorithm $\mathcal{A}$'s update time is worst case, so is the new algorithm's update time.*

This reduction (which we elaborate on shortly), together with the state of the art dynamic maximum matching algorithms, implies most of the best currently best bounds for dynamic maximum weight matching, in Table 2.

A somewhat more involved and worse update time bound than that given in Lemma 5.1 was presented in [50], as that paper's authors sought to obtain a persistent matching, in a sense that this matching should not change completely after a single step (i.e., no more than $O(T(n,m))$ changes to the matching per edge update, if $T(n,m)$ is the algorithm's update time). However, a simpler and more efficient reduction yielding a non-persistent matching algorithm with the performance guarantees of Lemma 5.1 is implied immediately from the driving observation of Stubbs and Vassilevska Williams [50] (and indeed, is discussed in [50]). This observation, previously made by Crouch and Stubbs [18] in the streaming setting, is as follows: denote by E_i the edges of weights in the range $((1+\epsilon)^i, (1+\epsilon)^{i+1}]$, and let M_i be an α-approximate matching in $G[E_i]$. Then, greedily constructing a matching by adding edges from each M_i in decreasing order of i yields a $2\alpha(1+\epsilon)$-approximate maximum weight matching. Adding to this observation the observation that if we are content with a $(1+\epsilon)$-approximate (or worse) maximum weight matching we may safely ignore all edges of weight less than ϵ/n of the maximum edge weight (a trivial lower bound on the maximum weight matching's weight), we find that we can focus on the ranges $((n/\epsilon)^i, (n/\epsilon)^{i+2}]$, for some $i \in \mathbb{Z}$, noting that each edge belongs to at most two such ranges.

Table 2 Our Results and Previous State-of-the-Art for Fully-Dynamic MWM.

Approx.	Update Time	det.	w.c.	reference
$4+\epsilon$	$O(\log(n/\epsilon)/\epsilon)$	✗	✗	5.1 + Solomon (FOCS '16) [49]
$4+\epsilon$	$\text{poly}(\log n, 1/\epsilon)$	✓	✗	5.1 + Bhattacharya et al. (STOC '16) [14]
$4+\epsilon$	$O(m^{1/3}\log(n/\epsilon)/\epsilon^3)$	✓	✓	5.1 + Bhattacharya et al. (SODA '15) [13]
$4+\epsilon$	$\text{poly}(\log n, 1/\epsilon)$	✗	✓	5.1 + **This work**
$\approx 3.009+\epsilon$	$O(\sqrt{m}\log C\epsilon^{-3})$	✓	✓	Gupta and Peng (FOCS '13) [27]
$3+\epsilon$	$O(\sqrt[4]{m}\log(n/\epsilon)\epsilon^{-3})$	✓	✗	5.1 + Bernstein and Stein (SODA '16) [12]
$2+\epsilon$	$O(\sqrt{m}\cdot\frac{\log^2(n/\epsilon)}{\epsilon^4})$	✓	✓	5.1 + Gupta and Peng (FOCS '13) [27]
$1+\epsilon$	$O(\sqrt{m}C\epsilon^{-3})$	✓	✓	Gupta and Peng (FOCS '13) [27]
$1+\epsilon$	$O(\sqrt{m}\log C\epsilon^{-O(1/\epsilon)})$	✓	✓	Gupta and Peng (FOCS '13) [27]

In each such range $((n/\epsilon)^i, (n/\epsilon)^{i+2}]$, the argument of [18, 50] implies that maintaining α-approximate matchings in the sub-ranges $((1+\epsilon)^j, (1+\epsilon)^{j+1}]$ for integral ranges and combining these greedily result in a $2\alpha(1+\epsilon)$-approximate maximum weight matching in the range $((n/\epsilon)^i, (n/\epsilon)^{i+2}]$. Therefore, in the range containing a $(1+\epsilon)$-approximate MWM (such a range exists, by the above), this approach maintains a $2\alpha(1+O(\epsilon))$-approximate MWM. The only possible difficulty is combining these matchings greedily *dynamically*. This is relatively straightforward to do in $O(\log(n/\epsilon)/\epsilon)$ worst-case time per change of the α-approximate matching algorithm, however, implying the bound of Lemma 5.1.

As seen in Table 2, this reduction of Stubbs and Vassilevska Williams [50] implies a slew of improved bounds for fully-dynamic approximate maximum weight matching. Plugging in our bounds of Theorem 1.4 for fully-dynamic maximum matching into the reduction of Lemma 5.1 similarly yields the first constant-approximate maximum weight matching with polylogarithmic *worst-case* update time, given in Theorem 1.5 below.

▶ **Theorem 1.5.** *For all $\epsilon \leq \frac{1}{2}$, there exists a randomized fully-dynamic $(4+O(\epsilon))$-approximate maximum weight matching algorithm with poly$(\log n, 1/\epsilon)$ worst-case update time. The approximation guarantee holds with high probability and in expectation.*

6 Conclusion and Future Work

In this work we presented a simple randomized reduction from $(2c+\epsilon)$-approximate fully-dynamic matching to fully-dynamic (c,d)-approximately-maximal fractional matching with a slowdown of d. Using the recent algorithm of Bhattacharya et al. [15], our work yields the first fully-dynamic matching algorithms with faster-than-polynomial worst-case update time for any constant approximation ratio; specifically, it yields a $(2+O(\epsilon))$-approximate matching with polylog worst case update time. Our work raises several natural questions and future research directions to explore.

Faster Fractional Algorithms. Given our reduction, in order to obtain faster $(2+\epsilon)$-approximate matching algorithms, it would suffice to improve the update time for fully-dynamic $(1+\epsilon, O_\epsilon(\log n))$-approximately-maximal fractional matching algorithm compared to the algorithm of Bhattacharya et al. [15]. Large enough improvements in the update time, number of edge weight changes and parameter d of $(1+\epsilon, d)$-approximately-maximal

fractional matching algorithms used would result in even better bounds. We note however that our current approach does not seem to allow for sub-logarithmic update bounds, due to its update time's dependence on $d \geq \log n$, so obtaining worst-case sub-logarithmic bounds may require different ideas.

More Efficient/Deterministic Reduction. Given the above, one may ask whether the dependence on d may be removed in a black-box reduction such as ours, yielding randomized integral matching algorithms with the same running time as their fractional counterparts. One may further wonder whether or not one can obtain a *deterministic* counterpart to our black-box reduction from integral matching to approximately-maximal fractional matching. Such a reduction with polylogarithmic overhead would yield for example a deterministic $(2+\epsilon)$-approximate algorithm with worst-case polylogarithmic update time.

Maximal Matching. Finally, a natural question from our work and prior work is whether or not a *maximal* matching can be maintained in worst-case polylogarithmic time (also implying a 2-approximate minimum vertex cover within the same time bounds). We leave this as a tantalizing open question.

References

1. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 55th Symposium on Foundations of Computer Science (FOCS)*, pages 434–443, 2014.
2. Ittai Abraham and Shiri Chechik. Dynamic decremental approximate distance oracles with $(1+\epsilon, 2)$ stretch. *CoRR*, abs/1307.1516, 2013.
3. Ittai Abraham, Shiri Chechik, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. On dynamic approximate shortest paths for planar graphs with worst-case costs. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 740–753, 2016.
4. Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 440–452, 2017.
5. Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows - theory, algorithms, and applications*. Prentice hall, 1993.
6. Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. *CoRR*, abs/1711.06625, 2017. `arXiv:1711.06625`.
7. Giorgio Ausiello, Giuseppe F. Italiano, Alberto Marchetti-Spaccamela, and Umberto Nanni. Incremental algorithms for minimal length paths. *Journal of Algorithms*, 12(4):615–638, 1991.
8. Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In *Proceedings of the 52nd Symposium on Foundations of Computer Science (FOCS)*, pages 383–392, 2011.
9. Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $O(\log n)$ update time. *SIAM Journal on Computing (SICOMP)*, 44(1):88–113, 2015.
10. Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. *Journal of Algorithms*, 62(2):74–92, 2007.

11 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 167–179, 2015.

12 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 692–711, 2016.

13 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 785–804, 2015.

14 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 398–411, 2016.

15 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 470–489, 2017.

16 Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial barrier for worst-case time bounds. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming (ICALP)*, 2018.

17 Michael B Cohen, Aleksander Mądry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\tilde{O}(m^{10/7} \log W)$ time. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 752–771, 2017.

18 Michael Crouch and Daniel M Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *Proceedings of the 17th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, page 96, 2014.

19 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM (JACM)*, 51(6):968–992, 2004.

20 Camil Demetrescu and Giuseppe F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. *Journal of Computer and System Sciences*, 72(5):813–837, 2006.

21 Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM (JACM)*, 61(1):1, 2014.

22 Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

23 Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

24 Harold N Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 434–443, 1990.

25 Harold N Gabow and Robert E Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing (SICOMP)*, 18(5):1013–1036, 1989.

26 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *Proceedings of the 53rd Symposium on Foundations of Computer Science (FOCS)*, pages 748–757, 2012.

27 Manoj Gupta and Richard Peng. Fully dynamic $(1+\epsilon)$-approximate matchings. In *Proceedings of the 54th Symposium on Foundations of Computer Science (FOCS)*, pages 548–557, 2013.

28 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the o(mn) barrier and derandomization. In *Proceedings of the 54th Symposium on Foundations of Computer Science (FOCS)*, pages 538–547, 2013.

29 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *Proceedings of the 55th Symposium on Foundations of Computer Science (FOCS)*, pages 146–155, 2014.

30 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 674–683, 2014.

31 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A subquadratic-time algorithm for decremental single-source shortest paths. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1053–1072, 2014.

32 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Improved algorithms for decremental single-source reachability on directed graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 725–736, 2015.

33 Monika Rauch Henzinger and Valerie King. Maintaining minimum spanning forests in dynamic graphs. *SIAM Journal on Computing (SICOMP)*, 31(2):364–374, 2001.

34 John E Hopcroft and Richard M Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (SWAT)*, pages 122–125, 1971.

35 Zoran Ivkovic and Errol L Lloyd. Fully dynamic maintenance of vertex cover. In *Proceedings of the 19th International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 99–111, 1993.

36 Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS)*, pages 81–91, 1999.

37 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1272–1287, 2016.

38 László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Society, 2009.

39 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *Proceedings of the 54th Symposium on Foundations of Computer Science (FOCS)*, pages 253–262, 2013.

40 Silvio Micali and Vijay V Vazirani. An $O(\sqrt{|V|}|E|)$ algoithm for finding maximum matching in general graphs. In *Proceedings of the 21st Symposium on Foundations of Computer Science (FOCS)*, pages 17–27, 1980.

41 Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *Proceedings of the 45th Symposium on Foundations of Computer Science (FOCS)*, pages 248–255, 2004.

42 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC)*, pages 745–754, 2013.

43 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Transactions on Algorithms (TALG)*, 12(1):7, 2016.

44 Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 457–464, 2010.

45 David Peleg and Shay Solomon. Dynamic $(1 + \epsilon)$-approximate matchings: a density-sensitive approach. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 712–729, 2016.

46 Seth Pettie. A simple reduction from maximum weight matching to maximum cardinality matching. *Information Processing Letters (IPL)*, 112(23):893–898, 2012.

47 Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 118–126, 2007.

48 Piotr Sankowski. Maximum weight bipartite matching in matrix multiplication time. *Theoretical Computer Science (TCS)*, 410(44):4480–4488, 2009.

49 Shay Solomon. Fully dynamic maximal matching in constant update time. In *Proceedings of the 57th Symposium on Foundations of Computer Science (FOCS)*, pages 325–334, 2016.

50 Daniel Stubbs and Virginia Vassilevska Williams. Metatheorems for dynamic weighted matching. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS)*, 2017.

51 Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 384–396, 2004.

52 Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 112–119, 2005.

53 Vadim G Vizing. On an estimate of the chromatic class of a p-graph. *Diskret analiz*, 3:25–30, 1964.

54 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 887–898, 2012.

Tighter Connections Between Formula-SAT and Shaving Logs

Amir Abboud[1]
IBM Almaden Research Center, San Jose, USA
amir.abboud@ibm.com

Karl Bringmann
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
kbringma@mpi-inf.mpg.de

Abstract

A noticeable fraction of Algorithms papers in the last few decades improve the running time of well-known algorithms for fundamental problems by logarithmic factors. For example, the $O(n^2)$ dynamic programming solution to the Longest Common Subsequence problem (LCS) was improved to $O(n^2/\log^2 n)$ in several ways and using a variety of ingenious tricks. This line of research, also known as *the art of shaving log factors*, lacks a tool for proving negative results. Specifically, how can we show that it is unlikely that LCS can be solved in time $O(n^2/\log^3 n)$?

Perhaps the only approach for such results was suggested in a recent paper of Abboud, Hansen, Vassilevska W. and Williams (STOC'16). The authors blame the hardness of shaving logs on the hardness of solving satisfiability on boolean formulas (Formula-SAT) faster than exhaustive search. They show that an $O(n^2/\log^{1000} n)$ algorithm for LCS would imply a major advance in circuit lower bounds. Whether this approach can lead to tighter barriers was unclear.

In this paper, we push this approach to its limit and, in particular, prove that a well-known barrier from complexity theory stands in the way for shaving five additional log factors for fundamental combinatorial problems. For LCS, regular expression pattern matching, as well as the Fréchet distance problem from Computational Geometry, we show that an $O(n^2/\log^{7+\varepsilon} n)$ runtime would imply new Formula-SAT algorithms.

Our main result is a reduction from SAT on formulas of size s over n variables to LCS on sequences of length $N = 2^{n/2} \cdot s^{1+o(1)}$. Our reduction is essentially as efficient as possible, and it greatly improves the previously known reduction for LCS with $N = 2^{n/2} \cdot s^c$, for some $c \geq 100$.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

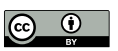
Keywords and phrases Fine-Grained Complexity, Hardness in P, Formula-SAT, Longest Common Subsequence, Frechet Distance

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.8

Related Version A full version can be found at `https://arxiv.org/abs/1804.08978`.

Acknowledgements Part of the work was performed while visiting the Simons Institute for the Theory of Computing, Berkeley, CA. We are grateful to Avishay Tal for telling us about his algorithm for SAT on bipartite formulas. We also thank Mohan Paturi, Rahul Santhanam, Srikanth Srinivasan, and Ryan Williams for answering our questions about the state of the art of Formula-SAT algorithms, and Arturs Backurs, Piotr Indyk, Mikkel Thorup, and Virginia

[1] The work was completed when A.A. was at Stanford University and was supported by Virginia Vassilevska Williams' NSF Grants CCF-1417238 and CCF-1514339, and BSF Grant BSF:2012338.



Vassilevska Williams for helpful discussions regarding regular expressions. We also thank an anonymous reviewer for ideas leading to shaving off a second log-factor for Formula-Pair.

1 Introduction

Since the early days of Algorithms research, a noticeable fraction of papers each year *shave log factors* for fundamental problems: they reduce the best known upper bound on the time complexity from $T(n)$ to $T(n)/\log^c n$, for some $c > 0$. While in some cases a cynic would call such results "hacks" and "bit tricks", there is no doubt that they often involve ingenious algorithmic ideas and suggest fundamental new ways to look at the problem at hand. In his survey, Timothy Chan calls this kind of research "The Art of Shaving Logs" [37]. In many cases, we witness a race of shaving logs for some problem, in which a new upper bound is found every few months, without giving any hints on when this race is going to halt. For example, in the last few years, the upper bound for combinatorial Boolean Matrix Multiplication dropped from $O(n^3/\log^2 n)$ [16], to $O(n^3/\log^{2.25} n)$ [20], to $O(n^3/\log^3 n)$ [38], and most recently to $O(n^3/\log^4 n)$ [99]. Perhaps the single most important missing technology for this kind of research is a tool for proving *lower bounds*.

Consider the problem of computing the Longest Common Subsequence (LCS) of two strings of length n. LCS has a simple $O(n^2)$ time dynamic programming algorithm [93, 46]. Several approaches have been utilized in order to shave log factors such as the "Four Russians" technique [16, 61, 74, 23, 58], utilizing bit-parallelism [10, 47, 62], and working with compressed strings [48, 54]. The best known upper bounds are $O(n^2/\log^2 n)$ for constant size alphabets [74], and $O(n^2 \log\log n/\log^2 n)$ for large alphabets [58]. But can we do better? Can we solve LCS in $O(n^2/\log^3 n)$ time? While the mathematical intrigue is obvious, we remark that even such mild speedups for LCS could be significant in practice. Besides its use as the *diff* operation in unix, LCS is at the core of highly impactful similarity measures between biological data. A heuristic algorithm called BLAST for a generalized version of LCS (namely, the Local Alignment problem [87]) has been cited more than sixty thousand times [14]. While such heurisitics are much faster than the near-quadratic time algorithms above, they are not guaranteed to return an optimal solution and are thus useless in many applications, and biologists often fall back to (highly optimized implementations of) the quadratic solutions, see, e.g. [71, 72].

How would one show that it is hard to shave logs for some problem? A successful line of work, inspired by NP-hardness, utilizes "fine-grained reductions" to prove statements of the form: a small improvement over the known runtime for problem A implies a breakthrough algorithm for problem B, refuting a plausible hypothesis about the complexity of B. For example, it has been shown that if LCS can be solved in $O(n^{2-\varepsilon})$ time, where $\varepsilon > 0$, then there is a breakthrough $(2-\delta)^n$ algorithm for CNF-SAT, and the Strong Exponential Time Hypothesis (SETH, defined below) is refuted [2, 29]. Another conjecture that has been used to derive interesting lower bounds states that the 3-SUM problem[2] cannot be solved in $O(n^{2-\varepsilon})$ time. It is natural to ask: can we use these conjectures to rule out log-factor improvements for problems like LCS? And even more optimistically, one might hope to base the hardness of LCS on a more standard assumption like $\mathsf{P} \neq \mathsf{NP}$. Unfortunately, we can formally prove that these assumptions are not sharp enough to lead to any consequences for log-factor improvements, if only Turing reductions are used. In Section 3 we prove the

[2] 3-SUM asks, given a list of n numbers, to find three that sum to zero. The best known upper bound is $O(n^2(\log\log n)^2/\log n)$ for real numbers [59, 53, 56] and $O(n^2(\log\log n/\log n)^2)$ for integers [21].

following theorem which also shows that an $O(f(n)/\log^c(f(n)))$ time algorithm for problem A cannot imply, via a fine-grained reduction, an $O(g(n)^{1-\varepsilon})$ algorithm for problem B, unless B is (unconditionally) solvable in $O(g(n)^{1-\delta})$ time.

▶ **Theorem 1.1** (Informally). *If for some $c > 0$ there is a fine-grained reduction proving that LCS is not in $O(n^2/\log^c n)$ time unless SETH fails, then SETH is false.*

Note that it also does not suffice to simply make SETH stronger by postulating a higher running time lower bound for CNF-SAT, since *superpolynomial improvements are known* for this problem [81, 34, 49, 8]. Similarly, we cannot base a study of log-factor improvements on the APSP conjecture, since *superlogarithmic improvements are known* for APSP [97]. (However, 3SUM could be a candidate to base higher lower bounds on, since only log-factor improvements are known [59, 53, 56, 21], see Section A of the full version for a discussion.)

Thus, in a time when super-linear lower bounds for problems like LCS are far out of reach, and our only viable approach to obtaining such negative results is reductions-based, we are left with two options. We could either leave the study of log-factor improvements in limbo, without a technology for proving negative results, or we could search for natural and convincing assumptions that are more fine-grained than SETH that could serve as the basis for the negative results we desire. Such assumptions were recently proposed by Abboud, Hansen, Vassilevska Williams and Williams [3]. The authors blame the hardness of shaving logs on the hardness of solving satisfiability on boolean formulas (Formula-SAT) faster than exhaustive search[3], by polynomial factors (which are log-factors in the runtime), a task for which there are well known "circuit lower bound" barriers in complexity theory. They show that an $O(n^2/\log^{1000} n)$ algorithm for LCS would imply a major advance in circuit lower bounds. In the final section of this paper, we give a more detailed argument in favor of this approach. Whether one should expect it to lead to *tight* barriers, i.e. explaining the lack of $O(n^2/\log^3 n)$ algorithms for LCS or any other natural problem, was completely unclear.

The Machine Model

We use the *Word-RAM* model on words of size $\Theta(\log n)$, where there is a set of operations on words that can be performed in time $O(1)$. Most papers do not fix the concrete set of allowed operations, and instead refer to "typical Boolean and arithmetic operations". In this paper, we choose a set of operations $\mathcal{P}$ that is *robust* with respect to changing the word size: For any operation $\circ \in \mathcal{P}$, given two words a, b (of size $\Theta(\log n)$) we can compute $a \circ b$ in time $(\log n)^{1+o(1)}$ on a Word RAM with word size $\Theta(\log \log n)$ and operation set $\mathcal{P}$. In other words, if we split a, b into $\Theta(\log n/\log \log n)$ words of size $\Theta(\log \log n)$ then $a \circ b$ can still be computed very efficiently.

This robustness in particular holds for the following standard set of operations: initializing a cell with a constant, bitwise AND, OR, NOT, shift, addition, subtraction, multiplication, and division with remainder (since multiplication and division have near-linear time algorithms).

The results in this paper will get gradually weaker as we relax the restriction on near-linear time per operation to higher runtimes, however, even with this restriction, to the best of our knowledge this model captures all log shaving results in the literature (on the "standard" Word RAM model without fancy word operations).

[3] In [3] the authors focus on SAT on Branching Programs (BPs) rather than formulas, but due to standard transformations between BPs and formulas, the two problems are equivalent up to polynomial factors. Focusing on Formula-SAT will be crucial to the progress we make in this paper.

Formula-SAT

A boolean formula over n input variables can be viewed as a tree in which every leaf is marked by an input variable or its negation and every internal node or *gate* represents some basic boolean operation. Throughout this introduction we will only talk about *deMorgan* formulas, in which every gate is from the set $\{\wedge, \vee\}$. The *size* of the formula is defined to be the number of leaves in the tree.

In the Formula-SAT problem we are given a formula F of size s over n inputs, and we have to decide whether there is an input $\{0,1\}^n$ that makes it output 1. A naive algorithm takes $O(2^n \cdot s)$ time, since evaluating the formula on some input takes $O(s)$ time. Can we do better? We will call a SAT algorithm *non-trivial*[4] if it has a runtime at most $O(\frac{2^n}{n^\varepsilon})$, for some $\varepsilon > 0$.

It seems like a clever algorithm must look at the given formula F and try to gain a speedup by analyzing it. The more complicated F can be, the harder the problem becomes. Indeed, Dantsin and Hirsch [49] survey dozens of algorithms for SAT on CNF formulas which exploit their structure. For k-CNF formulas of size s there are $2^n s/2^{\Omega(n/k)}$ time algorithms (e.g. [81]), and for general CNF formulas the bound is $2^n s/2^{\Omega(n/\log \Delta)}$ where $\Delta = s/n$ is the clause-to-variable ratio [34, 49, 8]. The popular SETH [66, 35] essentially says that this is close to optimal, and that there is no $2^n s/2^{\Omega(n)}$ algorithm for CNF-SAT. For arbitrary deMorgan formulas, the upper bounds are much worse. A FOCS'10 paper by Santhanam [84] and several recent improvements [41, 43, 42, 70, 91] solve Formula-SAT on formulas of size $s = n^{3-16\varepsilon}$ in time $2^n s^{O(1)}/2^{n^\varepsilon}$, which is non-trivial only for $s = o(n^3)$, and going beyond cubic seems extremely difficult. This leads us to the first barrier which we will transform into a barrier for shaving logs.

▶ **Hypothesis 1.2.** *There is no algorithm that can solve SAT on deMorgan formulas of size $s = n^{3+\Omega(1)}$ in $O(\frac{2^n}{n^\varepsilon})$ time, for some $\varepsilon > 0$, in the Word-RAM model.*

Perhaps the main reason to believe this hypothesis is that despite extensive algorithmic attacks on variants of SAT (perhaps the most extensively studied problem in computer science) over decades, none of the ideas that anyone has ever come up with seem sufficient to refute it. Recent years have been particularly productive in non-trivial algorithms designed for special cases of Circuit-SAT [84, 86, 64, 35, 98, 22, 40, 67, 63, 44, 83, 57] (in addition to the algorithms for deMorgan formulas above) and this hypothesis still stands.

A well-known "circuit lower bounds" barrier seems to be in the way for refuting Hypothesis 1.2: can we find an explicit boolean function that cannot be computed by deMorgan formulas of cubic size? Functions that require formulas of size $\Omega(n^{1.5})$ [89] and $\Omega(n^2)$ [69] have been known since the 60's and 70's, respectively. In the late 80's, Andreev [15] proved an $\Omega(n^{2.5})$ which was later gradually improved to $\Omega(n^{2.55})$ by Nisan and Wigderson [65] and to $\Omega(n^{2.63})$ by Paterson and Zwick [79] until Håstad proved his $n^{3-o(1)}$ lower bound in FOCS'93 [60] (a recent result by Tal improves the $n^{o(1)}$ term [90]). All these lower bound results use the "random restrictions" technique, first introduced in this context by Subbotovskaya in 1961 [89], and it is known that a substantially different approach must be taken in order to go beyond the cubic barrier. What does this have to do with Formula-SAT algorithms? Interestingly, this same "random restrictions" technique was crucial to *all* the non-trivial Formula-SAT algorithms mentioned above. This is not a coincidence, but only one out of

[4] Some works on SAT algorithms used this term for runtimes of the form $2^n \mathrm{poly}(s)/n^{\omega(1)}$. In our context, we need to be a bit more fine-grained.

the many examples of the intimate connection between the task of designing non-trivial algorithms for SAT on a certain class $\mathcal{F}$ of formulas or circuits and the task of proving lower bounds against $\mathcal{F}$. This connection is highlighted in many recent works and in several surveys [85, 78, 96]. The intuition is that both of these tasks seem to require identifying a strong structural property of functions in $\mathcal{F}$. There is even a formal connection shown by Williams [95], which in our context implies that solving Formula-SAT on formulas of size $O(n^{3.1})$ in $O(2^n/n^{10})$ time (which is only slightly stronger than refuting Hypothesis 1.2) is sufficient in order to prove that there is a function in the class E^{NP} that cannot be computed by formulas of size $O(n^{3.1})$ (see [3] for more details). This consequence would be the first polynomial progress on the fundamental question of worst case formula lower bounds since Håstad's result.

1.1 Our Results: New Reductions

Many recent papers have reduced CNF-SAT to fundamental problems in P to prove SETH-based lower bounds (e.g. [80, 82, 6, 4, 27, 18, 7, 1, 33, 5, 19, 75, 39]). Abboud et al. [3] show that even SAT on formulas, circuits, and more, can be efficiently reduced to combinatorial problems in P. In particular, they show that Formula-SAT on formulas of size s over n inputs can be reduced to an instance of LCS on sequences of length $N = O(2^{n/2} \cdot s^{1000})$. This acts as a barrier for shaving logs as follows. A hypothetical $O(N^2/\log^c N)$ time algorithm for LCS can be turned into an algorithm for Formula-SAT in time

$$n^{1+o(1)} \cdot (2^{n/2} \cdot s^{1000})^2/(\log 2^{\Omega(n)})^c = O(2^n \cdot s^{2000}/n^{c-1}),$$

which for a large enough $c \geq 2001$ would refute Hypothesis 1.2. The first $n^{1+o(1)}$ factor in the runtime comes from the jump from n to $N = 2^n$ and our Word-RAM machine model: whenever the LCS algorithm wants to perform a unit-cost operation on words of size $\Theta(\log N)$ (this is much more than the word size of our SAT algorithm which is only $\Theta(\log n) = \Theta(\log \log N)$), the SAT algorithm can simulate it in $(\log N)^{1+o(1)} = n^{1+o(1)}$ time in the Word-RAM model with words of size $\Theta(\log n)$.

Our main result is a *much* more efficient reduction to LCS. For large but constant size alphabets, we get a near-linear dependence on the formula size, reducing the s^{1000} factor to just $s^{1+o(1)}$.

▶ **Theorem 1.3.** *Formula-SAT on formulas of size s on n inputs can be reduced to an instance of LCS on two sequences over an alphabet of size σ of length $N = 2^{n/2} \cdot s^{1+O(1/\log\log\sigma)}$, in $O(N)$ time.*

Thus, if LCS on sequences of length N and alphabet of size $\omega(1)$ can be solved in $O(N^2/\log^c N)$ time, then Formula-SAT can be solved in $2^n \cdot \frac{s^{2+o(1)}}{n^c} \cdot n^{1+o(1)}$ time. Recall that the known upper bound for LCS is $O(n^2/\log^c n)$ for any constant alphabet size, with $c = 2$, and we can now report that the barrier of cubic formulas stands in the way of improving it to $c > 7$ (see Corollary 1.6 below).

The novelty in the proof of Theorem 1.3 over [3] is discussed in Section 2. As an alternative to Theorem 1.3, in Section D of the full version we present another reduction to LCS which is much simpler than all previously known reductions, but uses a larger alphabet.

Fréchet Distance

An important primitive in computational geometry is to judge how similar are two basic geometric objects, such as polygonal curves, represented as sequences of points in d-dimensional Euclidean space. Such curves are ubiquitous, since they arise naturally as trajectory data of

moving objects, or as time-series data of stock prices and other measures. The most popular similarity measure for curves in computational geometry is the Fréchet distance, also known as dog-leash-distance. For formal definitions see Section F of the full version. The Fréchet distance has found many applications (see, e.g., [76, 26, 30]) and developed to a rich field of research with many generalizations and variants (see, e.g., [11, 17, 13, 51, 36, 45, 32, 50, 73, 68]).

This distance measure comes in two variants: the continuous and the discrete. A classic algorithm by Alt and Godau [12, 55] computes the continuous Fréchet distance in time $O(n^2 \log n)$ for two given curves with n vertices. The fastest known algorithm runs in time $O(n^2(\log\log n)^2)$ (on the Word RAM) [31]. If we only want to decide whether the Fréchet distance is at most a given value δ, this algorithm runs in time $O(n^2(\log\log n)^2/\log n)$. For the discrete Fréchet distance, the original algorithm has running time $O(n^2)$ [52], which was improved to $O(n^2 \log\log n/\log n)$ by Agarwal et al. [9]. Their algorithm runs in time $O(n^2 \log\log n/\log^2 n)$ for the decision version. It is known that both versions of the Fréchet distance are SETH-hard [27]. However, this does not rule out log factor improvements. In particular, no reduction from versions of SETH on formulas or branching programs is known.

In this paper we focus on the *decision version of the discrete Fréchet distance* (which we simply call "Fréchet distance" from now on). We show that Fréchet distance suffers from the same barriers for shaving logs like LCS. In particular, this reduction allows us to base the usual $\Omega(n^{2-\varepsilon})$ lower bound on a weaker assumption than SETH, such as NC-SETH (see the discussion in [3]). This is the first NC-SETH hardness for a problem that does not admit alignment gadgets (as in [29]).

▶ **Theorem 1.4.** *Formula-SAT on formulas of size s on n inputs can be reduced to an instance of the Fréchet distance on two curves of length $N = O(2^{n/2} \cdot s)$, in $O(N)$ time.*

Regular Expression Pattern Matching

Our final example is the fundamental *Regular Expression Pattern Matching* problem: Decide whether a given regular expression of length m matches a substring of a text of length n. Again, there is a classical $O(nm)$ algorithm [92], and the applicability and interest in this problem resulted in algorithms shaving log factors; the first one by Myers [77] was improved by Bille and Thorup [24] to time $O(mn/\log^{1.5} n)$. Recently, Backurs and Indyk proved an $n^{2-o(1)}$ SETH lower bound [19], and performed an impressive study of the exact time complexity of the problem with respect to the complexity of the regular expression. This study was essentially completed by Bringmann, Grønlund, and Larsen [28], up to $n^{o(1)}$ factors. In Section E of the full version we show that this problem is also capable of efficiently simulating formulas and thus has the same barriers as LCS and Fréchet distance.

▶ **Theorem 1.5.** *Formula-SAT on formulas of size s on n inputs can be reduced to an instance of Regular Expression Pattern Matching on text and pattern of length $N = O(2^{n/2} \cdot s \log s)$ over a constant size alphabet, in $O(N)$ time.*

Consequences of the Cubic Formula Barrier

We believe that SAT on formulas can be tightly connected to many other natural problems in P. As we discuss in the next section, such reductions seem to require problem-specific engineering and are left for future work. The main point of this paper is to demonstrate the possibility of basing such ultra fine-grained lower bounds on one common barrier. Our conditional lower bounds are summarized in the following corollary, which shows that current log-shaving algorithms are very close to the well-known barrier from complexity theory of cubic formula lower bounds.

▶ **Corollary 1.6.** *For all $\varepsilon > 0$, solving any of the following problems in $O(n^2/\log^{7+\varepsilon} n)$ time refutes Hypothesis 1.2, and solving them in $O(n^2/\log^{17+\varepsilon} n)$ time implies that E^{NP} cannot be computed by non-uniform formulas of cubic size:*

- *LCS over alphabets of size $\omega(1)$*
- *The Fréchet distance on two curves in the plane*
- *Regular Expression Pattern Matching over constant size alphabets.*

The main reason that our lower bounds above are not tight (the gap between 2 and 7) is that we need to start from SAT on *cubic* size formulas rather than linear size ones, due to the fact that clever algorithms do exist for smaller formulas. We remark that throughout the paper we will work with a class of formulas we call $\mathcal{F}_1$ (see Section 2 below), also known as bipartite formulas, that are more powerful than deMorgan formulas yet our reduction to LCS can support them as well. This makes our results stronger, since $\mathcal{F}_1$-Formula-SAT could be a harder problem than SAT on deMorgan formulas. In fact, in an earlier version of the paper we had suggested the hypothesis that $\mathcal{F}_1$-Formula-SAT does not have non-trivial algorithms even on *linear size* formulas. This stronger hypothesis would give higher lower bounds. However, Avishay Tal (personal communication) told us about such a non-trivial algorithm for formulas of size up to $n^{2-\Omega(1)}$ using tools from quantum query complexity. We are optimistic that one could borrow such ideas or the "random restrictions" technique from SAT algorithms in order to shave more logs for combinatorial problems such as LCS. This is an intriguing direction for future work.

2 Technical Overview and the Reduction to LCS

We first define the class of frmulas $\mathcal{F}_1$. A formula F of size s over n variables $x_1, \ldots, x_n$ is in the class $\mathcal{F}_1$ iff it has the following properties. The gates in the first layer (nodes in the tree whose children are all leaves) compute arbitrary functions $C : \{0,1\}^{n/2} \to \{0,1\}$, as long as C can be computed in $2^{o(n)}$ time and all children of a gate are marked with variables in $\{x_1, \ldots, x_{n/2}\}$ or with variables in $\{x_{n/2+1}, \ldots, x_n\}$ but not with both. W.l.o.g. we can assume that the inputs are only connected to nodes in the first layer. The gates in the other layers compute deMorgan gates, i.e., OR and AND gates. The size of F is considered to be the *number of gates in the first layer*. Since F is a formula and thus has fanout 1, our size measure is up to constant factors equal to the total number of all gates except the inputs. Note that the complexity of the functions in the first layer and their number of incoming wires, i.e. the number of leaves in the tree, do not count towards the size of F.

All the reductions from SAT to problems in P mentioned in the introduction start with a split-and-list reduction to some "pair finding" problem. In the SETH lower bounds, CNF-SAT is reduced to the Orthogonal-Vectors problem of finding a pair $a \in A, b \in B, A, B \subseteq \{0,1\}^d$ that are orthogonal [94]. When starting from Formula-SAT, we get a more complex pair-finding problem. In Section B of the full version we show a simple reduction from SAT on formulas from the class $\mathcal{F}_1$ (which contains deMorgan formulas) to the following problem.

▶ **Definition 2.1** (Formula-Pair Problem)**.** Given a deMorgan formula over $2m$ variables $F = F(x_1, \ldots, x_m, y_1, \ldots, y_m)$ (each appearing once in F), and two sets of vectors $A, B \subseteq \{0,1\}^m$ of size n, decide if there is a pair $a \in A, b \in B$ such that $F(a_1, \ldots, a_m, b_1, \ldots, b_m) =$ true.

In Section B of the full version we show a Four-Russians type algorithm that solves Formula-Pair in $O(n^2 m/\log^2 n)$ time, and even when $m = |F| = (\log n)^{1+o(1)}$ no $O(n^2/\log^{1+\varepsilon} n)$ upper bound is known. By our reduction, such an upper bound would

imply a non-trivial algorithm for SAT on formulas from $\mathcal{F}_1$. Moreover, Hypothesis 1.2 implies that we cannot solve Formula-Pair in $O(n^2/\log^{\varepsilon} n)$ time, for $m = (\log n)^{3+\Omega(1)}$. In the next sections, we reduce Formula-Pair to LCS, from which Theorem 1.3 follows. A simpler reduction using much larger alphabet size can be found in Section D of the full version.

▶ **Theorem 2.2.** Formula-Pair *on formulas of size s and lists of size n can be reduced to an instance of LCS on two strings over alphabet of size $\sigma \geq 2$ of length $O(n \cdot s^{1+O(1/\log\log\sigma)})$, in linear time.*

The reduction constructs strings x, y and a number ρ such that $\mathrm{LCS}(x, y) \geq \rho$ holds if and only if the given Formula-Pair instance (F, A, B) is satisfiable. The approach is similar to the reductions from Orthogonal-Vectors to sequence alignment problems (e.g. [6, 27, 18, 2, 29]). The big difference is that our formula F can be much more complicated than a CNF, and so we will need more powerful gadgets. Sequence gadgets that are able to simulate the evaluation of deMorgan formulas were (implicitly) constructed in [3] with a recursive approach. Our main contribution is an *extremely efficient* implementation of such gadgets with LCS.

The main part of the reduction is to construct *gate gadgets*: for any vectors $a, b \in \{0,1\}^m$ and any gate g of F, we construct strings $x(g, a)$ and $y(g, b)$ whose LCS determines whether gate g evaluates to true for input (a, b) to F (see Section 2.1). Once we have this, to find a pair of vectors $a \in A, b \in B$ satisfying F, we combine the strings $x(r, a), y(r, b)$, constructed for the root r of F, using a known construction of so-called alignment gadgets [2, 29] from previous work (see Section C.1 of the full version).

Let us quickly explain how [3] constructed gate gadgets and the main ideas that go into our new construction. There are two kinds of gadgets, corresponding to the two types of gates in F: AND and OR gates. Since the AND gadgets will be relatively simple, let us consider the OR gadgets. Fix two inputs a, b, and let $g = (g_1 \vee g_2)$ be an OR gate, and assume that we already constructed gate gadgets for g_1, g_2, namely $x_1 = x(g_1, a), y_1 = y(g_1, b), x_2 = x(g_2, a), y_2 = y(g_2, b)$ so that for $i \in \{1, 2\}$ we have that $\mathrm{LCS}(x_i, y_i)$ is large if the gate g_i outputs true on input (a, b), and it is smaller otherwise. In [3], these gadgets were combined as follows. Let β be an upper bound on the total length of the gadgets x_i, y_i. We add a carefully chosen padding of 0's and 1's, so that any optimal matching of the two strings will have to match *either* x_1, y_1 or x_2, y_2 but not both.

$$\begin{aligned} x &:= \quad 0^{4\beta}\ x_1\ 1^{\beta}\ x_2\ 0^{4\beta} \\ y &:= \quad y_1\ 1^{\beta} 0^{4\beta} 1^{\beta}\ y_2 \end{aligned}$$

One then argues that, in any optimal LCS matching of x, y, the $0^{4\beta}$ block of y must be matched either left or right. If it's matched left, then the total score will be equal to $4\beta + \beta + LCS(x_2, y_2)$ while if it's matched right, we will get $4\beta + \beta + LCS(x_1, y_1)$. Thus, $LCS(x, y)$ is determined by the OR of g_1, g_2. The blowup of this construction is a multiplicative factor of 11 with every level of the formula, and the length of the gadget of the root will end up roughly $11^{\mathrm{depth}(F)}$. To obtain our tight lower bounds, we will need to decrease this blowup to $1 + \varepsilon_\sigma$ at every level, where ε_σ goes to 0 when the alphabet size σ tends to infinity. With the above construction, decreasing the length of the padding will allow the optimal LCS matching to cheat, e.g. by matching y_1 to both x_1 and x_2, and no longer corresponding to the OR of g_1, g_2.

Our first trick is an ultra-efficient OR gadget in case we are allowed unbounded alphabet size. We take x_1, y_1 and transform all their letters into a new alphabet Σ^{g_1}, and we take x_2, y_2 and transform their letters into a disjoint alphabet Σ^{g_2}. Then our OR gadget does not

require any padding at all:

$$x := x_1\ x_2$$
$$y := y_2\ y_1$$

The crossing structure of this construction means that any LCS matching that matches letters from x_1, y_1 cannot also match letters from x_2, y_2, and vice versa, while the disjoint alphabets make sure that there can be no matches between x_1, y_2 or x_2, y_1. With such gadgets we can encode a formula of size s with $O(s)$ letters, for details see Section D of the full version.

But how would such an idea work for *constant size alphabets*? Once we allow x_1 and y_2 to share even a single letter, this argument breaks. Natural attempts to simulate this construction with smaller alphabets, e.g. by replacing each letter with a random sequence, do not seem to work, and we do not know how to construct such an OR gadget with a smaller alphabet *in a black box way*. The major part of our proof will be a careful examination of the formula and the sub-gadgets g_1, g_2 in order to reduce the alphabet size to a large enough constant, while using padding that is only $1 + \varepsilon_\sigma$ times the length of the sub-gadgets. We achieve this by combining this crossing gadget with a small padding that will reuse letters from alphabets that were used much deeper in the formula, and we will argue that the noise we get from recycling letters is dominated by our paddings, in any optimal matching.

We remark that the reduction of [3] can be implemented in a generic way with any problem that admits *alignment gadgets* as defined in [29], giving formula-gadgets of size $s^{O(1)}$. The list of such problems includes LCS and Edit-Distance on binary strings. However, to get gadgets of length $s^{1+o(1)}$ it seems that problem-specific reductions are necessary. A big open question left by our work is to find the most efficient reduction from Formula-SAT to Edit-Distance. A very efficient OR gadget, even if the alphabet is unbounded, might be (provably) impossible. Can we use this intuition to shave more log factors for Edit-Distance?

Fréchet Distance falls outside the alignment gadgets framework of [29] and no reduction from Formula-SAT was known before. In Section F of the full version we prove such a reduction by a significant boosting of the SETH-lower bound construction of [27]. In order to implement recursive AND/OR gadgets, our new proof utilizes the geometry of the curves, in contrast to [27] which only used ten different points in the plane.

In the remainder of this section we present the details of the reduction to LCS. Some missing proofs can be found in Section C of the full version.

2.1 Implementing Gates

Fix vectors $a, b \in \{0,1\}^m$ (where $2m$ is the number of inputs to F). In this section we prove the following lemma which demonstrates our main construction.

▶ **Lemma 2.3.** *For any sufficiently large $\sigma > 0$ let $\tau = (\log \sigma)^{1/4}$. We can inductively construct, for each gate g of F, strings $x(g) = x(g,a)$ and $y(g) = y(g,b)$ over alphabet size $5\sigma^2$ and a number $\rho(g)$ such that for $L(g) := \mathrm{LCS}(x(g), y(g))$ we have (1) $L(g) \leq \rho(g)$ and (2) $L(g) = \rho(g)$ if and only if gate g evaluates to true on input (a,b) to F. Moreover, we have $|x(g)| = |y(g)| = n(g) \leq 6\tau \cdot |F_g|(1 + 7/\tau)^{\mathrm{depth}(F_g)}$, where F_g is the subformula of F below g.*

In this construction, we use disjoint size-5 alphabets $\Sigma_1, \ldots, \Sigma_{\sigma^2}$, determining the total alphabet size as $5\sigma^2$. Each gate g is assigned an alphabet $\Sigma_{f(g)}$. We fix the function f later.

In the following, consider any gate g of F, and write the gate alphabet as $\Sigma_{f(g)} = \{0,1,2,3,4\}$. For readability, we write $x = x(g)$ and similarly define y, n, L, ρ. If g has fanin 2, write g_1, g_2 for the children of g. Moreover, let $x_1 = x(g_1)$ and similarly define y_1, n_1, L_1, ρ_1 and $x_2, y_2, n_2, L_2, \rho_2$.

Input Gate

The base case is an input bit a_i to F (input bits b_j are symmetric). Interpreting a_i as a string of length 1 over alphabet $\{0,1\}$, note that $\mathrm{LCS}(a_i, 1) = a_i$. Hence, the strings $x = a_i$ and $y = 1$, with $n = \rho = 1$, trivially simulate the input bit a_i.

AND Gates

Consider an AND gate g and let $\beta := \lceil (n_1 + n_2)/\tau^2 \rceil$. We construct strings x, y as

$$\begin{aligned} x &:= x_1\, 0^\beta\, 1^\beta\, x_2 \\ y &:= y_1\, 0^\beta\, 1^\beta\, y_2 \end{aligned}$$

▶ **Lemma 2.4.** *If* $\mathrm{LCS}(x_2, y_1), \mathrm{LCS}(x_1, y_2) \leq \beta/4$ *and the symbols* $0, 1$ *appear at most* $\beta/16$ *times in each of* x_1, x_2, y_1, *and* y_2, *then we have* $L = \mathrm{LCS}(x, y) = 2\beta + L_1 + L_2$.

Later we will choose the gate alphabets $\Sigma_{f(g)}$ such that the precondition of the above lemma is satisfied. Setting $\rho := 2\beta + \rho_1 + \rho_2$ we thus inductively obtain (1) $L \leq \rho$ and (2) $L = \rho$ if and only if g_1 and g_2 both evaluate to true. Thus, we correctly simulated the AND gate g. It remains to prove the lemma.

Proof. Clearly, we have $L \geq \mathrm{LCS}(x_1, y_1) + \mathrm{LCS}(0^\beta, 0^\beta) + \mathrm{LCS}(1^\beta, 1^\beta) + \mathrm{LCS}(x_2, y_2) = 2\beta + L_1 + L_2$. For the other direction, consider any LCS z of x, y. If z does not match any symbol of the left half of x, $x_1 0^\beta$, with any symbol of the right half of y, $1^\beta y_2$, and it does not match any symbol of the right half of x, $1^\beta x_2$, with any symbol of the left half of y, $y_1 0^\beta$, then we can split both strings in the middle and obtain $L = |z| \leq \mathrm{LCS}(x_1 0^\beta, y_1 0^\beta) + \mathrm{LCS}(1^\beta x_2, 1^\beta y_2)$. Greedy suffix/prefix matching now yields $L \leq \big(\mathrm{LCS}(x_1, y_1) + \beta\big) + \big(\beta + \mathrm{LCS}(x_2, y_2)\big) = 2\beta + L_1 + L_2$.

In the remaining case, there is a matching from some left half to some right half. By symmetry, we can assume that there is a matching from the left half of x to the right half of y. We can moreover assume that z matches a symbol of x_1 with a symbol of $1^\beta y_2$, since the case that z matches a symbol of y_2 with a symbol of $x_1 0^\beta$ is symmetric. Now no symbol in 0^β in x can be matched with a symbol in 0^β in y. We obtain a rough upper bound on $L = |z|$ by summing up the LCS length of all remaining $4 \cdot 4 - 1 = 15$ pairs of a part $x' \in \{x_1, 0^\beta, 1^\beta, x_2\}$ in x and a part $y' \in \{y_1, 0^\beta, 1^\beta, y_2\}$ in y. This yields $L \leq L_1 + L_2 + \beta + 2 \cdot \beta/4 + 8 \cdot \beta/16 = 2\beta + L_1 + L_2$, finishing the proof. ◀

OR Gates

Consider an OR gate g and again let $\beta := \lceil (n_1 + n_2)/\tau^2 \rceil$. We first make the LCS target values equal by adding $4^{|\rho_1 - \rho_2|}$ to the shorter of x_2/y_2 and x_1/y_1, i.e., we set $x_1' := 4^{\max\{0, \rho_2 - \rho_1\}} x_1$ and similarly $y_1' := 4^{\max\{0, \rho_2 - \rho_1\}} y_1$, $x_2' := 4^{\max\{0, \rho_1 - \rho_2\}} x_2$, $y_2' := 4^{\max\{0, \rho_1 - \rho_2\}} y_2$. Note that the resulting strings satisfy $L_1' := \mathrm{LCS}(x_1', y_1') \leq \rho' := \max\{\rho_1, \rho_2\}$ and $L_1' = \rho'$ if and only if g_1 evaluates to true, and similarly $L_2' := \mathrm{LCS}(x_2', y_2') \leq \rho'$ and $L_2' = \rho'$ if and only if g_2 evaluates to true. We construct the strings x, y as

$$\begin{aligned} x &:= 0^\beta 1^\beta\, x_1'\, 2^\beta 3^\beta\, x_2'\, 0^\beta 1^\beta \\ y &:= 2^\beta 3^\beta\, y_2'\, 0^\beta 1^\beta\, y_1'\, 2^\beta 3^\beta \end{aligned}$$

▶ **Lemma 2.5.** *If* $\mathrm{LCS}(x_2, y_1), \mathrm{LCS}(x_1, y_2) \leq \beta/8$ *and the symbols* $0, 1, 2, 3$ *appear at most* $\beta/48$ *times in each of* x_1, x_2, y_1, *and* y_2, *then* $L = \mathrm{LCS}(x, y) = 4\beta + \max\{L_1', L_2'\}$.

Later we will choose the gate alphabets $\Sigma_{f(g)}$ such that the precondition of the above lemma is satisfied. Setting $\rho := 4\beta + \rho' = 4\beta + \max\{\rho_1, \rho_2\}$ we thus inductively obtain (1) $L \leq \rho$ and (2) $L = \rho$ if and only if at least one of g_1 and g_2 evaluates to true, so we correctly simulated the OR gate g. The proof of the Lemma is in Section C of the full version.

Analyzing the Length

Note that the above constructions inductively yields strings $x(g), y(g)$ simulating each gate g. We inductively prove bounds for $n(g)$ and $\rho(g)$. See Section C of the full version.

▶ **Lemma 2.6.** *We have* $n(g) \leq 6\tau \cdot |F_g|(1+7/\tau)^{\mathrm{depth}(F_g)}$ *and* $\rho(g) \leq 6|F_g|(1+7/\tau)^{\mathrm{depth}(F_g)}$ *for any gate* g*, where* F_g *is the subformula of* F *below* g*.*

Fixing the Gate Alphabets

Now we fix the gate alphabet $\Sigma_{f(g)}$ for any gate g. Again let $\Sigma_{(i,j)}$, $i, j \in [\sigma]$, be disjoint alphabets of size 5, and let $\Sigma := \bigcup_{i,j} \Sigma_{(i,j)}$. For any gate g of F, we call its distance to the root the *height* $h(g)$. For any h, order the gates with height h from left to right, and let $\iota(g)$ be the index of gate g in this order, for any gate g with height h. Note that $(h(g), \iota(g))$ is a unique identifier of gate g. We define $f(g) := (h(g) \bmod \sigma, \iota(g) \bmod \sigma)$, i.e., we set the gate alphabet of g to $\Sigma_{f(g)} = \Sigma_{(h(g) \bmod \sigma, \iota(g) \bmod \sigma)}$. Note that the overall alphabet Σ has size $5\sigma^2$. Recall that we set $\tau := (\log \sigma)^{1/4}$.

It remains to show that the preconditions of Lemmas 2.4 and 2.5 are satisfied. Specifically, consider a gate g with children g_1, g_2. As before, let x, y, n be the strings and string length constructed for gate g, and let x_i, y_i, n_i be the corresponding objects for g_i, $i \in \{1, 2\}$. We need to show:

(1) $\mathrm{LCS}(x_2, y_1), \mathrm{LCS}(x_1, y_2) \leq (n_1 + n_2)/(8\tau^2)$, and
(2) each $c \in \Sigma_{f(g)}$ appears at most $(n_1 + n_2)/(48\tau^2)$ times in each of x_1, x_2, y_1, and y_2.

We call a gate g' in the subformula F_g *d-deep* if $h(g') \geq h(g) + d$, and *d-shallow* otherwise. For each symbol c in x or y we can trace our construction to find the gate g' in F_g at which we introduced c to x or y. In other words, each symbol in x, y *stems* from some gate g' below g.

First consider (2). Observe that all symbols in x, y stemming from σ-shallow gates do not belong to the gate alphabet $\Sigma_{f(g)}$, since the function $f(g')$ has $(h(g') \bmod \sigma)$ as the first component, which repeats only every σ levels. Thus, if a symbol $c \in \Sigma_{f(g)}$ occurs in x_i or y_i, then this occurence stems from a σ-deep gate. We now argue that only few symbols in x, y stem from deep gates. For any $d > 0$, let N_d be the number of symbols in x (or, equivalently, y) steming from d-deep gates. Note that N_d is equal to the total string length $\sum n(g')$, summed over all gates g' in F_g with height $h(g') = h(g) + d$. Observe that our construction increases the string lengths in each step by at least a factor $1 + 1/\tau^2$, i.e., $N_d \geq (1 + 1/\tau^2)N_{d+1}$ holds for any d. It follows that $N_\sigma \leq N_1/(1 + 1/\tau^2)^{\sigma-1} = (n_1 + n_2)/(1 + 1/\tau^2)^{\sigma-1}$. Hence, each symbol in $\Sigma_{f(g)}$ appears at most $(n_1+n_2)/(1+1/\tau^2)^{\sigma-1}$ times in each of x_1, x_2, y_1, y_2. Since $\tau = (\log \sigma)^{1/4}$, we have $(1 + 1/\tau^2)^{\sigma-1} = 2^{\Omega(\sigma/\sqrt{\log \sigma})} \geq 48\sqrt{\log \sigma} = 48\tau^2$ for sufficiently large σ. This proves (2).

For (1), remove all $\log(\sigma)$-deep symbols from x_1 and y_2 to obtain strings x_1', y_2'. Note that we removed exactly $N_{\log \sigma}$ symbols from each of x_1, y_2. This yields $\mathrm{LCS}(x_1, y_2) \leq 2N_{\log \sigma} + \mathrm{LCS}(x_1', y_2')$. For x_1', y_2', we claim that any $\log(\sigma)$-shallow gates $g_1' \neq g_2'$ in F_g have disjoint alphabets $\Sigma_{f(g_1')}, \Sigma_{f(g_2')}$. Indeed, if $h(g_1') \neq h(g_2')$ then since the first component $(h(g') \bmod \sigma)$ of $f(g')$ repeats only every σ levels we have $f(g_1') \neq f(g_2')$. If $h(g_1') = h(g_2') =: h$,

then note that each gate g' in height h has a unique label $\iota(g') \bmod \sigma$, since there are σ such labels and there are at most $2^{h-h(g)} < \sigma$ gates with height h in F_g. Hence, x'_1 and y'_2 use disjoint alphabets, and we obtain $\mathrm{LCS}(x'_1, y'_2) = 0$. Thus, $\mathrm{LCS}(x_1, y_2) \leq 2N_{\log\sigma}$. As above, we bound $N_{\log\sigma} \leq (n_1+n_2)/(1+1/\tau^2)^{\log\sigma-1}$, so that $\mathrm{LCS}(x_1, y_2) \leq 2(n_1+n_2)/(1+1/\tau^2)^{\log\sigma-1}$. Since $\tau = (\log\sigma)^{1/4}$, we have $(1+1/\tau^2)^{\log\sigma-1}/2 = 2^{\Omega(\sqrt{\log\sigma})} \geq 8\sqrt{\log\sigma} = 8\tau^2$ for sufficiently large σ. This yields (1), since the strings x_2, y_1 are symmetric, finishing the proof of Lemma 2.3.

Finalizing the Proof

Let us sketch how we complete the proof of Theorem 2.2. The full details are in Section C.1 of the full version. First, for all vectors $a \in A, b \in B$ we construct gate gadgets for the output gate of the formula, i.e. *formula gadgets*, by invoking Lemma 2.3. Then we combine all these gadgets by applying a standard alignment gadget [2, 29] to get our final sequences of length $O\big(n\tau|F|(1+7/\tau)^{\mathrm{depth}(F)}\big)$ and with alphabet of size $O(\sigma^2)$. The LCS of the final sequence will be determined by the existence of a satisfying pair. Since a priori the depth of F could be as large as $|F|$, the factor $(1+7/\tau)^{\mathrm{depth}(F)}$ in our length bound is not yet satisfactory. Thus, as a preprocessing before the above construction, we decrease the depth of F using a depth-reduction result of Bonet and Buss [88, 25]: for all $k \geq 2$ there is an equivalent formula F' with depth at most $(3k\ln 2)\log|F|$ and size $|F'| \leq |F|^{1+1/(1+\log(k-1))}$. Choosing the parameters correctly, we get final sequences of length $O\big(n|F|^{1+O(1/\log\log\sigma)}\big)$.

3 On the Limitations of Fine-Grained Reductions

With the increasingly complex web of reductions and conjectures used in the "Hardness in P" research, one might oppose to our use of nonstandard assumptions. Why can't we base the hardness of shaving logs on one of the more established assumptions such as SETH, or even better, on $\mathsf{P} \neq \mathsf{NP}$? We conclude the paper with a proof that such results are not possible if one is restricted to fine-grained reductions, which is essentially the only tool we have in this line of research.

Let A be a problem with best known upper bound of $T_A(n)$ on inputs of size n, and let B be a problem with best known upper bound of $T_B(n)$ on inputs of size n. Throughout this section we assume that these runtime are non-decreasing functions, such as 2^n or n^2. A fine-grained reduction from "solving A in time $T_A(n)/g(n)$" to "solving B in time $T_B(n)/f(n)$" proves that improving $T_B(n)$ to $T_B(n)/f(n)$ improves T_A to $T_A(n)/g(n)$. Formally, it is an algorithm X that solves A and it is allowed to call an oracle for problem B, as long as the following bound holds. Let n_i be the size of the instance in the i^{th} call to problem B that our algorithm performs, where $i \leq t$ for some value t, and let $T_X(n)$ be the runtime of X excluding the time it takes to answer all the instances of problem B. It must be that $T_X(n) + \sum_{i=1}^{t} T_B(n_i)/f(n_i) \leq T_A(n)/g(n)$. This is a natural adaptation of the definition of fine-grained reductions from previous works, where the improvements were restricted to be by polynomial factors. We can now give a formal version of Theorem 1.1 from the introduction.

▶ **Theorem 3.1.** *If for some $c, \varepsilon > 0$ and all $k \geq 2$ there is a fine-grained reduction from solving k-SAT in time $\mathrm{poly}(n,m)2^n/2^{\varepsilon n}$ to solving LCS in time $O(n^2/\log^c n)$, then SETH is false.*

Proof. Assume there was a fine-grained reduction from k-SAT to LCS as above. This means that there is an algorithm X for k-SAT that makes t calls to LCS with instances of size $n_1, \ldots, n_t$ such that $T_X(n) + \sum_{i=1}^{t} n_i^2/\log^c n_i = O(\mathrm{poly}(n,m)2^n/2^{\varepsilon n})$. But then consider

algorithm X' which simulates X and whenever X makes a call to the LCS oracle with an instance of size n_i, our algorithm will execute the known quadratic time solution for LCS. Let n_{max} be the size of the largest instance we call, and note that $n_{max} < 2^n$. Simple calculations show that X' solves k-SAT and has a running time of $T_X(n) + \sum_{i=1}^{t} n_i^2 = O\big(\text{poly}(n,m)2^n/2^{\varepsilon n}\big) \cdot \log^c n_{max} = O(\text{poly}(n,m)2^n/2^{\varepsilon n})$ for all k, refuting SETH. ◀

References

1. Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir. Subtree isomorphism revisited. In *Proc. of 27th SODA*, pages 1256–1271, 2016.
2. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight Hardness Results for LCS and other Sequence Similarity Measures. In *Proc. of 56th FOCS*, pages 59–78, 2015.
3. Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proc. of the 48th STOC*, pages 375–388, 2016.
4. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. of 55th FOCS*, pages 434–443, 2014.
5. Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proc. of 27th SODA*, pages 377–391, 2016.
6. Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster sequence alignment. In *Proc. of 41st ICALP*, pages 39–51, 2014.
7. Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Proc. of 47th STOC*, pages 41–50, 2015.
8. Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proc. of 26th SODA*, pages 218–230, 2015.
9. P. Agarwal, R. B. Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. In *Proc. 24th ACM-SIAM Symposium on Discrete Algorithms (SODA'13)*, pages 156–167, 2013.
10. Lloyd Allison and Trevor I Dix. A bit-string longest-common-subsequence algorithm. *Information Processing Letters*, 23(5):305–310, 1986.
11. Helmut Alt and Maike Buchin. Can we compute the similarity between surfaces? *Discrete & Computational Geometry*, 43(1):78–99, 2010.
12. Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(1-2):78–99, 1995.
13. Helmut Alt, Christian Knauer, and Carola Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2004.
14. Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.
15. Alexander E Andreev. About one method of obtaining more than quadratic effective lower bounds of complexity of pi-schemes, 1987.
16. V Arlazarov, E Dinic, I Faradzev, and M Kronrod. On economic construction of the transitive closure of a direct graph. In *Sov. Math (Doklady)*, volume 11, pages 1209–1210, 1970.

17 Boris Aronov, Sariel Har-Peled, Christian Knauer, Yusu Wang, and Carola Wenk. Fréchet distance for curves, revisited. In *Proc. 14th Annual European Symposium on Algorithms (ESA'06)*, volume 4168 of *LNCS*, pages 52–63. Springer, 2006.

18 Arturs Backurs and Piotr Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false). In *Proc. of 47th STOC*, pages 51–58, 2015.

19 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *FOCS*, 2016.

20 Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. In *Proc. of 50th FOCS*, pages 745–754, 2009.

21 Ilya Baran, Erik D. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3sum. *Algorithmica*, 50(4):584–596, 2008. doi:10.1007/s00453-007-9036-3.

22 Paul Beame, Russell Impagliazzo, and Srikanth Srinivasan. Approximating acˆ 0 by small height decision trees and a deterministic algorithm for# acˆ 0sat. In *Computational Complexity (CCC), 2012 IEEE 27th Annual Conference on*, pages 117–125. IEEE, 2012.

23 Philip Bille and Martin Farach-Colton. Fast and compact regular expression matching. *Theoretical Computer Science*, 409(3):486–496, 2008. doi:10.1016/j.tcs.2008.08.042.

24 Philip Bille and Mikkel Thorup. Faster regular expression matching. In *International Colloquium on Automata, Languages, and Programming*, pages 171–182. Springer, 2009.

25 Maria Luisa Bonet and Samuel R. Buss. Size-depth tradeoffs for boolean fomulae. *Inf. Process. Lett.*, 49(3):151–155, 1994. doi:10.1016/0020-0190(94)90093-0.

26 Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *Proc. 31st International Conference on Very Large Data Bases (VLDB'05)*, pages 853–864, 2005.

27 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless seth fails. In *Proc. of 55th FOCS*, pages 661–670, 2014.

28 Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. *CoRR*, abs/1611.00918, 2016. URL: http://arxiv.org/abs/1611.00918, arXiv:1611.00918.

29 Karl Bringmann and Marvin Künnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *Proc. of 56th FOCS*, pages 79–97, 2015.

30 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo. Detecting commuting patterns by clustering subtrajectories. *International Journal of Computational Geometry & Applications*, 21(3):253–282, 2011.

31 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four soviets walk the dog - with an application to Alt's conjecture. In *Proc. 25th ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*, pages 1399–1413, 2014.

32 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*, pages 645–654, 2009.

33 Massimo Cairo, Roberto Grossi, and Romeo Rizzi. New bounds for approximating extremal distances in undirected graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 363–376, 2016.

34 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *Proc. of 21st CCC*, pages 252–260, 2006.

35 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Proc. of 4th IWPEC*, pages 75–85, 2009.

36 Erin Wolf Chambers, Éric Colin de Verdière, Jeff Erickson, Sylvain Lazard, Francis Lazarus, and Shripad Thite. Homotopic Fréchet distance between curves or, walking your dog in the woods in polynomial time. *Computational Geometry*, 43(3):295–311, 2010.

37 Timothy M. Chan. The art of shaving logs. In *Proc. of the 13th WADS*, page 231, 2013.

38 Timothy M. Chan. Speeding up the four russians algorithm by about one more logarithmic factor. In *Proc. of 26th SODA*, pages 212–217, 2015.

39 Krishnendu Chatterjee, Wolfgang Dvorák, Monika Henzinger, and Veronika Loitzenbauer. Model and objective separation with conditional lower bounds: Disjunction is harder than conjunction. *CoRR*, abs/1602.02670, 2016. URL: `http://arxiv.org/abs/1602.02670`, `arXiv:1602.02670`.

40 Ruiwen Chen. Satisfiability algorithms and lower bounds for boolean formulas over finite bases. In *International Symposium on Mathematical Foundations of Computer Science*, pages 223–234. Springer, 2015.

41 Ruiwen Chen and Valentine Kabanets. Correlation bounds and #sat algorithms for small linear-size circuits. In *Computing and Combinatorics - 21st International Conference, COCOON 2015, Beijing, China, August 4-6, 2015, Proceedings*, pages 211–222, 2015.

42 Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. *computational complexity*, 24(2):333–392, 2015.

43 Ruiwen Chen, Valentine Kabanets, and Nitin Saurabh. An improved deterministic# sat algorithm for small de morgan formulas. In *International Symposium on Mathematical Foundations of Computer Science*, pages 165–176. Springer, 2014.

44 Ruiwen Chen and Rahul Santhanam. Satisfiability on mixed instances. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 393–402, 2016.

45 A. F. Cook and Carola Wenk. Geodesic Fréchet distance inside a simple polygon. *ACM Transactions on Algorithms*, 7(1):193–204, 2010.

46 Thomas H.. Cormen, Charles Eric Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*, volume 6. MIT press Cambridge, 2001.

47 Maxime Crochemore, Costas S Iliopoulos, Yoan J Pinzon, and James F Reid. A fast and practical bit-vector algorithm for the longest common subsequence problem. *Information Processing Letters*, 80(6):279–285, 2001.

48 Maxime Crochemore, Gad M Landau, and Michal Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM journal on computing*, 32(6):1654–1673, 2003.

49 Evgeny Dantsin and Edward A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*, pages 403–424. IOS Press, 2009. `doi:10.3233/978-1-58603-929-5-403`.

50 Anne Driemel and Sariel Har-Peled. Jaywalking your dog: computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013.

51 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, 2012.

52 Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1994.

53 Ari Freund. Improved subquadratic 3sum. *Algorithmica*, pages 1–19, 2015.

54 Paweł Gawrychowski. Faster algorithm for computing the edit distance between slp-compressed strings. In *International Symposium on String Processing and Information Retrieval*, pages 229–236. Springer, 2012.

55 Michael Godau. A natural metric for curves - computing the distance for polygonal chains and approximation algorithms. In *Proc. 8th Symposium on Theoretical Aspects of Computer Science (STACS'91)*, volume 480 of *LNCS*, pages 127–136. Springer, 1991.

56 Omer Gold and Micha Sharir. Improved bounds for 3sum, k-sum, and linear degeneracy. *CoRR*, abs/1512.05279, 2015. URL: `http://arxiv.org/abs/1512.05279`, `arXiv:1512.05279`.

57 Alexander Golovnev, Alexander S Kulikov, Alexander Smal, and Suguru Tamaki. Circuit size lower bounds and# sat upper bounds through a general framework. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 23, page 22, 2016.

58 Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. In *Stringology*, pages 202–211, 2014. URL: `http://www.stringology.org/event/2014/p19.html`.

59 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 621–630, 2014.

60 Johan Håstad. The shrinkage exponent of de morgan formulas is 2. *SIAM J. Comput.*, 27(1):48–64, 1998.

61 John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *Journal of the ACM (JACM)*, 24(2):332–337, 1977.

62 Heikki Hyyrö. Bit-parallel lcs-length computation revisited. In *Proc. 15th Australasian Workshop on Combinatorial Algorithms (AWOCA 2004)*, pages 16–27. Citeseer, 2004.

63 Russell Impagliazzo, Shachar Lovett, Ramamohan Paturi, and Stefan Schneider. 0-1 integer linear programming with a linear number of constraints. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:24, 2014. URL: `http://eccc.hpi-web.de/report/2014/024`.

64 Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for ac 0. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 961–972. SIAM, 2012.

65 Russell Impagliazzo and Noam Nisan. The effect of random restrictions on formula size. *Random Structures & Algorithms*, 4(2):121–133, 1993.

66 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

67 Russell Impagliazzo, Ramamohan Paturi, and Stefan Schneider. A satisfiability algorithm for sparse depth two threshold circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 479–488, 2013.

68 Piotr Indyk. Approximate nearest neighbor algorithms for Fréchet distance via product metrics. In *Proc. 18th Annual Symposium on Computational Geometry (SoCG'02)*, pages 102–106, 2002.

69 Valeriy M Khrapchenko. Method of determining lower bounds for the complexity of p-schemes. *Mathematical Notes*, 10(1):474–479, 1971.

70 Ilan Komargodski, Ran Raz, and Avishay Tal. Improved average-case lower bounds for demorgan formula size. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 588–597. IEEE, 2013.

71 Isaac TS Li, Warren Shum, and Kevin Truong. 160-fold acceleration of the smith-waterman algorithm using a field programmable gate array (fpga). *BMC bioinformatics*, 8(1):1, 2007.

72 Yongchao Liu, Adrianto Wirawan, and Bertil Schmidt. Cudasw++ 3.0: accelerating smith-waterman protein database search by coupling cpu and gpu simd instructions. *BMC bioinformatics*, 14(1):1, 2013.

73 Anil Maheshwari, Jörg-Rüdiger Sack, Kaveh Shahbaz, and Hamid Zarrabi-Zadeh. Fréchet distance with speed limits. *Computational Geometry*, 44(2):110–120, 2011.

74 William J Masek and Michael S Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System sciences*, 20(1):18–31, 1980.

75 Daniel Moeller, Ramamohan Paturi, and Stefan Schneider. Subquadratic algorithms for succinct stable matching. In *Computer Science - Theory and Applications - 11th International Computer Science Symposium in Russia, CSR 2016, St. Petersburg, Russia, June 9-13, 2016, Proceedings*, pages 294–308, 2016.

76 Mario E. Munich and Pietro Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *Proc. 7th IEEE International Conference on Computer Vision*, volume 1, pages 108–115, 1999.

77 Gene Myers. A four russians algorithm for regular expression pattern matching. *Journal of the ACM (JACM)*, 39(2):432–448, 1992.

78 Igor C Oliveira. Algorithms versus circuit lower bounds. *arXiv preprint arXiv:1309.0249*, 2013.

79 Michael S Paterson and Uri Zwick. Shrinkage of de morgan formulae under restriction. *Random Structures & Algorithms*, 4(2):135–150, 1993.

80 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In *Proc. of 21st SODA*, pages 1065–1075, 2010.

81 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-sat. *J. ACM*, 52(3):337–364, 2005.

82 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proc. of 45th STOC*, pages 515–524, 2013.

83 Takayuki Sakai, Kazuhisa Seto, Suguru Tamaki, and Junichi Teruyama. A satisfiability algorithm for depth-2 circuits with a symmetric gate at the top and and gates at the bottom. In *Electronic Colloquium on Computational Complexity (ECCC)*, 2015.

84 Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *Proc. of the 51th FOCS*, pages 183–192, 2010.

85 Rahul Santhanam et al. Ironic complicity: Satisfiability algorithms and circuit lower bounds. *Bulletin of EATCS*, 1(106), 2013.

86 Kazuhisa Seto and Suguru Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. *computational complexity*, 22(2):245–274, 2013.

87 Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

88 Philip M Spira. On time-hardware complexity tradeoffs for boolean functions. In *Proceedings of the 4th Hawaii Symposium on System Sciences*, pages 525–527, 1971.

89 Bella Abramovna Subbotovskaya. Realizations of linear functions by formulas using+. *Doklady Akademii Nauk SSSR*, 136(3):553–555, 1961.

90 Avishay Tal. Shrinkage of de morgan formulae by spectral techniques. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 551–560. IEEE, 2014.

91 Avishay Tal. #sat algorithms from shrinkage. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:114, 2015. URL: `http://eccc.hpi-web.de/report/2015/114`.

92 Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

93 Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

94 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

95 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal on Computing*, 42(3):1218–1244, 2013.

96 Ryan Williams. Algorithms for Circuits and Circuits for Algorithms: Connecting the Tractable and Intractable. In *Proc. International Congress of Mathematicians*, 2014.

97 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proc. of 46th STOC*, pages 664–673, 2014.

98 Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014.

99 Huacheng Yu. An improved combinatorial algorithm for boolean matrix multiplication. In *Proc. of 42nd ICALP*, pages 1094–1105, 2015.

New Approximation Algorithms for (1,2)-TSP

Anna Adamaszek
University of Copenhagen, Denmark
anad@di.ku.dk

Matthias Mnich[1]
Universität Bonn, Germany
and Maastricht University, The Netherlands
mmnich@uni-bonn.de
https://orcid.org/0000-0002-4721-5354

Katarzyna Paluch[2]
Wroclaw University, Poland
abraka@cs.uni.wroc.pl
https://orcid.org/0000-0002-7504-6340

Abstract

We give faster and simpler approximation algorithms for the $(1,2)$-TSP problem, a well-studied variant of the traveling salesperson problem where all distances between cities are either 1 or 2.

Our main results are two approximation algorithms for $(1,2)$-TSP, one with approximation factor $8/7$ and run time $O(n^3)$ and the other having an approximation guarantee of $7/6$ and run time $O(n^{2.5})$. The $8/7$-approximation matches the best known approximation factor for $(1,2)$-TSP, due to Berman and Karpinski (SODA 2006), but considerably improves the previous best run time of $O(n^9)$. Thus, ours is the first improvement for the $(1,2)$-TSP problem in more than 10 years. The algorithm is based on combining three copies of a minimum-cost cycle cover of the input graph together with a relaxed version of a minimum weight matching, which allows using "half-edges". The resulting multigraph is then edge-colored with four colors so that each color class yields a collection of vertex-disjoint paths. The paths from one color class can then be extended to an $8/7$-approximate traveling salesperson tour. Our algorithm, and in particular its analysis, is simpler than the previously best $8/7$-approximation.

The $7/6$-approximation algorithm is similar and even simpler, and has the advantage of not using Hartvigsen's complicated algorithm for computing a minimum-cost triangle-free cycle cover.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Approximation algorithms, traveling salesperson problem, cycle cover

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.9

1 Introduction

The metric traveling salesperson problem (TSP) is one of the most fundamental combinatorial optimization problems. Given a complete undirected graph G with a metric cost function c on the edges of G, the goal is to find a tour $\mathcal{T}$ (i.e., a Hamiltonian cycle) of minimum cost in G, where the cost of $\mathcal{T}$ is the sum of costs of the edges traversed by $\mathcal{T}$. Four decades ago, Christofides [8] devised a polynomial-time algorithm that always outputs a tour with cost at most $3/2$ times the cost of an optimal tour. Improving this factor remains a major open problem in the area of approximation algorithms.

[1] Supported by ERC Starting Grant 306465 (BeyondWorstCase) and DFG grant MN 59/4-1.
[2] Supported by Polish National Science Center grant UMO-2013/11/B/ST6/01748.

The metric TSP is well-known to be NP-hard; it is one of Karp's 21 NP-complete problems [16]. In fact, Karp showed that the special case of metric TSP in which all distances between the cities are either 1 or 2, i.e., the cost function is of the form $c : E(G) \to \{1, 2\}$, is NP-hard. This special case is generally known as the $(1, 2)$-TSP problem. Notice that any instance of $(1, 2)$-TSP satisfies the triangle inequality. The $(1, 2)$-TSP problem has been considered in numerous papers [1, 2, 4, 10, 12, 16, 18, 19, 21, 24].

After Karp established the NP-hardness of $(1, 2)$-TSP, Papadimitriou and Yannakakis showed the problem to be APX-hard [24]. The currently best known inapproximability bound for $(1, 2)$-TSP is 535/534 [18]. A certain restriction of $(1, 2)$-TSP was considered by Fernandez de la Vega and Karpinski [10]. It was this restriction of $(1, 2)$-TSP that Trevisan [27] reduced from to establish the inapproximability of TSP in $\mathbb{R}^{\log n}$ under any ℓ_p metric. This hardness complemented Arora's breakthrough result [3] that TSP in $\mathbb{R}^2$ admits a PTAS under any ℓ_p metric.

One can also view the $(1, 2)$-TSP problem as the problem of finding a traveling salesperson tour that uses the maximum number of 1-edges in the given instance; here and throughout the paper, we will refer to edges of cost i as *i-edges*, for $i \in \{1, 2\}$. Alternatively, $(1, 2)$-TSP may be seen as a generalization of the Hamiltonian Cycle problem with non-edges represented by 2-edges.

Both $(1, 2)$-TSP and $(1, 2)$-ATSP (i.e., when the underlying graph G is a complete directed graph) are well-studied from the approximation point of view. For $(1, 2)$-TSP, it is NP-hard to obtain a performance guarantee better than 535/534 [18]. Papadimitriou and Yannakakis [24] gave a 7/6-approximation algorithm for $(1, 2)$-TSP; their algorithm works by successively merging cycles of a triangle-free cycle cover of the graph which they obtained by running Hartvigsen's algorithm [14]. The approximation factor was improved by Bläser and Ram [5] to 65/56, and to 8/7 by Berman and Karpinski [4]. Berman and Karpinski [4] used a local search approach: starting from a path cover they employ local improvements according to certain criteria, and finally connect the paths arbitrarily to a tour. Their algorithm takes time $O(n^9)$ for n-city instances.

1.1 Our Results

Our main results are novel approximation algorithms for $(1, 2)$-TSP that obtain the approximation ratios of 8/7 and 7/6, respectively.. The 8/7-approximation matches the best approximation factor known for $(1, 2)$-TSP, obtained by Berman and Karpinski [4], while improving the run time from $O(n^9)$ to $O(n^3)$. This is the first improvement for this classical problem in over 10 years.

▶ **Theorem 1.** *The (1,2)-TSP problem admits an 8/7-approximation in time $O(n^3)$, and a 7/6-approximation in time $O(n^{2.5})$.*

In this extended abstract we focus on presenting the 7/6-approximation algorithm, which is the simpler of our two algorithms. It is worth noting that it does not rely on Hartvigsen's involved algorithm [14] for computing a minimum-cost triangle-free cycle cover; in contrast, the 7/6-approximation by Papadimitriou and Yannakakis [24] relies on Hartvigsen's algorithm. (Papadimitriou and Yannakakis also gave an 11/9-approximation algorithm that does not use Hartvigsen's algorithm.) We defer the full details of our 8/7-approximation algorithm to the full version of this paper.

Outline of the approach. The idea of the 7/6-approximation algorithm is as follows. We start with computing a minimum cost cycle cover $\mathcal{C}_{\min}$ of the input graph G. Recall that a *cycle cover* of a graph G is a collection of simple cycles of G such that each vertex belongs to

exactly one cycle. Notice that the cost of $\mathcal{C}_{\min}$ is a lower bound on $\mathsf{opt}(G,c)$, where $\mathsf{opt}(G,c)$ denotes the cost of an optimal traveling salesperson tour in the graph G with cost function c. The cost of a minimum cost perfect matching $M_{\min}$ of G is also a lower bound, but this time on $\mathsf{opt}(G,c)/2$. This leads to our key idea of constructing a multigraph $\hat{G}$ on $V(G)$ from two copies of $\mathcal{C}_{\min}$ and one copy of $M_{\min}$. It readily follows that the cost of $\hat{G}$ satisfies $c(\hat{G}) \leq \frac{5}{2}\mathsf{opt}(G,c)$.

Next, we would like to color each edge of $\hat{G}$ with one of three colors so that each color class consists of vertex-disjoint paths, i.e., we would like to "path-3-color" $\hat{G}$. Given a path-3-coloring of $\hat{G}$, the paths of the color class that contains the maximum number of 1-edges can be patched in an arbitrary manner to form a traveling salesperson tour of weight not exceeding $\frac{7}{6}\mathsf{opt}(G,c)$. The exact calculation is given in Sect. 3.1.

However, we observe that not every multigraph $\hat{G}$ obtained from $\mathcal{C}_{\min}$ and $M_{\min}$ in the above way is path-3-colorable. For example, a subgraph of $\hat{G}$ obtained from a 4-cycle (called a *square*) $C \in \mathcal{C}_{\min}$ such that two edges of $M_{\min}$ connect vertices of C cannot be path-3-colored. The reason is that $\hat{G}$ has two copies of each edge of C and additionally two more edges coming from $M_{\min}$, and clearly it is not possible to color these ten edges with three colors without creating a monochromatic cycle.

Similarly, a subgraph of $\hat{G}$ obtained from a 3-cycle (called a *triangle*) $C \in \mathcal{C}_{\min}$ such that one of the edges of $M_{\min}$ connects vertices of C cannot be path-3-colored. An edge of $M_{\min}$ connecting two vertices of a cycle $C \in \mathcal{C}_{\min}$ is going to be called an *internal edge* of C).

While triangles of the above sort can be handled, by flipping edges, squares with two internal edges of $M_{\min}$ are problematic. Moreover, there are problem instances where every perfect matching of weight at most $\mathsf{opt}/2$ uses two internal edges of some square of $\mathcal{C}_{\min}$.

To get around this obstacle, we relax the notion of a matching and allow it to contain "half-edges". A *half-edge* of an edge e is, informally speaking, half of the edge e that contains exactly one of its endpoints. The notion of half-edges has been introduced by Paluch et al. [23]. We call such a relaxed matching $M^{\frac{1}{2}}$ with half-edges *perfect* if every vertex of the graph has exactly one edge or half-edge of $M^{\frac{1}{2}}$ incident to it. Now, we would like to compute a minimum-cost perfect matching $M^{\frac{1}{2}}_{\min}$ with half-edges, such that the half-edges can appear in a controlled way. In particular, for each 4-cycle of $\mathcal{C}_{\min}$ the matching uses correspondingly at most three "internal" half-edges; here, a half-edge of edge e is *internal* for a cycle C if is derived from an edge of G whose both endpoints belong to C. In such a matching the problem described above cannot occur. In Sect. 3.2 we show that $M^{\frac{1}{2}}_{\min}$ can be computed in time $O(n^{2.5})$, and that its weight is at most $\mathsf{opt}(G,c)/2$.

Next, from two copies of $\mathcal{C}_{\min}$ and one copy of $M^{\frac{1}{2}}_{\min}$ we will build a multigraph $\hat{G}$ whose cost is at most $\frac{5}{2}\mathsf{opt}(G,c)$ and that, after some modifications, is path-3-colorable, which yields the desired 7/6-approximation algorithm for (1,2)-TSP.

Modifying the multigraph $\hat{G}$. Before the multigraph $\hat{G}$ can be path-3-colored, it needs to be modified in certain ways. First, $\hat{G}$ should not contain any half-edges; so we replace all half-edges by an appropriate number of "whole" edges. Second, while coloring $\hat{G}$ we can restrict ourselves to coloring of edges of cost 1. Third, we remove some 1-edges if some optimal solution contains 2-edges; the exact relationship between the required number of 1-edges in $\hat{G}$ and the number of 2-edges in $\mathsf{opt}(G,c)$ is given in Sect. 3.1.

Fourth, before $\hat{G}$ can be path-3-colored, we need to "flip" certain edges and half-edges. For example, a subgraph of $\hat{G}$ obtained from a triangle $C \in \mathcal{C}_{\min}$ and one internal edge of C contained in $M^{\frac{1}{2}}_{\min}$ cannot be path-3-colored, and we need to flip this edge to the edge of $\hat{G}$ outside of C. The algorithm for path-3-coloring essentially comes from Dudycz et al. [9].

1.2 Related Work

Despite extensive research, the best approximation algorithm for metric TSP is still Christofides' algorithm [8] from 1976, which has a performance guarantee of 3/2. Generally the bound 3/2 is not believed to be tight. However, the currently largest known lower bound on the performance guarantee obtainable in polynomial time is as low as 123/122 [17]. A promising approach to improving upon the factor of 3/2 for metric TSP is to round a linear programming relaxation known as the Held-Karp relaxation [15], that is widely conjectured to have an integrality gap upper bounded by 4/3. However, even for the graphic TSP, the best known approximation upper bound of 7/5 due to Sebő and Vygen [26] does not match this conjectured upper bound of 4/3.

Another LP relaxation for TSP is the *subtour elimination LP*, which has constraints prescribing any vertex to be incident to exactly two edges of the TSP tour and constraints ruling out incomplete subtours (hence the name) by forcing edges to leave any non-empty proper subset of nodes. The best known integrality gap lower bound of the subtour elimination LP for $(1,2)$-TSP is 10/9, due to Williamson [29]. Qian et al. [25] showed an integrality gap upper bound of 19/15 for $(1,2)$-TSP (in a revised version, they improve the integrality gap upper bound to 5/4 and to 26/21 for fractionally Hamiltonian instances), and of 7/6 if the integrality gap is attained by a basic solution of the fractional 2-matching polytope. With the additional assumption that a certain type of modification maintains the 2-vertex connectedness of the support graph, they were able to show a tight integrality gap of 10/9. For fractionally Hamiltonian instances (i.e., where the optimal value of the LP relaxation of the subtour elimination formulation equals the order of the instance), Mnich and Mömke [21] prove integrality upper bounds of 5/4 in the general case and of 10/9 in the case of subcubic support graphs.

For $(1,2)$-ATSP, it is NP-hard to obtain a performance ratio better than 207/206 [18]. The first non-trivial approximation algorithm for $(1,2)$-ATSP was given by Vishwanathan [28], with an approximation factor of 17/12. This was improved to 4/3 by Bläser and Manthey [7]. The currently best approximation factor is 5/4, and is due to Bläser [6] and Paluch [22]. For fractionally Hamiltonian instances of $(1,2)$-ATSP, Mnich and Mömke [21] prove an integrality upper bound of 7/6.

The approach of using half-edges for solving variants of TSP was first used by Paluch et al. [23], who used it to give a 2/3-approximation for Max-ATSP. Later, Paluch [22] used half-edges to improve the approximation guarantee to 3/4 for the special case of Max-ATSP where all edge costs are either zero or one. Recently, Dudycz et al. [9] used half-edges to give a 4/5-approximation for Max-TSP.

2 Preliminaries

An instance of the $(1,2)$-TSP problem consists of pair (G,c), where G is a complete undirected graph and $c : E(G) \to \{1,2\}$ is an edge cost function, where each edge $e \in E(G)$ has a cost of $c(e) \in \{1,2\}$. A *tour* for the instance (G,c) is a subset $T \subseteq E(G)$ of edges of G that forms a Hamiltonian cycle of G, that is, the edges of T form a cycle that visits each vertex of G exactly once; the *cost* of T is defined as $c(T) = \sum_{e \in T} c(e)$. The goal is to find an *optimal* tour opt for (G,c), which is a tour of minimum cost. For a real number r, a tour T for an instance (G,c) is *r-approximate* if $c(T)/c(\mathsf{opt}) \leq r$.

For a graph G, a *cycle* is a sequence $C = (v_0, \ldots, v_{\ell-1})$ for some $\ell \geq 3$ of pairwise distinct vertices $v_i \in V$ such that $\{v_i, v_{i+1 \pmod \ell}\} \in E$ for $i \in \{0, \ldots, \ell-1\}$. We refer to ℓ as the *length* of C, and denote it by $\ell(C)$. For an integer ℓ, an *ℓ-cycle* is a cycle of length exactly ℓ,

and an $(\leq \ell)$-*cycle* is a cycle of length at most ℓ. For the sake of convenience, we also refer to 3-, 4-, 5- and 6-cycles as triangles, squares, pentagons and hexagons, respectively.

Let C be an ℓ-cycle of G. We say that C is *short* if $\ell \leq 6$. Further, an ℓ'-cycle C' of G with $\ell' < \ell$ is a *subcycle* of C if $V(C') \subset V(C)$. Note that C and C' can visit the vertices of $V(C')$ in different order. An edge $e = \{u, u'\}$ is a *native edge* of C if u, u' are two consecutive vertices of C, and a *diagonal* of C if u, u' are two non-consecutive vertices of C. When e is a native edge or a diagonal of C, we say that e is an *internal edge* of C. Finally, we call a cycle C a *1-cycle* if $c(e) = 1$ for all $e \in E(C)$; notice that there is no confusion of this notion with ℓ-cycles as we consider simple undirected graphs without loops.

Cycle covers. Our algorithm utilizes the concept of cycle covers. A *cycle cover* of G is a collection of cycles of G such that each vertex of G belongs to exactly one cycle of the collection. Thus, a Hamiltonian cycle of G is a cycle cover of G that consists of a single cycle. Cycle covers of undirected graphs are also known as 2-factors, because every vertex is incident to exactly two edges.

A cycle cover of G is *triangle-free* if each of its cycles has a length of at least 4. An essential ingredient of our 8/7-approximation algorithm is the following result by Hartvigsen [14]; the algorithm can be implemented to run in time $O(n^3)$ for an n-vertex graph [13].

▶ **Proposition 2** ([14]). *There is an algorithm that, given a complete graph G with edge costs $c : E(G) \to \{1, 2\}$, in strongly polynomial time computes a triangle-free cycle cover of G with minimum cost under c.*

***b*-matchings.** We will use the classical notion of *b-matchings* in graphs, which are a generalization of matchings. Let H be a graph. For a vector $b = (b_v)_{v \in V(H)} \in \mathbb{N}^{|V(H)|}$ where each coordinate corresponds to a vertex of H, a *b-matching* in H is a collection of edges $E(b) \subseteq E(H)$ that contains at most b_v edges incident to any vertex $v \in V(H)$. Notice that a b-matching with $b_v = 1$ for all $v \in V(H)$ is a classical matching in H.

A b-matching in H is said to be *maximum* if among all b-matchings in H it contains a maximum number of edges. Maximum matchings as well as maximum cost b-matchings can be computed in polynomial time. We refer to Lovász and Plummer [20] for further background on b-matchings.

We are interested in computing a b-matching in a graph H where each vertex $v \in V(H)$ has a lower bound ℓ_v and an upper bound b_v - we say that a vertex v has capacity interval $[\ell_v, b_v]$; the b-matching $E(\ell, b) \subseteq E(G)$ then contains at least ℓ_v edges and at most b_v edges incident to any vertex $v \in V(H)$. Such b-matchings can also be computed efficiently:

▶ **Proposition 3** ([11]). *There is an algorithm that, given a graph H and capacity intervals $[\ell_v, b_v]$, in time $O(\sqrt{\sum_{v \in V(H)} b_v} |E(H)|)$, computes a largest subgraph H' of H for which $\ell_v \leq d_{H'}(v) \leq b_v$ for every $v \in V(H')$.*

It is possible to reduce the problem of computing a b-matching with capacity intervals to the computation of a matching in which each vertex has capacity interval $[0, 1]$ or $[1, 1]$, i.e., a matching in which every vertex with capacity interval $[1, 1]$ is required to be matched; we defer the details to the full version of the paper.

Half-edges. Intuitively, half-edges correspond to halves of the edges of a graph and incident to only one vertex of the graph. Formally, from an instance (G, c) of $(1, 2)$-TSP we construct an *extended instance* (G', c') from (G, c), as follows. We start by setting $V(G') = V(G)$ and

$E(G') = E^1(G)$, where $E^1(G)$ denotes the subset of $E(G)$ containing all 1-edges. Next, for each edge $e = \{u, u'\} \in E^1(G)$ we add to $V(G')$ a new vertex v_e, and to $E(G')$ the edges $\{u, v_e\}, \{v_e, u'\}$. We refer to the vertices v_e as *extended vertices*, and to the remaining vertices of G' as *basic vertices*. We denote the new edges of E' as *half-edges* and the other edges as *basic edges*. Put concisely, G' is the *extended graph* of G with $V(G') = V(G) \cup \{v_e \mid e \in E^1(G)\}$, $E(G') = E^1(G) \cup \{\{u, v_e\}, \{v_e, u'\} \mid e = \{u, u'\} \in E^1(G)\}$.

A *matching with half-edges* $M^{\frac{1}{2}}$ in G' is a collection of edges in G', in which each vertex has degree 0 or 1 Intuitively, a matching with half-edges in G' corresponds to a relaxation of a matching in G, where we can take halves of the edges, incident to only one vertex, to the matching. We define the cost of a matching with half-edges $M^{\frac{1}{2}}$ in G' as $c'(M^{\frac{1}{2}}) = \frac{1}{2}|\{v \in V(G) : v \text{ is matched in } M^{\frac{1}{2}}\}| + |\{v \in V(G) : v \text{ is unmatched in } M^{\frac{1}{2}}\}|$. In other words, a basic vertex that is unmatched in $M^{\frac{1}{2}}$ contributes twice as much cost to $c'(M^{\frac{1}{2}})$ as a matched basic vertex. (We might say that we treat an unmatched basic vertex as if it was matched to a half-edge of a basic 2-edge in a perfect matching with half-edges in a graph G'' in which we also add basic 2-edges and their half-edges.) Therefore, any maximum matching with half-edges in G' has minimum cost.

3 A Fast and Simple 7/6-Approximation Algorithm for (1,2)-TSP

3.1 Outline of the Algorithm

We give an outline of our 7/6-approximation algorithm for $(1, 2)$-TSP, which is listed as Algorithm 1. For an instance (G, c) of the problem and a fixed tour $\mathcal{T}$ of G, let $\alpha_{\mathcal{T}}$ and $\beta_{\mathcal{T}}$ denote the number of 1-edges and 2-edges, respectively, in $\mathcal{T}$.

▶ **Observation 4.** *It holds that* $c(\mathcal{T}) = \alpha_{\mathcal{T}} + 2\beta_{\mathcal{T}} = \alpha_{\mathcal{T}} + 2(|V(G)| - \alpha_{\mathcal{T}}) = 2|V(G)| - \alpha_{\mathcal{T}}$.

Let G_1 denote the subgraph of G containing all 1-edges. In step 1 of the algorithm we compute a path-cycle cover $\mathcal{C}_{\min}$ of minimum cost in (G, c), using the algorithm from Proposition 3. A *path-cycle cover* of G is any b-matching of G_1 such that each vertex v has capacity interval $[0, 2]$. The cost of a path-cycle cover $\mathcal{C}$ of G is defined as $2n - |\mathcal{C}|$. Let $\mathcal{C}_{\min}$ denote a minimum cost path-cycle cover of G. Then, clearly, its cost is a lower bound on $c(\mathsf{opt})$.

In step 2 we use $\mathcal{C}_{\min}$ to construct a minimum cost matching with half-edges (and some additional properties) $M^{\frac{1}{2}}$; this construction is described in Sect. 3.2. In Sect. 3.3 we describe step 3, i.e., the construction of the graph G^1 from $\mathcal{C}_{\min}$ and $M^{\frac{1}{2}}$. In step 4 we path-3-color G^1, for which we use a modification of a path-3-coloring proposed by Dudycz et al. [9] for Max-TSP.

In summary, the algorithm works as follows:

Algorithm 1 Computing a 7/6-approximate solution for an instance (G, w) of (1,2)-TSP.

Input: An instance (G, c) of (1,2)-TSP.
Output: A tour $\mathcal{T}$ of G with cost $c(\mathcal{T}) \leq \frac{7}{6}c(\mathsf{opt})$.

1: Find a minimum-cost path-cycle cover $\mathcal{C}_{\min}$ of (G, c).
2: Find a minimum cost matching with half-edges (and some additional properties) $M^{\frac{1}{2}}$.
3: Based on $\mathcal{C}_{\min}$ and $M^{\frac{1}{2}}$, construct a multigraph G^1 on vertex set $V(G)$ with at least $\frac{5}{2}\alpha_{\mathsf{opt}} - \beta_{\mathsf{opt}}$ edges of cost 1 from G.
4: Path-3-color the edges of G^1.
5: Extend the set of edges of G^1 from the largest color class arbitrarily to a tour $\mathcal{T}$ of G.
6: **return** $\mathcal{T}$

▶ **Lemma 5.** *Algorithm 1 gives a 7/6-approximate solution for (1,2)-TSP.*

Proof. Let alg be a solution output by Algorithm 1 on input (G, c), and let α_{alg} and β_{alg} be the number of 1-edges resp. 2-edges in alg. Then

$$\alpha_{\mathsf{alg}} \geq \frac{\frac{5}{2}\alpha_{\mathsf{opt}} - \beta_{\mathsf{opt}}}{3} = \frac{5}{6}\alpha_{\mathsf{opt}} - \frac{1}{3}\beta_{\mathsf{opt}} \ .$$

By multiplying by 6 and using that $n = \alpha_{\mathsf{opt}} + \beta_{\mathsf{opt}}$, we obtain

$$6\alpha_{\mathsf{alg}} \geq 7\alpha_{\mathsf{opt}} - 2n = 7\alpha_{\mathsf{opt}} - 2(\alpha_{\mathsf{opt}} + \beta_{\mathsf{opt}}) = 5\alpha_{\mathsf{opt}} - 2\beta_{\mathsf{opt}} \ .$$

Subtracting $14n$ from both sides, and substituting $2(\alpha_{\mathsf{opt}} + \beta_{\mathsf{opt}})$ for $2n$ on the right hand side yields $12n - 6\alpha_{\mathsf{alg}} \leq 14n - 7\alpha_{\mathsf{opt}}$, which is equivalent to the desired result of

$$\frac{c(\mathsf{alg})}{c(\mathsf{opt})} = \frac{2n - \alpha_{\mathsf{alg}}}{2n - \alpha_{\mathsf{opt}}} \leq \frac{7}{6} \ . \qquad \blacktriangleleft$$

3.2 Computing a Minimum Cost Matching with Half-Edges

In this section we describe the construction of a minimum-cost matching with half-edges (and some additional properties) $M^{\frac{1}{2}}$ in the extended instance (G', c'), as defined in Sect. 2. Recall that a square is a 4-cycle. We refer to a diagonal of cost i as an *i-diagonal*, for $i = 1, 2$.

Intuitively, we want to ensure that $M^{\frac{1}{2}}$ matches at least one vertex of each 1-square of $\mathcal{C}_{\min}$ with some vertex from outside the square or leaves at least one vertex of a 1-square unmatched. As we want to find such a matching $M^{\frac{1}{2}}$ efficiently, and we want the cost of $M^{\frac{1}{2}}$ to be at most $c(\mathsf{opt})/2$, we allow $M^{\frac{1}{2}}$ to contain half-edges. However, we will only allow half-edges in a controlled manner. The idea is to allow a half-edge $\{u, v_e\}$ within $M^{\frac{1}{2}}$ only when the corresponding edge e is a native edge of a 1-square of $\mathcal{C}_{\min}$. Also, we want to allow at most one half-edge per each 1-square of $\mathcal{C}_{\min}$. For technical reasons (due to parity issues), we have to relax these simple conditions to a more complex set of conditions.

▶ **Definition 6.** A matching with half-edges $M^{\frac{1}{2}}$ in (G', c') is *good* for $\mathcal{C}_{\min}$ if it satisfies the following properties.

($M^{\frac{1}{2}}$.1) For each half-edge $\{u, v_e\} \in M^{\frac{1}{2}}$, except at most one *special half-edge*, the edge e is a native edge of a 1-square of $\mathcal{C}_{\min}$. Also, each 1-square of $\mathcal{C}_{\min}$ is incident to at most one half-edge of $M^{\frac{1}{2}}$.

($M^{\frac{1}{2}}$.2) For every 1-square $C \in \mathcal{C}_{\min}$ (i) there is a 1-edge $e_C \in M^{\frac{1}{2}}$ incident to C such that the other endpoint of e_C is incident to a cycle of $\mathcal{C}_{\min}$ different from C, or (ii) at least one of the vertices of C is unmatched in $M^{\frac{1}{2}}$.

($M^{\frac{1}{2}}$.3) $M^{\frac{1}{2}}$ may contain a special half-edge $\{u, v_e\}$ only if $|V(G)|$ is odd and $\mathcal{C}_{\min}$ does not contain a 2-edge. The following conditions are satisfied for the special half-edge $\{u, v_e\}$:

- **a.** e is an edge of a fixed odd-length 1-cycle $C_0 \in \mathcal{C}_{\min}$ called a *special cycle*.
- **b.** If $\mathcal{C}_{\min}$ contains a cycle of length at least 7, then C_0 has length at least 7.
- **c.** If C_0 is a triangle or pentagon and $\mathcal{C}_{\min}$ consists of at least two cycles, then at least two vertices of C_0 are incident to external edges of $M^{\frac{1}{2}}$, or some vertex of C_0 is unmatched in $M^{\frac{1}{2}}$.

▶ **Lemma 7.** *Consider an instance (G, c) of $(1, 2)$-TSP with minimum-cost path-cycle cover $\mathcal{C}_{\min}$ and extended instance (G', c'). A matching $M^{\frac{1}{2}}$ of minimum cost among all matchings with half-edges $M^{\frac{1}{2}}$ which are good for $\mathcal{C}_{\min}$ can be computed in time $O(n^{2.5})$ where $n = |V(G)|$.*

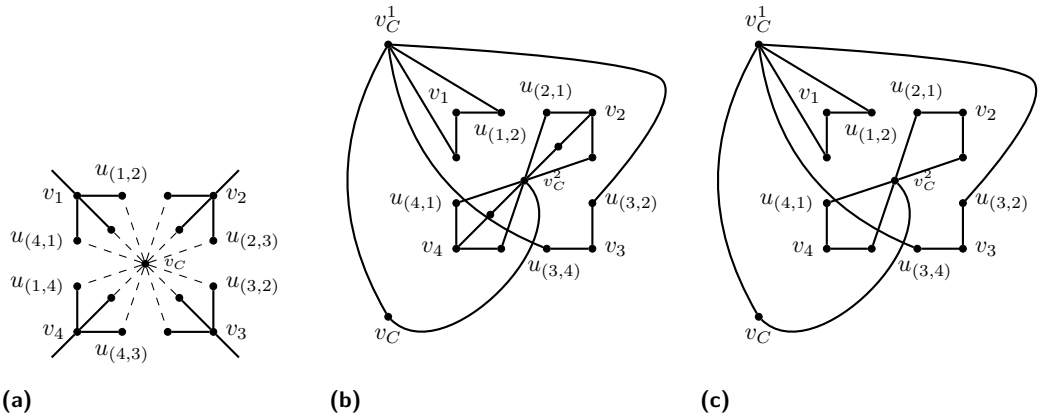


Figure 1 The gadget for modifying a 1-square $\{v_1, v_2, v_3, v_4\}$ of $\mathcal{C}_{\min}$ which has (a) two 1-diagonals, (b) one 1-diagonal $\{v_2, v_4\}$, and (c) no 1-diagonals.

Proof. First, from the instance (G, c) we create an unweighted graph G_0'' with a vertex capacity interval for each vertex of G_0''. We do that by locally and independently modifying each 1-square in $\mathcal{C}_{\min}$. The modification introduces new vertices and edges, as well as vertex capacity intervals $[\ell_v, u_v]$ for each vertex $v \in V(G_0'')$.

We start by setting $V(G_0'') = V(G)$, and assigning the capacity interval $[0, 1]$ to each vertex. For each edge $e \in E(G)$ which is not an internal 1-edge of a 1-square of $\mathcal{C}_{\min}$, we add e to $E(G_0'')$. Then, for each 1-square $C = (v_1, \ldots, v_4) \in \mathcal{C}_{\min}$ we proceed as follows. For each internal 1-edge $e_{\{i,j\}} = \{v_i, v_j\} \in E(G)$ of C (note that we consider both the native 1-edges and the 1-diagonals of C here), we introduce two new vertices $u_{(i,j)}, u_{(j,i)}$ with capacity intervals $[1, 1]$, and two new edges $\{v_i, u_{(i,j)}\}, \{u_{(j,i)}, v_j\}$. We call these added vertices *subdivision vertices*.

The exact type of further modification depends on whether the number of 1-diagonals of C in (G, c) is two, one, or zero.

- **If C has two 1-diagonals:** (See Fig. 1a) Introduce a vertex v_C of capacity interval $[9, 12]$; then connect v_C to all 12 subdivision vertices $u_{(i,j)}, u_{(j,i)}$.
- **If C has exactly one 1-diagonal $\{v_2, v_4\}$:** (See Fig. 1b) Introduce two vertices v_C^1, v_C^2 of capacity intervals $[3, 4]$ and $[5, 5]$, respectively, and one vertex v_C of capacity interval $[0, 1]$. Connect v_{C_1} to each vertex $u_{(i,j)}, u_{(j,i)}$ that is a neighbour of v_1 or v_3, and connect v_{C_2} to each vertex $u_{(i,j)}, u_{(j,i)}$ that is a neighbour of v_2 or v_4. Further, add two edges $\{v_C, v_C^1\}, \{v_C, v_C^2\}$.
- **If C is a square with no 1-diagonal:** (See Fig. 1c) Introduce two vertices v_C^1 and v_C^2 of capacity interval $[3, 3]$ and one vertex v_C of capacity interval $[0, 1]$. Connect v_{C_1} to each vertex $u_{(i,j)}, u_{(j,i)}$ that is a neighbour of v_1 or v_3, and connect v_{C_2} to each vertex $u_{(i,j)}, u_{(j,i)}$ that is a neighbour of v_2 or v_4. Further, add twoedges $\{v_C, v_C^1\}, \{v_C, v_C^2\}$.

This completes the construction of the graph G_0'' with vertex capacity intervals $[\ell_v, u_v]$ for each $v \in V(G_0'')$.

The cost of a b-matching M_0'' in G_0'' is defined as $c'(M_0'') = \frac{1}{2}|\{v \in V(G) : v \text{ is matched in } M_0''\}| + |\{v \in V(G) : v \text{ is unmatched in } M_0''\}|$. For the graph G_0'' and vertex capacity intervals $[\ell_v, u_v]$ for each $v \in V(G_0'')$, we compute a minimum-cost b-matching M_0'' that respects the vertex capacity intervals or, equivalently a b-matching that respects the vertex capacity intervals and minimizes the number of basic vertices unmatched in M_0''.

▶ **Claim 1.** *For the graph G_0'' and vertex capacity intervals $[\ell_v, u_v]$ for each $v \in V(G_0'')$, a minimum-cost b-matching M_0'' that respects the vertex capacity intervals can be computed in $O(n^{2.5})$ time.*

We defer the proof of Claim 1 to the full version of this paper.

▶ **Claim 2.** *The b-matching M_0'' in G_0'' can be transformed into a matching with half-edges $M^{\frac{1}{2}}$ which is good for $\mathcal{C}_{\min}$, has the same cost as M_0'', and contains no special half-edge.*

Proof of Claim 2. We construct the matching $M^{\frac{1}{2}}$ as follows. For any edge $e = \{u, v\} \in E(G_0'')$ such that both u, v are basic vertices of G_0'' (i.e., they correspond to the vertices of G, and not to the vertices introduced during the gadget construction), we add the edge e to $M^{\frac{1}{2}}$. Then, consider each 1-square $C = \{v_1, .., v_4\} \in \mathcal{C}_{\min}$ and a gadget corresponding to it. If there are some two vertices $v_i, v_j \in C$ such that both v_i and v_j are matched by the b-matching with subdivision vertices, and the edge $e = \{v_i, v_j\}$ in G' is a 1-edge, we add e to $M^{\frac{1}{2}}$. We construct such pairings greedily. For all vertices v_i which are matched by the b-matching with subdivision vertices, and which have not been paired, we add a half-edge $\{v_i, v_e\}$ to $M^{\frac{1}{2}}$, where $e = \{v_i, v_{(i \mod 4)+1}\}$.

The degree of each basic vertex $v \in V(G_0'')$ in $M^{\frac{1}{2}}$ is the same as the degree of v in the b-matching. The degree of each extended vertex $v \in V(G_0'')$ in $M^{\frac{1}{2}}$ is either 0 or 1. Therefore, $M^{\frac{1}{2}}$ is a matching with half-edges. Also, is it easy to see that the cost of $M^{\frac{1}{2}}$ is the same as the cost of the b-matching. We now have to prove that $M^{\frac{1}{2}}$ is good for $\mathcal{C}_{\min}$. As we did not denote any half-edge of $M^{\frac{1}{2}}$ as special, we only need to check properties 1 and 2 of Definition 6.

Consider a 1-square $C = \{v_1, \ldots, v_4\} \in \mathcal{C}_{\min}$. From the gadgets construction we can see that the vertex capacities for v_C, v_C^1, v_C^2 enforce that at most three of the vertices $\{v_1, \ldots, v_4\}$ are matched with a subdivision vertex. Therefore, at least one of the vertices $\{v_1, \ldots, v_4\}$ is matched by the b-matching via an edge not belonging to the gadget, i.e., an external 1-edge or at least one vertex of C is unmatched in M_0''. Therefore, Property 2 holds.

From the construction of $M^{\frac{1}{2}}$, each half-edge of $M^{\frac{1}{2}}$ corresponds to a native edge $e = \{v_i, v_{(i \mod 4)+1}\}$ of a 1-square. To prove Property 1, we need to show that each 1-square $C \in \mathcal{C}_{\min}$ is incident to at most one half-edge. We already know that at most three of the vertices $\{v_1, \ldots, v_4\}$ of C are matched with a subdivision vertex. If there are three, some two of them are incident to neighboring vertices of C, and will be transformed into one native edge in $M^{\frac{1}{2}}$, which will result in only one half-edge of $M^{\frac{1}{2}}$ incident to C. If exactly two of the vertices $\{v_1, \ldots, v_4\}$ of C were matched with a subdivision vertex, they yield two half-edges within $M^{\frac{1}{2}}$ only if they are incident with the opposite corners of C, and the corresponding diagonal has cost 2. We show that the construction of the gadgets prevents this from happening.

First, consider the case when C has no 1-diagonal, see Fig. 1c. Assume, without loss of generality, that exactly the vertices v_1, v_3 are matched by the b-matching with the subdivision vertices. Then, as the capacity interval of v_C^1 is $[3, 3]$, and the capacity interval of v_C is $[0, 1]$, v_C must be matched with v_C^1, and v_C^1 must be matched with 2 of the subdivision vertices. Then, as the capacity interval of v_C^2 is $[3, 3]$, v_C^2 must be matched with 3 subdivision vertices. But that leaves one subdivision vertex unmatched, and it therefore cannot happen.

Now, consider the case when C has exactly one 1-diagonal $\{v_2, v_4\}$, see Fig. 1b. Assume that exactly the vertices v_1, v_3 are matched by the b-matching with the subdivision vertices. Then, by the capacity intervals of v_C^1 and v_C, again v_C must be matched with v_C^1, and v_C^1 must be matched with 2 of the subdivision vertices. Then, as the capacity interval of v_C^2 is $[5, 5]$, v_C^2 must be matched with 5 subdivision vertices. But that leaves one subdivision vertex unmatched, and it therefore cannot happen.

Each 1-square of $\mathcal{C}_{\min}$ is incident to at most one half-edge, and therefore Property 1 holds and the matching $M^{\frac{1}{2}}$ is good. This completes the proof of Claim 2. ◀

▶ **Claim 3.** *Any matching with half-edges $M^{\frac{1}{2}}$ which is good for $\mathcal{C}_{\min}$ and contains no special half-edge can be transformed into a b-matching M_0'' in G_0'' of the same cost.*

Proof of Claim 3. Consider a matching with half-edges $M^{\frac{1}{2}}$ which is good for $\mathcal{C}_{\min}$ and contains no special half-edge. We will construct a corresponding b-matching for $M^{\frac{1}{2}}$. For any edge $e = \{u, v\} \in M^{\frac{1}{2}}$ which is not a half-edge or a 1-edge of a 1-square, e is also present in the graph G_0'', and we add e to the b-matching. Now, consider any half-edge $\{u, v_e\} \in M^{\frac{1}{2}}$, where $e = \{u, u'\}$. From Property 1 of Definition 6, e is a native edge of a square $C \in \mathcal{C}_{\min}$. In the b-matching, we connect u with any subdivision edge neighbouring with it. Last, for any edge $\{u_i, u_j\}$ which is a 1-edge of a 1-square, we take the two edges $\{u_i, u_{(i,j)}\}$ and $\{u_{(j,i)}, u_j\}$ into the b-matching. From Property 2 of Definition 6, at most 3 subdivision edges corresponding to any 1-square $C \in \mathcal{C}_{\min}$ have been matched by this procedure. Moreover, if there were two or three, then some two of them must be incident to two endpoints of a 1-edge of C (either a native edge, or a diagonal).

We now show how to extend this matching to a b-matching. For any 1-square $C \in \mathcal{C}_{\min}$ with two 1-diagonals, we match v_C with the at least 9 unmatched subdivision vertices.

This completes the proof of Claim 3. ◀

If $|V(G)|$ is odd and $\mathcal{C}_{\min}$ contains only 1-edges, we also build another unweighted graph G_1'' from G', in which we find a b-matching M_1''. The graph G_1'' is quite similar to G_0''. The details of constructing G_1'' and computing M_1'' are given in the full version.

From M_1'' we obtain a matching $M_1^{\frac{1}{2}}$ with half-edges good for $\mathcal{C}_{\min}$. If $|V(G)|$ is odd and $\mathcal{C}_{\min}$ does not contain any 2-edge, we set as $M^{\frac{1}{2}}$ that one of the matching $M^{\frac{1}{2}}$ and $M_1^{\frac{1}{2}}$ that has smaller cost. This completes the proof. ◀

▶ **Lemma 8.** *Any minimum-cost matching $M^{\frac{1}{2}}$ of (G', c') that is good for $\mathcal{C}_{\min}$ satisfies $c'(M^{\frac{1}{2}}) \leq c(\mathsf{opt})/2$.*

Proof. Let (G'', c'') denote the extension of the graph G, in which we add two half-edges of each edge of G, also those of cost 2. Each edge e of G has cost $c''(e) = c(e)$ in G''. Each half-edge of a 1-edge $e \in G$ has cost $\frac{1}{2}$ and each half-edge of a 2-edge $e \in G$ has cost 2. The cost of a matching M in G'' is defined in the usual way as $c''(M) = \sum_{e \in M} c''(e)$. We notice that for any matching M in G'' it holds $c''(M) = c'(M')$, where $M' = M \cap E(G')$.

To prove the lemma, we partition the edges of a fixed but arbitrary tour opt of minimum cost in (G, c) into two perfect matchings $M_1 \cup M_2$ in (G'', c'') with half-edges, each of which constitutes in (G', c') a matching with half-edges, which is good for $\mathcal{C}_{\min}$.

To this end, let $\mathcal{S}_2$ denote the set of squares in $\mathcal{C}_{\min}$ such that opt uses two of its internal 1-edges. Similarly, let $\mathcal{S}_3$ denote the set of squares in $\mathcal{C}_{\min}$ such that opt uses three of its internal 1-edges. Let us note that if $\mathcal{S}_2 \cup \mathcal{S}_3 \neq \emptyset$, then partitioning opt into two perfect matchings might yield a matching or matchings that are not good for $\mathcal{C}_{\min}$. Therefore, for each square $C \in \mathcal{S}_2 \cup \mathcal{S}_3$, we take one of its internal 1-edges e_C belonging to opt, and split e_s into two half-edges. For each such edge e_C, we place one of its half-edges into M_1 and place its other half-edge into M_2.

If the parities of $|\mathcal{S}_2 \cup \mathcal{S}_3|$ and $|V(G)|$ are the same, then this way we have already decomposed opt into two perfect matchings with half-edges M_1 and M_2. Assume, without loss of generality, that $c'(M_1) \leq c'(M_2)$. From M_1 we construct a matching M' in G'. We first initialize $M' = M_1$. This way, the condition $c'(M') \leq c'(\mathsf{opt})/2$ is clearly satisfied.

However, M' potentially is not a perfect matching with half-edges, as it might contain a half-edge of a diagonal of a square $C \in \mathcal{S}_2$. We can, however, replace such a half-edge with a half-edge of an edge of C, without increasing the cost of M'.

If the parities of $|\mathcal{S}_2 \cup \mathcal{S}_3|$ and $|V(G)|$ differ and opt uses a 2-edge, then we choose any such 2-edge $e \in \mathsf{opt}$ and split it into two half-edges. Otherwise, if $\mathcal{S}_2 \cup \mathcal{S}_3$ is empty, $|V(G)|$ is odd. Then any path-cycle cover of G must contain at least one odd cycle C. We split any edges of opt which is incident to a vertex of C into two half-edges. We decompose opt into two perfect matchings M_1 and M_2. Since $c'(M_1) = c'(M_2)$, we may choose that one which contains a half-edge incident to a vertex of C.

The remaining case is when the parities of $|\mathcal{S}_2 \cup \mathcal{S}_3|$ and $|V(G)|$ differ, each edge of opt has cost 1 and $\mathcal{S}_2 \cup \mathcal{S}_3$ is non-empty. Then we choose one square $C \in S_2 \cup S_3$ and do not split any of its edges. At least one of the perfect matchings M_1, M_2 from the decomposition of opt is such that it does not use two internal edges of C. Since again $c'(M_1) = c'(M_2)$, we may choose that one, which does not use two internal edges of C. ◀

3.3 Constructing the Multigraph

We will now construct a multigraph G^1 from the path-cycle cover $\mathcal{C}_{\min}$, and the minimum-cost matching $M^{\frac{1}{2}}$ that is good for $\mathcal{C}_{\min}$. We set $V(G^1) = V(G)$. The idea is to take into G^1 two copies of each 1-edge of $\mathcal{C}_{\min}$, and one copy of each 1-edge and of each 1/2-half-edge of $M^{\frac{1}{2}}$. However, to ensure that we will be able to color the graph at a later stage, and as we do not have the extended vertices v_e in $V(G^1)$ to accommodate half-edges, we first need to modify the matching $M^{\frac{1}{2}}$ into a collection of edges M. The set M does not have to be a matching—it may contain multiple edges incident to the same vertex, and even multiple copies of the same edge of G. Also, we need to ensure that the multigraph G^1 has at least $\frac{5}{2}\alpha_{\mathsf{opt}} - \beta_{\mathsf{opt}}$ edges of cost 1 in (G, c), i.e., that the set M has at least $\frac{1}{2}\alpha_{\mathsf{opt}} - \beta_{\mathsf{opt}}$ edges of cost 1 in (G, c).

We start by setting M to be the collection of edges and half-edges of $M^{\frac{1}{2}}$. Then we modify M by executing the following sequence of steps:

(O1) For every 1-triangle (u, v, w) of $\mathcal{C}_{\min}$ for which $\{u, v\} \in M$ and $\{w, t\} \in M$ for some t, we remove the edge $\{u, v\}$ from M, and instead we add a second copy of $\{w, t\}$ into M. Notice that if we perform a similar operation with the cycle of $\mathcal{C}_{\min}$ containing the vertex t, it will result in the third copy of $\{w, t\}$ being added to the graph.

(O2) For every 1-triangle (u, v, w) of $\mathcal{C}_{\min}$ for which $\{u, v\} \in M$ and w is unmatched in M, we remove the edge $\{u, v\}$ from M.

(O3) For every 1-square (u, v, w, z) of $\mathcal{C}_{\min}$ for which $\{u, v\}, \{w, v_e\} \in M$ and $\{z, t\} \in M$ for some t, we remove $\{w, v_e\}$ from M, and instead we add a second copy of $\{z, t\}$ into M. Notice that in this case we really need half of the additional edge $\{z, t\}$, so if we perform such operation twice for $\{z, t\}$ (i.e., the component containing t is also a 1-square), we have to add only one, and not two copies of $\{z, t\}$. We also perform the same operation if $\{u, v\}$ is a diagonal, and not a native edge of the square.

(O4) For every 1-square (u, v, w, z) of $\mathcal{C}_{\min}$ for which $\{u, v\}, \{w, v_e\} \in M$ and z is unmatched in M, we remove the half-edge $\{w, v_e\}$ from M. As before, we perform the same operation if $\{u, v\}$ is a diagonal, and not a native edge of the square.

(O5) For every 1-square (u, v, w, z) of $\mathcal{C}_{\min}$ for which $\{u, v_e\} \in M$ and no other edge or diagonal of the square (or its half) is in M, we remove $\{u, v_e\}$ from M and instead we add $\{u, v\}$ into M where $e = \{u, v\}$.

(O6) If the matching M contains a special half-edge $\{u, v_e\}$, then we add e to M.

After these operations, there are no half-edges left in M. We now construct the multigraph G^1 by setting $V(G^1) = V(G)$, and by adding to G^1 two copies of each 1-edge of $\mathcal{C}_{\min}$ and all edges of M. We can show a lower bound on the number of edges of G^1.

▶ **Lemma 9.** *The multigraph G^1 has at least $\frac{5}{2}\alpha_{\mathsf{opt}} - \beta_{\mathsf{opt}}$ edges which are 1-edges of (G, c).*

Proof. The minimum-cost path-cycle cover $\mathcal{C}_{\min}$ contains at least α_{opt} edges of cost 1 in (G, c). Therefore, two copies of $\mathcal{C}^1_{\min}$ contain at least $2\alpha_{\mathsf{opt}}$ edges. The matching $M^{\frac{1}{2}}$ has cost at most $c(opt)/2$. Therefore, the number of 1-edges in $M^{\frac{1}{2}}$ is at least $\alpha_{\mathsf{opt}}/2$, where half-edges count as half of an edge each. We further have that $\beta_{\mathsf{opt}} \geq \beta_M$, where β_M denotes the number of basic verices unmatched in $M^{\frac{1}{2}}$.

The only modifications that decrease the number of edges in G^1 are operations 2 and 4. However, for each triangle or square for which we remove one 1-edge or 1/2-half-edge, we can uniquely charge it to an unmatched basic vertex. Thus, the number of such deletions is at most β_M and can be charged against β_{opt}. Consequently, we always have at least $\frac{5}{2}\alpha_{\mathsf{opt}} - \beta_{\mathsf{opt}}$ edges in the resulting multigraph G^1. ◀

The multigraph G^1 can be essentially path-3-colored using the path-3-coloring procedure by Dudycz et al. [9]. The multigraph colored by Dudycz et al. [9] is built from two copies of a maximum-cost cycle cover and a maximum cost perfect matching. Several not very serious modifications are needed in order to deal with double and triple edges of M; note that the existence of such edges means that some vertices in G^1 have degree greater than 5. Again, details are deferred to the full version of the paper.

4 A New 8/7-Approximation Algorithm for (1,2)-TSP

The 8/7-approximation algorithm is quite similar to the algorithm with an approximation factor of 7/6. Instead of a minimum-cost path-cycle cover $\mathcal{C}_{\min}$ of (G, c) we use a minimum-cost *triangle-free* cycle cover $\mathcal{C}^t_{\min}$. We also compute a minimum-cost matching $M^{\frac{1}{2}}$ with half-edges with additional properties. To obtain an 8/7-approximation, $M^{\frac{1}{2}}$ has to additionally satisfy the condition that for every 1-hexagon C from $\mathcal{C}^t_{\min}$ at least one of the vertices of C must be incident to an external edge of $M^{\frac{1}{2}}$ or be unmatched in $M^{\frac{1}{2}}$. Next, we build a multigraph G^1 that consists of three copies of $\mathcal{C}^t_{\min}$ and one copy of $M^{\frac{1}{2}}$. We do some flipping of edges and half-edges and path-4-color the multigraph G^1. Path-4-coloring is based on the same ideas as path-3-coloring but is a little more complicated. We defer the details to the full version of this paper.

References

1 Eric Angel, Evripidis Bampis, and Laurent Gourvès. Approximating the Pareto curve with local search for the bicriteria TSP(1,2) problem. *Theoret. Comput. Sci.*, 310(1):135–146, 2004.

2 Eric Angel, Evripidis Bampis, Laurent Gourvès, and Jérôme Monnot. (Non)-approximability for the multi-criteria TSP(1,2). In *Proc. FCT 2005*, volume 3623 of *Lecture Notes Comput. Sci.*, pages 329–340, 2005.

3 Sanjeev Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proc. FOCS 1996*, pages 2–11. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548458.

4 Piotr Berman and Marek Karpinski. 8/7-approximation algorithm for (1, 2)-TSP. In *Proc. SODA 2006*, pages 641–648, 2006.

5 M. Bläser and L. Shankar Ram. An improved approximation algorithm for TSP with distances one and two. In *Proc. FCT 2005*, volume 3623 of *LNCS*, pages 504–515. Springer, 2005. doi:10.1007/11537311_44.

6 Markus Bläser. A 3/4-approximation algorithm for maximum ATSP with weights zero and one. In *Proc. APPROX-RANDOM 2004*, volume 3122 of *Lecture Notes Comput. Sci.*, pages 61–71. Springer, 2004. doi:10.1007/978-3-540-27821-4_6.

7 Markus Bläser and Bodo Manthey. Approximating maximum weight cycle covers in directed graphs with weights zero and one. *Algorithmica*, 42(2):121–139, 2005.

8 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem, 1976. TR 388, Graduate School of Industrial Administration, CMU.

9 Szymon Dudycz, Jan Marcinkowski, Katarzyna Paluch, and Bartosz Rybicki. *A 4/5-Approximation Algorithm for the Maximum Traveling Salesman Problem*, volume 10328 of *Lecture Notes Comput. Sci.*, pages 173–185. Springer, 2017. doi:10.1007/978-3-319-59250-3_15.

10 W. Fernandez de la Vega and M. Karpinski. On the approximation hardness of dense TSP and other path problems. *Inform. Process. Lett.*, 70(2):53–55, 1999.

11 Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proc. STOC 1983*, pages 448–456, 1983.

12 Mikael Gast, Matthias Hauptmann, and Marek Karpinski. Approximability of TSP on power law graphs. *CoRR*, abs/1509.03976, 2015. arXiv:1509.03976.

13 David B. Hartvigsen. Personal communication.

14 David B. Hartvigsen. *Extensions of Matching Theory*. PhD thesis, Carnegie-Mellon University, 1984.

15 Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Res.*, 18:1138–1162, 1970.

16 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 85–103. Plenum, New York, 1972.

17 Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for TSP. In *Proc. ISAAC 2013*, volume 8283 of *Lecture Notes Comput. Sci.*, pages 568–578. Springer, 2013. doi:10.1007/978-3-642-45030-3_53.

18 Marek Karpinski and Richard Schmied. On approximation lower bounds for TSP with bounded metrics. *Electr. Colloquium Computat. Complexity*, 19:8, 2012.

19 Marek Karpinski and Richard Schmied. Approximation hardness of graphic tsp on cubic graphs. *RAIRO Oper. Res.*, 49(4):651–668, 2015.

20 László Lovász and Michael D. Plummer. *Matching theory*. AMS Chelsea Publishing, Providence, RI, 2009. Corrected reprint of the 1986 original.

21 Matthias Mnich and Tobias Mömke. Improved integrality gap upper bounds for traveling salesperson problems with distances one and two. *European J. Oper. Res.*, 266(2):436–457, 2018.

22 Katarzyna Paluch. Maximum ATSP with weights zero and one via half-edges. *Theory Comput. Syst.*, 62(2):319–336, 2018.

23 Katarzyna Paluch, Khaled Elbassioni, and Anke van Zuylen. Simpler approximation of the maximum asymmetric traveling salesman problem. In *Proc. STACS 2012*, volume 14 of *Leibniz Int. Proc. Informatics*, pages 501–506, 2012.

24 Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18(1):1–11, 1993.

25 Jiawei Qian, Frans Schalekamp, David P. Williamson, and Anke van Zuylen. On the integrality gap of the subtour LP for the 1,2-TSP. In *Proc. LATIN 2012*, volume 7256 of *Lecture Notes Comput. Sci.*, pages 606–617, 2012.

26 András Sebő and Jens Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-TSP, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014.

27 Luca Trevisan. When Hamming meets Euclid: the approximability of geometric TSP and Steiner tree. *SIAM J. Comput.*, 30(2):475–485, 2000.

28 Sundar Vishwanathan. An approximation algorithm for the asymmetric travelling salesman problem with distances one and two. *Inform. Process. Lett.*, 44(6):297–302, 1992.

29 David P. Williamson. Analysis of the Held-Karp heuristic for the traveling salesman problem. Master's thesis, MIT, 1990.

Union of Hypercubes and 3D Minkowski Sums with Random Sizes

Pankaj K. Agarwal[1]
Department of Computer Science, Duke University, Durham, NC 27708, USA
pankaj@cs.duke.edu

Haim Kaplan[2]
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
haimk@tau.ac.il

Micha Sharir[3]
School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel
michas@tau.ac.il

Abstract

Let $T = \{\triangle_1, \ldots, \triangle_n\}$ be a set of of n pairwise-disjoint triangles in $\mathbb{R}^3$, and let B be a convex polytope in $\mathbb{R}^3$ with a constant number of faces. For each i, let $C_i = \triangle_i \oplus r_i B$ denote the Minkowski sum of $\triangle_i$ with a copy of B scaled by $r_i > 0$. We show that if the scaling factors $r_1, \ldots, r_n$ are chosen randomly then the expected complexity of the union of $C_1, \ldots, C_n$ is $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$; the constant of proportionality depends on ε and the complexity of B. The worst-case bound can be $\Theta(n^3)$.

We also consider a special case of this problem in which T is a set of points in $\mathbb{R}^3$ and B is a unit cube in $\mathbb{R}^3$, i.e., each C_i is a cube of side-length $2r_i$. We show that if the scaling factors are chosen randomly then the expected complexity of the union of the cubes is $O(n \log^2 n)$, and it improves to $O(n \log n)$ if the scaling factors are chosen randomly from a "well-behaved" probability density function (pdf). We also extend the latter results to higher dimensions. For any fixed odd value of d, we show that the expected complexity of the union of the hypercubes is $O(n^{\lfloor d/2 \rfloor} \log n)$ and the bound improves to $O(n^{\lfloor d/2 \rfloor})$ if the scaling factors are chosen from a "well-behaved" pdf. The worst-case bounds are $\Theta(n^2)$ in $\mathbb{R}^3$, and $\Theta(n^{\lceil d/2 \rceil})$ in higher dimensions.

2012 ACM Subject Classification Theory of computation → Computational geometry, Theory of computation → Generating random combinatorial structures

Keywords and phrases Computational geometry, Minkowski sums, Axis-parallel cubes, Union of geometric objects, Objects with random sizes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.10

[1] Work on this paper by Pankaj Agarwal and Micha Sharir has been supported by Grant 2012/229 from the U.S.-Israel Binational Science Fund. Work by Pankaj Agarwal has also been supported by NSF under grants CCF-09-40671, CCF-10-12254, and CCF-11-61359, by ARO grants W911NF-07-1-0376 and W911NF-08-1-0452, and by an ERDC contract W9132V-11-C-0003.
[2] Work by Haim Kaplan has been supported by grant 1841-14 from the Israel Science Fund, and grant 1367-2017 from the German-Israeli Science Fund.
[3] Work by Micha Sharir has also been supported by Grant 892-13 from the Israel Science Fund, by the Blavatnik Research Fund in Computer Science at Tel Aviv University, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

1 Introduction

We advance the state of the art regarding the complexity of the union of combinatorial objects in dimensions $d \geq 3$ beyond worst-case analysis. To that end we study the union complexity of Minkowski sums of pairwise disjoint triangles with a randomly scaled copies of a fixed convex polytope, and Minkowski sums of points with cubes of random sizes. We hope that our techniques will be useful in future studies of such stochastic arrangements.

Specifically, let $T = \{\triangle_1, \ldots, \triangle_n\}$ be a collection of n pairwise-disjoint triangles in $\mathbb{R}^3$, and let B be a fixed convex polytope in $\mathbb{R}^3$ with a constant number of faces. We consider the setup where we are given a sequence $\mathbf{r} = \langle r_1, \ldots, r_n \rangle$ of non-negative scaling factors (or *sizes*), and we let $C_i = \triangle_i \oplus r_i B$ denote the Minkowksi sum of $\triangle_i$ with a copy of B scaled by r_i, for $i = 1, \ldots, n$. Each C_i is a convex polytope with a constant number of faces. Let $\mathcal{C} = \{C_1, \ldots, C_n\}$, and let $\mathcal{U} = \mathcal{U}(T, \mathbf{r}) = \bigcup_{i=1}^{n} C_i$ denote their union. We also use $\mathcal{U}(\mathcal{C})$ to denote $\mathcal{U}(T, \mathbf{r})$.

The *combinatorial complexity* of $\mathcal{U}$ is the total number of faces of all dimensions on the boundary $\partial\mathcal{U}$ of $\mathcal{U}$. Each vertex of $\partial\mathcal{U}$ is a vertex of some set C_i, an intersection point between an edge of some C_i and a face of another C_j, or an intersection point of three faces of three distinct C_i's. By Euler's formula, the overall complexity of $\partial\mathcal{U}$ is proportional to the number of vetices on $\partial\mathcal{U}$. Therefore we measure the combinatorial complexity of $\mathcal{U}$ by the number of its vertices, and denote this quantity by $\psi(T, \mathbf{r})$.

Our goal is to obtain an upper bound on the *expected* value of $\psi(T, \mathbf{r})$, under a suitable stochastic model for choosing the scaling factors $\mathbf{r}$ for the members of $\mathcal{C}$. We refer to this problem as the *stochastic Minkowski-sum union* problem. Our expected bounds are significantly better than the worst case bounds (for any such union), indicating that inputs which are not adversarial are likely to have lower union complexity.

Another motivation for our analysis is for an efficient computation of the most vulnerable location of an attack in a three-dimensional scene. Concretely, we use the model where we have a scene consisting of objects, modeled as a collection of pairwise disjoint triangles. An attack occurs at some point, and the probability of a triangle to be hit decreases as its distance from the attacking point increases (up to some threshold). We want to compute a point of attack in which we maximize the expected number of triangles that we hit. The same technique to approximately solve this problem, as done in the planar case [1, 2], leads to questions about the complexity of the union of Minkowski sums of the input triangles with balls of random radii. The case studied here replaces the ball by a convex polytope, which we choose to approximate the Euclidean ball. The results obtained in this paper lead to improved solution to the vulnerability problem, by roughly one factor of n.
We consider two stochastic models for choosing the sequence $\mathbf{r} = \langle r_1, \ldots, r_n \rangle$ of scaling factors:

The density model. We are given an arbitrary probability density (or mass) function (pdf) π over the non-negative reals; for each $1 \leq i \leq n$, we take r_i to be a random value drawn independently from the distribution determined by π.

The permutation model. We are given a sequence $\Theta = \langle \theta_1, \ldots, \theta_n \rangle$ of n arbitrary non-negative real numbers, with $\theta_1 \leq \theta_2 \leq \cdots \leq \theta_n$. We draw a random permutation σ on $[1 : n]$, where all permutations are equally likely to be chosen, and assign $r_i := \theta_{\sigma(i)}$ to $\triangle_i$, for each $i = 1, \ldots, n$.

For the density model, let $\psi(T, \pi)$ denote the expected value of $\psi(T, \mathbf{r})$, where the expectation is taken over the random choices of $\mathbf{r}$, drawn from π in the manner specified above. Set $\psi(n) = \max \psi(T, \pi)$, where the maximum is taken over all probability density (mass) functions and over all sets T of n pairwise-disjoint triangles in $\mathbb{R}^3$. For the permutation model, in an analogous manner, we let $\psi(T, \Theta)$ denote the expected value of $\psi(T, \mathbf{r})$, where the expectation is taken over the choices of $\mathbf{r}$, obtained by randomly shuffling the members of Θ. Then, with a slight overloading of the notation, we define $\psi(n) = \max \psi(T, \Theta)$, where the maximum is over all possible choices of the multi-set Θ and over all sets T of n pairwise-disjoint triangles in $\mathbb{R}^3$. Our goal is to prove sharp bounds on $\psi(n)$ under these two models.

As noted in [2], the permutation model is more general than the density model, in the sense that an upper bound on $\psi(n)$ in the permutation model immediately implies the same bound on $\psi(n)$ in the density model. Although we mostly focus on the permutation model, we also show that some of our bounds can be improved, by a logarithmic factor, in the density model if the underlying pdf is "well behaved," in a sense to be made precise below.

Related work. There is extensive work on bounding the complexity of the union of a set of geometric objects, especially in $\mathbb{R}^2$ and $\mathbb{R}^3$, and optimal or near-optimal bounds have been obtained for many interesting cases. We refer the reader to the survey paper by Agarwal *et al.* [18] for a comprehensive summary of most of the known results on this topic.

The complexity of the union of n arbitrary shapes of constant complexity in $\mathbb{R}^2$ is $\Theta(n^2)$, but the bound improves to near linear for a large class of well behaved planar objects; see [18]. Analogously, the complexity of the union of n arbitrary shapes of constant complexity in $\mathbb{R}^3$ is $\Theta(n^3)$. Over more than a decade, a series of papers have considered geometric objects in $\mathbb{R}^3$ that have some special properties, and derived *near-quadratic* bounds on the complexity of their union. These cases include: a family of arbitrary convex polytopes[4] [8], a family of cylinders (of arbitrary radii) [12], (arbitrarily aligned) congruent cubes in three dimensions [19], a family of fat tetrahedra [13], a family of κ-round objects [4], and Minkowski sums of a family of pairwise-disjoint convex polytopes with a fixed convex polytope [7] or with a fixed ball [3]. Quadratic lower bounds are known for all these cases.

The case of the union of axis-parallel cubes is a highly-structured special instance. If the cubes are *isothetic* (that is, they are all congruent), the complexity of their union is[5] $O(n)$. In dimension d, Boissonnat *et al.* [10] proved that the complexity of the union of n isothetic hypercubes is $\Theta\left(n^{\lfloor d/2 \rfloor}\right)$. If the hypercubes are (axis-parallel but) of arbitrary sizes, the complexity of their union is $\Theta\left(n^{\lceil d/2 \rceil}\right)$ in $\mathbb{R}^d$. These two bounds coincide for even values of d, but there is a gap, by a factor of d, for odd values of d.

There is a rich literature on bounding the complexity of geometric structures under a stochastic model in which the locations of points are drawn randomly from a distribution; see [14, 20, 21] and references therein. The complexity of the union of a set of objects in a semi-stochastic model, in which the locations of the objects were arbitrary but their scaling factors were chosen randomly, was first studied by the authors (with Har-Peled) in [2]. They investigated two planar variants of the stochastic Minkowski-sum union problem. In the first variant, one is given a set S of n pairwise-disjoint line segments in $\mathbb{R}^2$, and one replaces each $e \in S$ by the Minkowski sum $e \oplus r_i B$, where B is the unit disk and the scaling factors r_i are

[4] Here the bound is cubic in the number of polytopes but is only near linear in the number of facets.

[5] In contrast, the complexity of the union of congruent balls in $\mathbb{R}^3$ is quadratic in the worst case; see, e.g., [18].

randomly chosen under either of the above two models. It is shown in [2] that the expected complexity of the union of these sums is $O(n \log n)$. In the second variant, S is a collection of arbitrary pairwise-disjoint convex sets (of constant complexity) in the plane, and then the expected complexity of the union of the corresponding random Minkowski sums (again, with randomly scaled copies of the unit disk) is shown in [2] to be $O(n^{1+\varepsilon})$, for any $\varepsilon > 0$. In both cases, the bounds are a significant improvement over the worst case quadratic bound, and almost match the linear upper bound when all scaling factors are equal—the Minkowski sums then form a collection of *pseudo-disks*. Here we study related random arrangements in higher dimensions, a situation that required some new ideas in order to apply some exisiting techniques.

In a different, but closely related context, Har-Peled and Raichel [16] proved that the expected complexity of the multiplicatively weighted Voronoi diagram of a set of points or line segments in $\mathbb{R}^2$ is $O(n \operatorname{polylog}(n))$ if the weights are randomly chosen under the permutation model.[6] Recall that if the weights are arbitrarily chosen, then the worst-case complexity of the weighted Voronoi diagram is quadratic [9]. Chang *et al.* [11] studied various generalizations of multiplicatively weighted Voronoi diagrams and proved sharp bounds on their expected complexity.

Our results. We have two main results in this paper:

Union of hypercubes. We first study (in Section 2) the interesting special case of the stochastic Minkowski-sum union problem in which each triangle is a point in $\mathbb{R}^3$ and B is an axis-aligned cube in $\mathbb{R}^3$ of side-length 2. That is, we have a set $P = \{p_1, \ldots, p_n\}$ of n points in $\mathbb{R}^3$, and $B = \{x \in \mathbb{R}^3 \mid \|x\|_\infty \leq 1\}$, and for each $i \leq n$, $C_i = r_i B + p_i$ is a cube of side-length $2r_i$ centered at p_i. In other words, we study the complexity of the union of n cubes C_i each having a random (according to each of our models) L_∞ diameter and a fixed center. In fact, we study this problem not only in $\mathbb{R}^3$ but in $\mathbb{R}^d$ for general d. The reader should note that all our definitions (e.g. of $\psi(P, \Theta)$) extend to $\mathbb{R}^d$. Hypercubes in $\mathbb{R}^d$ arise in many applications and the following result indicates that if their diameters are not chosen in an adversarial manner then the complexity of their union is likely to be smaller than the worst case, by nearly a factor of n. Specifically, we prove:

▶ **Theorem 1.1.** *Let P be a set of n points in $\mathbb{R}^d$, let B be the axis-aligned hypercube of side-length 2 in $\mathbb{R}^d$ centered at the origin, and let Θ be a multi-set of scaling factors. Under the permutation model, $\psi(P, \Theta) = O(n \log^2 n)$ for $d = 3$, and $\psi(P, \Theta) = O(n^{\lfloor d/2 \rfloor} \log n)$ for any fixed odd value of $d > 3$. The same bounds hold for the density model.*

Tame distributions. We call a pdf π, with Π as its cdf (cumulative distribution function), *tame* if for every $x > 0$ and for every integer $k \geq 0$,

$$1 - \Pi(kx) \leq (1 - \Pi(x))^k. \tag{1}$$

It is an easy exercise to verify that (1) is satisfied for a large class of pdfs, including uniform, exponential, (one-sided) normal, and log-normal distributions. If the scaling factors are chosen from a tame pdf, then the bounds can be improved by a logarithmic factor:

▶ **Theorem 1.2.** *Let P be a set of n points in $\mathbb{R}^d$, and let B be the hypercube of side-length 2 centered at the origin. If the scaling factors are drawn randomly from a tame pdf π, then $\psi(P, \pi) = O(n \log n)$ for $d = 3$ and $\psi(P, \pi) = O(n^{\lfloor d/2 \rfloor})$ for any fixed odd value of $d > 3$.*

---

[6] Given a point set $P = \{p_1, \ldots, p_n\}$ and a weight $w_i > 0$ for each point p_i, the Voronoi cell of p_i in the multiplicatively weighted Voronoi diagram is $\{x \in \mathbb{R}^d \mid w_i \|x - p_i\| \leq w_j \|x - p_j\| \, \forall j \leq n\}$.

By Theorems 1.1 and 1.2, the expected complexity of the union of a set of axis-aligned hypercubes, whose sizes are chosen randomly, is almost the same as when their sizes are equal. The overall structure of the proof of both theorems is the same, and consists of two stages. The first stage bounds the expected number of *outer* vertices on the union, i.e., the vertices that lie on a $(d-2)$-face or a lower-dimensional face of one of the hypercubes. This is where we had to develop a new technique to exploit the randomness of the input. The second stage then obtains a bound on the number of *inner* vertices, namely vertices formed by the intersection of d facets of d distinct hypercubes, by adapting the argument in [10] and using the bound, derived earlier, on the expected number of outer vertices.

The stochastic Minkowski-sum union problem. Next, we study (in Section 3) the general case in which T is a set of pairwise disjoint triangles in $\mathbb{R}^3$ and B is a convex polytope in $\mathbb{R}^3$ with $O(1)$ faces, and prove the following near-quadratic upper bound:

▶ **Theorem 1.3.** *Let T be a collection of n pairwise-disjoint triangles in $\mathbb{R}^3$, let Θ be a multi-set of scaling factors, and let B be a convex polytope with $O(1)$ vertices. Then the value of $\psi(T,\Theta)$ under the permutation model is $O(n^{2+\varepsilon})$, for any fixed constant $\varepsilon > 0$; the constant of proportionality depends on ε and the complexity of B. The same bound holds for the density model.*

If all sizes r_i are equal, the complexity of $\mathcal{U}$ is $O(n^2 \log n)$ [7], where the constant of proportionality depends on the complexity of B. On the other hand, for a bad layout of the triangles of T and a bad (non-random) choice of sizes, the complexity of $\mathcal{U}$ can be $\Theta(n^3)$; see Section 3 for a lower-bound construction. (The bound in Theorem 1.3 is coarse enough so that drawing the scaling factors from a tame pdf does not seem to affect it.

2 Union of Hypercubes

In this section we prove Theorems 1.1 and 1.2. For a point $x \in \mathbb{R}^d$, we use $\|x\|_\infty$ to denote the L_∞-norm of x. For a compact set $S \subset \mathbb{R}^d$, let $d_\infty(x,S) = \min_{y\in S} \|x-y\|_\infty$. A hypercube C of L_∞-*radius* (or *radius* for brevity) r centered at c is the set $C = \{x \in \mathbb{R}^d \mid \|x-c\|_\infty \le r\}$; the *side-length* of this hypercube is $2r$. Let $P = \{p_1, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^d$, and let B be the (axis-aligned) hypercube of radius 1 centered at the origin. Given a sequence $\mathbf{r} = \langle r_1, \ldots, r_n\rangle$ of non-negative scaling factors, we define, for each $i \le n$, $C_i = r_i B + p_i$ to be a hypercube of radius r_i centered at p_i. Let $\mathcal{C} = \{C_1, \ldots, C_n\}$, and let $\mathcal{U} = \mathcal{U}(\mathcal{C}) = \bigcup_{i=1}^n C_i$ denote the union of the hypercubes.

For simplicity, we assume that the hypercubes of $\mathcal{C}$ are in *general position*, in the sense that no two facets of any pair of hypercubes lie in a common hyperplane. This assumption is made only to simplify the analysis, and the general, possibly degenerate case can be reduced to the general-position setup by means of a suitable small perturbation.

A vertex v in the arrangement $\mathcal{A}(\mathcal{C})$ of $\mathcal{C}$ is the intersection of d facets of the hypercubes in $\mathcal{C}$ and each of these facets is orthogonal to a different coordinate axis. These facets need not belong to distinct hypercubes. We refer to v as an *inner* vertex if the d facets on which it lies belong to d distinct hypercubes, and as an *outer* vertex if at least two of these facets belong to the same hypercube. The *level* of v is j if it lies in the interior of exactly j hypercubes of $\mathcal{C}$; the level of every vertex on $\partial\mathcal{U}$ is 0. For $j \ge 0$, let $\mathcal{I}_j(\mathcal{C})$ (resp. $\mathcal{O}_j(\mathcal{C})$) denote the number of inner (resp. outer) vertices at level j in $\mathcal{A}(\mathcal{C})$.

If the sequence $\mathbf{r}$ of scaling factors is chosen randomly, using either the permutation model or the density model, $\mathcal{I}_j(\mathcal{C})$ and $\mathcal{O}_j(\mathcal{C})$ are random variables. For a sequence Θ of scaling factors, let $\bar{\mathcal{I}}_j(P,\Theta) = \mathsf{E}[\mathcal{I}_j(\mathcal{C})]$ denote the expected value of $\mathcal{I}_j(\mathcal{C})$ in the permutation

model, over a random permutation of Θ. For any $n > 0$ and for any $j \leq n - d$, let $\bar{\mathcal{I}}_j(n) = \max \bar{\mathcal{I}}_j(P, \Theta)$ denote the expected number of inner vertices of level j under the permutation model, where the maximum is taken over all point sets of size n. We define $\bar{\mathcal{O}}_j(P, \Theta)$ and $\bar{\mathcal{O}}_j(n)$ similarly. By definition (recall the terminology introduced earlier), $\psi(P, \Theta) = \bar{\mathcal{I}}_0(P, \Theta) + \bar{\mathcal{O}}_0(P, \Theta)$ and $\psi(n) \leq \bar{\mathcal{I}}_0(n) + \bar{\mathcal{O}}_0(n)$.

Finally, in the density model, which we use explicitly only when the scaling factors are chosen randomly from a tame pdf π, we define $\bar{\mathcal{I}}_j(P, \pi)$ and $\bar{\mathcal{O}}_j(P, \pi)$ as the expected number of inner and outer level-j vertices, respectively, and we also define $\bar{\mathcal{I}}_j(n)$ and $\bar{\mathcal{O}}_j(n)$ as the maximum value of these respective quantities, where the maximum is taken over all sets P of n points and all tame pdf's.[7]

Our goal is to bound $\bar{\mathcal{I}}_0(n)$ and $\bar{\mathcal{O}}_0(n)$. Our strategy is first to derive an upper bound for $\bar{\mathcal{O}}_0(n)$, either for the permutation model or for the density model for a tame pdf, and then use either of these bounds in a charging scheme that leads to a corresponding bound on $\bar{\mathcal{I}}_0(n)$. The following lemma will be crucial in proving the bound on $\bar{\mathcal{I}}_0(n)$.

▶ **Lemma 2.1.** $\bar{\mathcal{O}}_1(n) = O(\bar{\mathcal{O}}_0(n))$.

To keep the presentation simple, we first bound $\bar{\mathcal{O}}_0(n)$ in $\mathbb{R}^3$ (Section 2.1) and then extend the argument to higher dimensions (Section 2.2). Finally, we prove an upper bound on $\bar{\mathcal{I}}_0(n)$ in $\mathbb{R}^d$, for any odd dimension $d \geq 3$ (Section 2.3).

2.1 Outer vertices in 3D

An outer vertex of $\mathcal{U}(\mathcal{C})$ in $\mathbb{R}^3$ is either a vertex of a cube in $\mathcal{C}$ or the intersection point of an edge of a cube C_i with a face of another cube C_j. There are $O(n)$ vertices of the first type, so it suffices to bound the expected number of the second type of outer vertices.

We fix a point, say p_1, of P and an edge e of the cube $C_1 = p_1 + r_1 B$ centered at p_1. We bound the expected number of outer vertices of $\mathcal{U}$ lying on e. Note that e is not a fixed segment in $\mathbb{R}^3$—its physical location in $\mathbb{R}^3$ as well as its length depend on the value of r_1. Nevertheless, we can combinatorially define e so that it refers to a fixed edge (one of the 12 possible edges) of C_1. For simplicity, we assume that e is parallel to the z-axis. We bound the expected number of vertices lying on e, first for the permutation model, and then for the density model for a tame pdf.

The permutation model. We are given a sequence $\Theta = \langle 0 \leq \theta_1 \leq \cdots \leq \theta_n \rangle$ of n arbitrary non-negative scaling factors. We choose a random permutation σ on $[1 : n]$, and assign $r_i := \theta_{\sigma(i)}$.

Let $\mu(e, P, \Theta)$ denote the expected number of connected components of $\partial\mathcal{U} \cap e$. The expected number of outer vertices on e is at most $2\mu(e, P, \Theta)$. For a fixed value $\theta_i \in \Theta$ for r_1, let $\mu(e, P, \Theta \mid r_1 = \theta_i)$ denote the conditional expected number of connected components of $\partial\mathcal{U} \cap e$ assuming that $r_1 = \theta_i$. With this fixed choice of r_1, e is a segment of length $2\theta_i$, lying at a fixed location in $\mathbb{R}^3$. Observe that

$$\mu(e, P, \Theta) = \frac{1}{n} \sum_{i=1}^{n} \mu(e, P, \Theta \mid r_1 = \theta_i). \tag{2}$$

The following probabilistic lemma is the main technical tool used in the analysis in $\mathbb{R}^3$.

[7] One can also take the maximum over all pdfs, but then the various expectations are at most the corresponding quantities under the permutation model, and no better bounds are known so far. We will therefore be interested only in tame pdfs.

▶ **Lemma 2.2.** *For any* $1 \le k \le n$, $\mu(e, P, \Theta \mid r_1 = \theta_{n-k+1}) \le \frac{4n}{k}$.

Proof. Let $r = \theta_{n-k+1}$. Let e^+ (resp. e^-) denote the top (resp. bottom) half of e, i.e., the portion lying above (resp. below) p_1. Further partition e^+ into two parts e_1^+ and e_2^+, each of length $r/2$, and similarly partition e^-. We bound the expected number of connected components of $\mathcal{U}$ lying on e_1^+, e_2^+, e_1^- and e_2^- separately. Let $e' = e_1^+$, the bounds for the other parts of e are derived analogously.

Let $P(e^*) = \langle p_{i_1}, p_{i_2}, \ldots, p_{i_{n-1}} \rangle$ denote the sorted sequence of the points of $P \setminus \{p_1\}$ by increasing values of $\delta_i = d_\infty(p_i, e')$. We fix an assignment $r_2, \ldots, r_n$ of scaling factors (which is a permutation of $\Theta \setminus \{r\}$) to the points of $P \setminus \{p_1\}$, so that p_i is assigned the scaling factor r_i, for $i = 2, \ldots, n$. Let p_{i_j} be the first point in the sequence $P(e^*)$ for which $r_{i_j} \ge r$. If $\delta_{i_j} \le r/2$ then, since $|e'| = r/2$, it follows by the triangle inequality that $e' \subseteq C_{i_j}$ and there are no connected components of $\mathcal{U}$ on e'.

So assume that $\delta_{i_j} > r/2$. Then, since the sequence $P(e^*)$ is sorted by distance to e', it follows that $\delta_{i_\ell} > r/2$ for every $\ell \ge j$. Hence, any cube C_{i_ℓ}, for $\ell \ge j$, will intersect e' only if $r_{i_\ell} \ge r/2$. But then $e' \cap C_{i_\ell}$ must contain an endpoint of e'. It follows that all the cubes C_{i_ℓ}, for $\ell \ge j$, can contribute at most one connected component to $\partial\mathcal{U} \cap e'$. The cubes C_{i_ℓ}, for $\ell < j$, can increase the number of components by at most $j - 1$, so in total we get at most j connected components of $\mathcal{U}$ on e'.

As argued in [2, Lemma 3.3] (see also [15]), the expected value of the index j of the first appearance of one of $k-1$ distinguished elements in a random permutation of $n-1$ elements is n/k, so the expected value of j is n/k. This gives an average of at most n/k connected components of $\partial\mathcal{U} \cap e'$. From this the lemma follows. ◀

Plugging Lemma 2.2 into (2), we obtain $\mu(e, P, \Theta) = O(\log n)$. Summing this bound over all $O(n)$ edges of cubes in $\mathcal{C}$ and using Lemma 2.1, we obtain the following.

▶ **Corollary 2.3.** *Let P be a set of n points in $\mathbb{R}^3$, and let Θ be a multi-set of n non-negative scaling factors. Then $\bar{\mathcal{O}}_0(P, \Theta), \bar{\mathcal{O}}_1(P, \Theta) = O(n \log n)$.*

The density model for tame distributions. Next, we show that if the scaling factors of the cubes in $\mathcal{C}$ are chosen randomly and independently from a tame pdf then the expected number of outer vertices on a fixed edge g of any cube, say for concreteness the cube C_1 centered at p_1, is only $O(1)$. For simplicity, we assume that p_1 is at the origin. Let π be a tame pdf and Π its cdf. By definition, $1 - \Pi(kx) \le (1 - \Pi(x))^k$ for all $x > 0$ and for each $k \ge 1$. Adapting the previous notation, let $\mu(g, P, \pi)$ denote the expected number of connected components on $\partial\mathcal{U} \cap g$.

▶ **Lemma 2.4.** $\mu(g, P, \pi) \le 2e^3/(e-1)$.

Proof. Assume, as before, that g is parallel to the z-axis, and let g^+ (resp. g^-) denote the top (resp. bottom) half of g, i.e., the portion of g lying above (resp. below) p_1. Recall that g^+ is a random segment whose length and location depend on the value of r_1, the scaling factor assigned to p_1. We bound $\mu(g^+, P, \pi)$, the expected number of connected components of $\partial\mathcal{U} \cap g^+$. A symmetric argument will bound $\mu(g^-, P, \pi)$, and the sum of these two quantities will bound $\mu(g, P, \pi)$.

Set $\alpha = \Pi^{-1}(1 - 1/e)$ and let $I_k = [k\alpha, (k+1)\alpha)$, for $k = 0, 1, \ldots$. For a fixed value of r_1 we partition g^+ into $\ell + 1$ intervals, $\Delta_0, \ldots, \Delta_\ell$, such that, for $k < \ell$, the projection of Δ_k on the z-axis is I_k and the projection of Δ_ℓ is contained in I_ℓ. For convenience we define Δ_k, for $k > \ell$, to be an empty interval. Note that ℓ is a random variable whose value depends

on r_1. Let $\mu(\Delta_k) := \mu(\Delta_k, P, \pi)$ denote the expected number of connected components of $\partial\mathcal{U} \cap \Delta_k$. Then

$$\mu(g^+, P, \pi) \leq \sum_{k=0}^{\infty} \mu(\Delta_k). \tag{3}$$

The probability that Δ_k is not empty is equal to $\Pr[r_1 \geq k\alpha]$, which is

$$\Pr[r_1 \geq k\alpha] = 1 - \Pi(k\alpha) \leq (1 - \Pi(\alpha))^k = e^{-k}. \tag{4}$$

Fix a value of r_1 for which Δ_k is not empty. Let $P(\Delta_k) = \langle p_{i_1}, p_{i_2}, \ldots, p_{i_{n-1}} \rangle$ denote the sorted sequence of the points of $P \setminus \{p_1\}$ by increasing values of $\delta_i = d_\infty(p_i, \Delta_k)$. Fix an assignment $r_2, \ldots, r_n$ of scaling factors to the points of $P \setminus \{p_1\}$, so that p_i is assigned the scaling factor r_i, drawn from π, for $i = 2, \ldots, n$.

Let p_{i_j} be the first point in the sequence $P(\Delta_k)$ for which $r_{i_j} \geq 2\alpha$. If $\delta_{i_j} \leq \alpha$ then since $|\Delta_k| \leq \alpha$ it follows by the triangle inequality that $\Delta_k \subseteq C_{i_j}$, and there are no connected components of $\mathcal{U}$ on Δ_k.

So let us assume that $\delta_{i_j} > \alpha$. Then, since the sequence $P(\Delta_k)$ is sorted by distance to Δ_k, it follows that $\delta_{i_\ell} \geq \alpha$ for every $\ell \geq j$. Thus, for a cube C_{i_ℓ}, for $\ell \geq j$, to intersect Δ_k, we must have that $r_{i_\ell} \geq \alpha$. But then $\Delta_k \cap C_{i_\ell}$ must contain an endpoint of Δ_k. It follows that all the cubes C_{i_ℓ}, for $\ell \geq j$, can contribute at most one connected component to $\partial\mathcal{U} \cap \Delta_k$. The cubes C_{i_ℓ}, for $\ell < j$, can increase the number of components by at most $j - 1$, so in total we get at most j connected components of $\mathcal{U}$ on Δ_k.

Notice that j is a geometric random variable, where the success probability of each trial is $p = 1 - \Pi(2\alpha) \leq (1 - \Pi(\alpha))^2 = 1/e^2$. The expectation of j is thus $1/p = e^2$. Therefore $\mu(\Delta_k \mid \Delta_k \neq \emptyset) \leq e^2$.

Combining this with the bound on the probability that Δ_k is not empty, in Equation (4), we get that $\mu(\Delta_k) = \Pr[\Delta_k \neq \emptyset] \cdot \mu(\Delta_k \mid \Delta_k \neq \emptyset) \leq e^{-k+2}$. Substituting these bounds into Equation (3), the lemma follows. ◀

Putting everything together and using Lemma 2.1, we obtain the following:

▶ **Lemma 2.5.** *Let P be a set of n points in $\mathbb{R}^3$ and π a tame pdf. Then $\bar{\mathcal{O}}_0(P, \pi), \bar{\mathcal{O}}_1(P, \pi) = O(n)$.*

2.2 Outer vertices in higher odd dimensions

We now bound $\bar{\mathcal{O}}_0(n)$ in $\mathbb{R}^d$, for any fixed odd value $d > 3$. We fix a point of P, say, p_1, and a $(d-2)$-dimensional face f of C_1. We bound the expected number of outer vertices lying on f. As in the previous case, we first obtain the bound for the permutation model and then for the density model (with a tame pdf).

The permutation model. Let Θ denote a fixed set of scaling factors, and let $\mu(\mathsf{f}, P, \Theta)$ denote the expected number of vertices of $\mathcal{U}$ lying on f. We bound the conditional expectation $\mu(\mathsf{f}, P, \Theta \mid r_1 = \theta_i)$, assuming that the scaling factor of p_1 is θ_i, and then use (2) to bound $\mu(\mathsf{f}, P, \Theta)$.

Recall that once the value r_1 is fixed to θ_i, the hypercube $C_1 = p_1 + r_1 B$ is also fixed, and so is its face f. The problem is therefore to bound the expected number of outer vertices lying on a fixed $(d-2)$-dimensional hypercube f. A naive approach, based on induction on d within f does not work. We use instead a different approach.

The following lemma is analogous to Lemma 2.2.

▶ **Lemma 2.6.** *For any* $1 \leq k \leq n$, $\mu(\mathsf{f}, P, \Theta \mid r_1 = \theta_{n-k+1}) = O\left(\frac{n^{\lfloor d/2 \rfloor}}{k}\right)$.

Proof. The proof is similar to that of Lemma 2.2. Let $r = \theta_{n-k+1}$. We partition f into 4^{d-2} hypercubes (subfaces) each of radius $r/4$. Fix such a smaller hypercube g. Let $\mu(\mathsf{g}, P, \Theta \mid r_1 = r)$ be the expected number of outer vertices of $\mathcal{U}$ incident to g, conditioned on the choice of r_1.

Let $\delta_i = d_\infty(p_i, f)$ and let $P(\mathsf{g}) = \langle p_{i_1}, p_{i_2}, \ldots, p_{i_{n-1}} \rangle$ denote the sorted sequence of the points of $P \setminus \{p_1\}$ by increasing values of $\delta_i = d_\infty(p_i, \mathsf{g})$. We fix an assignment $r_2, \ldots, r_n$ of scaling factors from $\Theta \setminus \{r\}$ to the points of $P \setminus \{p_1\}$, so that p_i is assigned the scaling factor r_i, for $i = 2, \ldots, n$. Let $\hat{\mathcal{C}} = \{C_i := p_i + r_i B \mid p_i \in P \setminus \{p_1\}\}$ be the resulting set of hypercubes. Let $U_\mathsf{g} = \mathcal{U}(\hat{\mathcal{C}}) \cap \mathsf{g}$. Our goal is to bound the number of vertices of U_g. Let F be the $(d-2)$-dimensional flat spanned by g. For a hypercube $C_i \in \hat{\mathcal{C}}$, let $K_i = C_i \cap F$. If nonempty, K_i is a $(d-2)$-dimensional hypercube of radius r_i.

Let p_{i_j} be the first point in the sequence $P(\mathsf{g})$ for which $r_{i_j} \geq r$. Set $\mathcal{K}_< = \{K_{i_\ell} \mid \ell < j \wedge K_{i_\ell} \neq \emptyset\}$ and $\mathcal{K}_\geq = \{K_{i_\ell} \mid \ell \geq j \wedge K_{i_\ell} \neq \emptyset\}$. By definition, $|\mathcal{K}_<| < j$ and $U_\mathsf{g} = \mathcal{U}(\mathcal{K}_< \cup K_\geq) \cap \mathsf{g}$.

If $\delta_{i_j} \leq r/2$ then, since the side-length of g is $r/2$, it follows by the triangle inequality that $f \subseteq C_{i_j}$ and the number of vertices in U_g is 0. So assume that $\delta_{i_j} > r/2$. Then, since the p_i's are sorted by their distance to g, it follows that $\delta_{i_\ell} > r/2$ for every $\ell \geq j$. Hence, for a cube C_{i_ℓ} with $\ell \geq j$ to intersect g we must have $r_{i_\ell} > r/2$.

▶ **Lemma 2.7.** *Let* g *be a* $(d-2)$*-dimensional hypercube of radius* $r/4$*. For any* $(d-2)$*-dimensional hypercube* K *of radius at least* $r/4$*, there is another hypercube* $\tilde{K}$ *of radius exactly* $r/4$ *such that* $K \cap \mathsf{g} = \tilde{K} \cap \mathsf{g}$.

Since all hypercubes in $\mathcal{K}_\geq$ have radius at least $r/4$, we obtain, by applying Lemma 2.7 to all of them, a collection $\tilde{\mathcal{K}}_\geq = \{\tilde{K}_i \mid K_i \in \mathcal{K}_\geq\}$ such that $\mathcal{U}(\mathcal{K}_\geq) \cap \mathsf{g} = \mathcal{U}(\tilde{\mathcal{K}}_\geq) \cap \mathsf{g}$. It follows that $U_\mathsf{g} = \mathcal{U}(\mathcal{K}_< \cup \tilde{\mathcal{K}}_\geq) \cap \mathsf{g}$. It therefore suffices to bound the number of vertices of $\tilde{U}_\mathsf{g} = \mathcal{U}(\mathcal{K}_< \cup \tilde{\mathcal{K}}_\geq)$, and also the number of vertices of $\tilde{U}_\mathsf{g} \cup \mathsf{g}$ (in order to upper bound the number of vertices in U_g which are on the boundary of g). We bound the number of vertices of $\tilde{U}_\mathsf{g}$, the analysis for $\tilde{U}_\mathsf{g} \cup \mathsf{g}$ is similar.

We call a vertex v of $\tilde{U}_\mathsf{g}$ *pure* if it does not lie on the boundary of any hypercube in $\mathcal{K}_<$, otherwise we call it *mixed*. A pure vertex is also a vertex of $\mathcal{U}(\tilde{\mathcal{K}}_\geq)$. Since $\tilde{\mathcal{K}}_\geq$ is a set of at most n congruent hypercubes in $\mathbb{R}^{d-2}$, the number of vertices in $\mathcal{U}(\tilde{\mathcal{K}}_\geq)$ is $O(n^{\lfloor (d-2)/2 \rfloor}) = O(n^{\lfloor d/2 \rfloor - 1})$.

Each mixed vertex of $\tilde{U}_f$ is incident to at least one facet of a hypercube in $\mathcal{K}_<$. Fix such a facet ϕ, and let Φ be the $(d-3)$-dimensional flat spanned by ϕ. For each hypercube $K \in \mathcal{K}_< \cup \tilde{\mathcal{K}}_\geq$, let $K^* = K \cap \Phi$ be the (possibly empty) $(d-3)$-dimensional hypercube contained in Φ. Set $\mathcal{K}^* = \{K^* \mid K \in \mathcal{K}_< \cup \tilde{\mathcal{K}}_\geq\}$. A mixed vertex incident on ϕ is also a vertex of $\mathcal{U}(\mathcal{K}^*)$. Since $d-3$ is even, the result in [10] implies the number of such vertices is (always) $O\left(n^{(d-3)/2}\right) = O\left(n^{\lfloor d/2 \rfloor - 1}\right)$. Multiplying the bound by the number of facets in the hypercubes of $\mathcal{K}_<$, the total number of mixed vertices is $O(|\mathcal{K}_<| n^{\lfloor d/2 \rfloor - 1})$. Hence, the total number of vertices in U_g is $O(|\mathcal{K}_<| n^{\lfloor d/2 \rfloor - 1} + n^{\lfloor d/2 \rfloor - 1}) = O(jn^{\lfloor d/2 \rfloor - 1})$. As in the proof of Lemma 2.2, the expected value of j (assuming that $r_1 = \theta_{n-k+1}$) is $O(n/k)$. Hence, $\mu(\mathsf{g}, P, \Theta \mid r_1 = \theta_{n-k+1}) = O\left(\frac{n^{\lfloor d/2 \rfloor}}{k}\right)$. The lemma follows by summing this over all 4^{d-2} subfaces g in the subdivision of f. ◀

Plugging Lemma 2.6 in (2), we obtain that $\mu(\mathsf{f}, P, \Theta) = \frac{1}{n} \sum_{k=1}^{n} O\left(\frac{n^{\lfloor d/2 \rfloor}}{k}\right) = O\left(n^{\lfloor d/2 \rfloor - 1} \log n\right)$. Finally, summing this bound over all points of P and using Lemma 2.1, we obtain the main result of this section:

▶ **Lemma 2.8.** *Let P be a set of n points in $\mathbb{R}^d$ for some fixed odd value of $d > 3$, and let Θ be a multi-set of scaling factors. Then $\bar{\mathcal{O}}_0(P,\Theta), \bar{\mathcal{O}}_1(P,\Theta) = O\left(n^{\lfloor d/2 \rfloor} \log n\right)$.*

The density model for tame distributions. Next, we show (see the full version for details) that if the scaling factors of the cubes in $\mathcal{C}$ are chosen randomly and independently from a tame pdf π (whose cdf is denoted by Π) then the expected number of outer vertices on a fixed $(d-2)$-dimensional face f of any cube, say of C_1, is $O\left(n^{\lfloor d/2 \rfloor - 1}\right)$. That is,

▶ **Lemma 2.9.** $\mu(\mathsf{f}, P, \pi) = O(n^{\lfloor d/2 \rfloor - 1})$.

Putting everything together, using Lemma 2.1, we obtain the following:

▶ **Lemma 2.10.** *Let P be a set of n points in $\mathbb{R}^d$ for some fixed odd value of $d > 3$, and let π be a tame pdf. Then $\bar{\mathcal{O}}_0(P,\Theta), \bar{\mathcal{O}}_1(P,\Theta) = O\left(n^{\lfloor d/2 \rfloor}\right)$.*

2.3 Inner vertices

We now bound the expected number of inner vertices in $\mathcal{U}(\mathcal{C})$ in $\mathbb{R}^d$ under both the permutation model and the density model for tame distributions. We first consider a fixed assignment of scaling factors to the points of P, so $\mathcal{C} = \{C_1, \ldots, C_n\}$ is a fixed set of n hypercubes in $\mathbb{R}^d$. We obtain a recurrence for $\mathcal{I}_0(\mathcal{C})$ for this fixed $\mathcal{C}$, by closely following the charging scheme proposed in Boissonnat *et al.* [10], which shows that there are many distinct outer vertices of level 0 plus vertices (inner or outer) of level 1 in the neighborhood of any inner vertex of $\mathcal{U}(\mathcal{C})$. This gives an upper bound on $\mathcal{I}_0(\mathcal{C})$ in terms of $\mathcal{I}_1(\mathcal{C})$, $\mathcal{O}_0(\mathcal{C})$, and $\mathcal{O}_1(\mathcal{C})$. Next, we consider a random assignment of scaling factors and obtain a recurrence for $\bar{\mathcal{I}}_0(n)$ in terms of $\bar{\mathcal{I}}_1(n)$, $\bar{\mathcal{O}}_0(n)$, and $\bar{\mathcal{O}}_1(n)$, from the recurrence for $\mathcal{I}_0(\mathcal{C})$. We use results from Sections 2.1 and 2.2 to bound the last two terms, and we use a counting argument to bound $\bar{\mathcal{I}}_1(n)$ in terms of $\bar{\mathcal{I}}_0(n-1)$.

The charging scheme. Let v be an inner vertex of $\partial\mathcal{U}$, lying on d facets $\mathbf{f} = \{f_1, \ldots, f_d\}$ of a set $\mathbf{C} = \{C_1, \ldots, C_d\}$ of d distinct hypercubes, such that f_i is a facet of C_i orthogonal to the x_i-axis. For each $i \leq d$, let γ_i denote the intersection segment of the $d-1$ facets in $\mathbf{f} \setminus \{f_i\}$. The segment γ_i is parallel to the x_i-axis. The vertex v is incident to all d segments $\gamma_1, \ldots, \gamma_d$. For each i, we trace the segment γ_i from v entering the hypercube C_i (recall that v lies on the x_i-orthogonal facet f_i of C_i) until we encounter another vertex, w_i, of the arrangement $\mathcal{A}(\mathcal{C})$ of $\mathcal{C}$. Three cases can arise:

- **(i)** The vertex w_i lies on the facet of C_i opposite to f_i (i.e., the other facet of C_i orthogonal to the x_i-axis). This event can happen only if C_i is smaller than all the other $d-1$ hypercubes in $\mathbf{C}$, i.e., it can happen for at most one index i.
- **(ii)** The vertex w_i is an outer vertex at level 1 (it is contained in C_i), lying on a $(d-2)$-face of one of the hypercubes $C_j \in \mathbf{C} \setminus \{C_i\}$.
- **(iii)** The vertex w_i is an inner vertex at level 1 (it is contained in C_i), lying on the boundary of another hypercube $C_0 \in \mathcal{C} \setminus \mathbf{C}$.

In the first case, we simply ignore this segment γ_i and the vertex w_i—at most one of the d segments will be ignored. In cases (ii) and (iii), v charges w_i one unit. This way, each inner vertex v of $\mathcal{U}$ receives at least $d-1$ units of charge.

An outer vertex w at level 1, lying on a $(d-2)$-face ϕ of C_j, can be reached from an inner vertex only along one of the two corresponding facets of C_j (in a direction normal to the other facet containing ϕ), so w is charged at most twice by a type (ii) event.

An inner vertex w at level 1 can be charged at most d times, once along each of its d incident intersection segments. Suppose w is charged s times. If $s \leq 1$, then w pays one unit of charge for its unique charging inner vertex (or does not pay at all). If $s > 1$, we distribute $s-1$ of the s units of charge to nearby outer vertices at level 0 or 1 so that each outer vertex is charged at most once (at most one inner vertex can pass a charge to it), and w also pays only one unit of charge. The charge distribution scheme culminates in the following recurrence.

$$(d-1)\mathcal{I}_0(\mathcal{C}) \leq \mathcal{I}_1(\mathcal{C}) + 3\mathcal{O}_1(\mathcal{C}) + \mathcal{O}_0(\mathcal{C}). \tag{5}$$

Recurrence for a fixed assignment. We now obtain a recurrence for $\mathcal{I}_0(\mathcal{C})$ by using the following lemma; see [10] for the easy proof.

▶ **Lemma 2.11.** $\displaystyle\sum_{i=1}^{n} \mathcal{I}_0(\mathcal{C} \setminus \{C_i\}) = (n-d)\mathcal{I}_0(\mathcal{C}) + \mathcal{I}_1(\mathcal{C}).$

Using Lemma 2.11 and (5), we obtain the following recurrence.

$$\begin{aligned}(n-1)\mathcal{I}_0(\mathcal{C}) &= (n-d)\mathcal{I}_0(\mathcal{C}) + (d-1)\mathcal{I}_0(\mathcal{C}) \\ &\leq (n-d)\mathcal{I}_0(\mathcal{C}) + \mathcal{I}_1(\mathcal{C}) + 3\mathcal{O}_1(\mathcal{C}) + \mathcal{O}_0(\mathcal{C}) \qquad \text{(Eq. (5))} \\ &= \sum_{i=1}^{n} \mathcal{I}_0(\mathcal{C} \setminus \{C_i\}) + 3\mathcal{O}_1(\mathcal{C}) + \mathcal{O}_0(\mathcal{C}) \qquad \text{(Lemma 2.11).}\end{aligned} \tag{6}$$

Recurrence for a random assignment. We remind the reader that so far the analysis has been applied to a fixed assignment of scaling factors to the points of P, that is, to a fixed set $\mathcal{C}$ of hypercubes. We now want to replace (6) by the expectations of the terms appearing there, with respect to a random assignment of the scaling factors. By linearity of expectation,

$$(n-1)\mathsf{E}[\mathcal{I}_0(\mathcal{C})] \leq \sum_{i=1}^{n} \mathsf{E}[\mathcal{I}_0(\mathcal{C} \setminus \{C_i\})] + 3\mathsf{E}[\mathcal{O}_1(\mathcal{C})] + \mathsf{E}[\mathcal{O}_0(\mathcal{C})]. \tag{7}$$

For a fixed choice of the scaling factor r_i of C_i, the assignment of the remaining scaling factors to the elements of $\mathcal{C} \setminus \{C_i\}$ is a random permutation (of the $n-1$ elements in $\Theta \setminus \{r_i\}$) under the permutation model, and a random assignment under the density model. The conditional expected value of $\mathcal{I}_0(\mathcal{C} \setminus \{C_i\})$ over any fixed choice of r_1 is thus at most $\bar{\mathcal{I}}_0(n-1)$, and therefore the same also holds for the unconditional expected value of $\mathcal{I}_0(\mathcal{C} \setminus \{C_i\})$. Hence, we obtain the following:

$$(n-1)\bar{\mathcal{I}}_0(n) \leq n\bar{\mathcal{I}}_0(n-1) + 3\bar{\mathcal{O}}_1(n) + \bar{\mathcal{O}}_0(n). \tag{8}$$

Dividing both sides by $n(n-1)$ and setting $\chi(m) = \frac{1}{m}\bar{\mathcal{I}}_0(m)$, we obtain

$$\chi(n) \leq \chi(n-1) + \frac{1}{n(n-1)}(3\bar{\mathcal{O}}_1(n) + \bar{\mathcal{O}}_0(n)). \tag{9}$$

We now substitute the bounds on $\bar{\mathcal{O}}_0(n)$ and $\bar{\mathcal{O}}_1(n)$ from Section 2.1 and Section 2.2 for both the permutation model and the density model for tame distributions. For example, for the permutation model in $\mathbb{R}^3$, $\bar{\mathcal{O}}_0(n), \bar{\mathcal{O}}_1(n) = O(n \log n)$. Therefore, we obtain $\chi(n) \leq \chi(n-1) + \frac{b \log n}{n}$, where $b > 0$ is a constant. Hence, $\chi(n) = O(\log^2 n)$ and $\bar{\mathcal{I}}_0(n) = O(n \log^2 n)$ in this case. For any fixed odd $d > 3$ and still under the permutation model, we obtain

$\chi(n) \leq \chi(n-1) + b'n^{\lfloor d/2 \rfloor - 2} \log n$, where b' is another constant. Since $d > 3$, the solution of the above recurrence is $\chi(n) = O(n^{\lfloor d/2 \rfloor - 1} \log n)$ and thus $\bar{\mathcal{I}}_0(n) = O(n^{\lfloor d/2 \rfloor} \log n)$. This completes the proof of Theorem 1.1.

If the scaling factors of all points are chosen from a tame pdf, the bounds for $\bar{\mathcal{O}}_0(n), \bar{\mathcal{O}}_1(n)$ can be obtained using Lemmas 2.5 and 2.10. This implies that in this case $\bar{\mathcal{I}}_0(n) = O(n \log n)$ for $d = 3$ and $\bar{\mathcal{I}}_0(n) = O(n^{\lfloor d/2 \rfloor})$ for any fixed $d > 3$. This completes the proof of Theorem 1.2.

3 Union of Stochastic Minkowski Sums

We now consider the general (three-dimensional) stochastic Minkowski-sum union problem defined in the introduction, i.e., we now have a set $T = \{\triangle_1, \ldots, \triangle_n\}$ of n pairwise-disjoint triangles in $\mathbb{R}^3$ and a convex polytope B with $O(1)$ vertices. Let $\mathbf{r} = \langle r_1, \ldots, r_n \rangle$ be a sequnce of n non-negative scaling factors. $C_i = \triangle_i \oplus r_i B$ is a convex polytope with $O(1)$ vertices. Set $\mathcal{C} = \{C_i \mid 1 \leq i \leq n\}$. Let $\Theta = \langle \theta_1, \ldots, \theta_n \rangle$ be a sequence of n scaling factors. Our goal is to bound the expected number of vertices in $\mathcal{U}(T, \mathbf{r})$, denoted by $\psi(T, \Theta)$, when $\mathbf{r}$ is chosen from Θ using the permutation model, i.e., we choose a random permutation σ of $[1 : n]$ and set $r_i = \theta_{\sigma(i)}$. For simplicity, we assume that T, B, Θ are such that $\mathcal{C}$ is in general position for any permutation of Θ. Let $\mathcal{F}(T, \mathbf{r})$ denote the complement of $\mathcal{U}(T, \mathbf{r})$.

As in Section 2, we first analyze a certain conditional expectation and then bound the overall expected value $\psi(T, \Theta)$. A crucial ingredient in our analysis is the following technical lemma, adapted from Aronov and Sharir [5, Slicing Theorem] (see also [6, Chopping Theorem]).

▶ **Lemma 3.1.** *Let $R \subseteq T$ be a subset of t triangles and let $\mathbf{r} = \langle r, r, \ldots, r \rangle$. Then (a) the complexity of $\mathcal{U}(T, \mathbf{r})$ is $O(t^2 \log t)$, and (b) there exists a decomposition $\mathcal{F}^{\nabla}$ of $\mathcal{F} := \mathcal{F}(R, \mathbf{r})$ into $O(t^2 \log t)$ tetrahedra with pairwise-disjoint interiors.*

We note that each tetrahedron in $\mathcal{F}^{\nabla}$ is defined by at most s triangles of T, where s is a constant that specifies the (maximum) number of input triangles that are needed to define the features (vertices, edges, and facets) of the corresponding tetrahedron. Moreover, any s or fewer triangles define at most $O(1)$ tetrahedra. We thus obtain the following lemma:

▶ **Lemma 3.2.** *The overall number of tetrahedra that can ever arise in the decomposition $\mathcal{F}^{\nabla}(R, \mathbf{r})$ for any subset $R \subseteq T$, where all the scaling factors in $\mathbf{r}$ are equal, is $O(n^s)$.*

The recursive partition scheme. We now follow, or rather adapt, the recursive scheme used in [2]. Specifically, fix a parameter t, whose value will be determined later, and put $\rho = \theta_{n-t}$, the $(t+1)$-st largest scaling factor in Θ. Let $\Theta^<$ be the sequence of the $n - t$ smallest values in Θ. We fix a subset $T^> \subseteq T$ of t triangles, and set $T^< = T \setminus T^>$. Let $\psi(T, \Theta \mid T^>)$ denote the conditional expected complexity of $\mathcal{U}$ conditioned over the subset of those permutations σ that assign the t largest scaling factors in Θ to the triangles of $T^>$; the restriction of a random permutation from this subset to $[1 : n - t]$ (i.e., the part of the permutation that assigns the scaling factors of $\Theta^<$ to the triangles in $T^<$) is a random permutation, i.e., each permutation of $[1 : n - t]$ is equally likely to arise.

We first obtain a bound on $\psi(T, \Theta \mid T^>)$. For each $\triangle_i \in T$, put $\bar{C}_i = \triangle_i \oplus \rho B$. Set $\bar{\mathcal{C}}^> = \{\bar{C}_i \mid \triangle_i \in T^>\}$ and $\bar{\mathcal{U}} := \mathcal{U}(\bar{\mathcal{C}}^>)$. Let $\bar{\mathcal{F}}$ denote the complement of $\bar{\mathcal{U}}$. Since the $\triangle_i$'s are pairwise disjoint and we now add the same convex polytope ρB to each of them, Lemma 3.1(a) implies that $\bar{\mathcal{U}}$ has $O(t^2 \log t)$ complexity. Moreover, by Lemma 3.1(b), there exists a decomposition, denoted as $\bar{\mathcal{F}}^{\nabla}$, of $\bar{\mathcal{F}}$ into $O(t^2 \log t)$ tetrahedra with pairwise-disjoint interiors. For each tetrahedron τ of $\bar{\mathcal{F}}^{\nabla}$, let $T^<_\tau = \{\triangle_i \in T^< \mid \bar{C}_i \cap \tau \neq \emptyset\}$; set $n_\tau = |T^<_\tau|$.

Note that $\bar{\mathcal{U}}$, $\bar{\mathcal{F}}$, $\bar{\mathcal{F}}^{\nabla}$, and the sets $T_\tau^<$ are independent of the choice of permutation; they only depend on ρ and the set $T^>$.[8]

We now fix a permutation σ, and thus an assignment of scaling factors, such that the largest t scaling factors are assigned to the triangles of $T^>$. Thus the scaling factors of all triangles in $T^<$ (resp. $T^>$) are at most (resp. at least) ρ. Therefore $C_i \subseteq \bar{C}_i$ for each $\triangle_i \in T^<$ and $C_i \supseteq \bar{C}_i$ for each $\triangle_i \in T^>$. Consequently, $\bar{\mathcal{U}} \subseteq \mathcal{U}(\mathcal{C})$.

For a cell $\tau \in \bar{\mathcal{F}}^{\nabla}$, set $\mathcal{C}_\tau^< = \{C_i \mid T_i \in T_\tau^<\}$. Clearly, no other triangle $\triangle_i \in T^<$ can have its real expansion C_i meet τ. We define $\mathcal{C}_\tau = \{C_i \cap \tau \mid \triangle_i \in T_\tau^< \cup T^>\}$, and set $\mathcal{U}_\tau = \mathcal{U}(\mathcal{C}_\tau)$. By construction,

$$\mathcal{U} = \mathcal{U}(\mathcal{C}) = \bar{\mathcal{U}} \cup \bigcup_{\tau \in \bar{\mathcal{F}}^{\nabla}} \mathcal{U}_\tau. \tag{10}$$

We call a vertex v of $\mathcal{U}$ *internal* if v does not lie on the boundary of any polytope C_i with $T_i \in T^>$, and we call v *external* otherwise. The number of external vertices is trivially $O(tn^2)$, so it suffices to bound the number of internal vertices of $\mathcal{U}$. Since $\bar{\mathcal{U}} \subseteq \mathcal{U}$, every internal vertex lies in $\bar{\mathcal{F}}$. Suppose v lies in the tetrahedron τ of $\bar{\mathcal{F}}^{\nabla}$. Then by (10), v is a vertex of $\mathcal{U}_\tau$. Furthermore, v is an internal vertex, so it is not incident on any C_i for $T_i \in T^>$, and thus v is a vertex of $\mathcal{U}(\mathcal{C}_\tau^<)$. The total number of internal vertices in τ is bounded by the combinatorial complexity of $\mathcal{U}(\mathcal{C}_\tau^<)$, denoted by $\psi(T_\tau^<, \mathbf{r}_\tau)$ (as defined in the introduction), where $\mathbf{r}_\tau$ is the set of scaling factors of the triangles in T_τ, Hence, the total number of internal vertices on $\mathcal{U}$ is $\sum_{\tau \in \bar{\mathcal{F}}^{\nabla}} \psi(T_\tau^<, \mathbf{r}_\tau)$, and (10) implies that the total number vertices $\psi(T, \mathbf{r})$ on $\mathcal{U}$ satisfies $\psi(T, \mathbf{r}) \leq \sum_{\tau \in \bar{\mathcal{F}}^{\nabla}} \psi(T_\tau^<, \mathbf{r}_\tau) + O(tn^2)$.

Expected number of vertices. We now bound the expected complexity of $\mathcal{U}$, conditioned on a fixed choice of $T^>$. That is, we condition the analysis on the subset of those permutations σ that assign the t largest scaling factors in Θ to the triangles of $T^>$. The set $\Theta_\tau^<$ of scaling factors assigned to the triangles in $T_\tau^<$ is not fixed, but, since $T_\tau^<$ is a fixed set, conditioned only on the choice of $T^>$, the set $\Theta_\tau^<$ is a *random subset* of $\Theta^< = \{\theta_1, \ldots, \theta_{n-t}\}$ of the fixed size n_τ. Moreover, the assignment (under the original random permutation σ) of these scaling factors to the triangles in $T_\tau^<$ is a *random* permutation of $\Theta_\tau^<$. Hence, conditioning further on the choice of $\Theta_\tau^<$, the expected value of $\psi(T_\tau^<, \mathbf{r}_\tau)$ is $\psi(T_\tau^<, \Theta_\tau^<) \leq \psi(n_\tau)$. Hence, the last expression also bounds the unconditional expected complexity of $\mathcal{U}(\mathcal{C}_\tau^-)$, albeit still conditioned on a fixed choice of $T^>$. Summing this over all tetrahedra of $\bar{\mathcal{F}}^{\nabla}$, we obtain that $\psi(T, \Theta \mid T^>) \leq \sum_{\tau \in \bar{\mathcal{F}}^{\nabla}} \psi(n_\tau) + O(tn^2)$.

To bound the unconditional expected value $\psi(T, \Theta)$ we notice that the subset $T^>$ of the triangles that are assigned the t largest scaling factors is a random subset of T. Since $T^>$ is a random sample of T of size t and since, by Lemma 3.2, there are only $O(n^s)$ tetrahedra that can appear in the decomposition $\bar{\mathcal{F}}^{\nabla}$, the following lemma is a consequence of a standard random-sampling argument; see [17, Section 4.6] for a proof.

▶ **Lemma 3.3.** *For any constant $c > 0$, with probability $1 - O\left(\frac{1}{n^c}\right)$, every (open) tetrahedron τ of $\bar{\mathcal{F}}^{\nabla}$ intersects at most $\frac{c'n}{t} \log n$ of the sets of $\bar{\mathcal{C}}^<$, where c' is a constant that depends on both c.*

Choosing $c = 3$ in Lemma 3.3, with probability $1 - O\left(\frac{1}{n^3}\right)$ our $T^>$ is such that $n_\tau \leq \frac{c'n}{t} \log n$ for every $\tau \in \bar{\mathcal{F}}^{\nabla}$. With probability $O\left(\frac{1}{n^3}\right)$, $T^>$ may fail to satisfy this property,

[8] Clearly, $\bar{\mathcal{U}}$ and $\bar{\mathcal{F}}$ are uniquely determined. For $\bar{\mathcal{F}}^{\nabla}$, the statement means that if we follow an agreed-upon implementation of the construction in the proof of Lemma 3.1, $\bar{\mathcal{F}}^{\nabla}$ is also uniquely determined.

but, since the complexity of $\mathcal{U}(\mathcal{C})$ is always $O(n^3)$ (again, with a constant of proportionality that depends on the complexity of B), the contribution of these "bad" choices of $T^>$ to $\psi(T, \Theta)$ is $O(1)$. If n is below some appropriate constant n_0, we can use a trivial bound of $O(n^3)$ for the complexity of $\mathcal{U}$. Altogether, we obtain the following recurrence,

$$\psi(n) \leq \begin{cases} a_0 n^3 & \text{for } n \leq n_0, \\ a_1 t^2 \log t \cdot \psi\left(\frac{c'n}{t} \log n\right) + a_2 n^2 t & \text{for } n > n_0, \end{cases} \tag{11}$$

where n_0, a_0, a_1, a_2, c' are suitable constants. With appropriate choice of parameters, the solution of this recurrence is $\psi(n) \leq An^{2+\varepsilon}$, for any $\varepsilon > 0$, where A depends on ε and on the other constants appearing in the recurrence. This proves Theorem 1.3.

References

1 P. K. Agarwal, A. Efrat, S. K. Ganjugunte, D. Hay, S. Sankararaman, and G. Zussman. The resilience of WDM networks to probabilistic geographical failures. *IEEE/ACM Trans. on Netw.*, 21(5):1525–1538, 2013.

2 P. K. Agarwal, S. Har-Peled, H. Kaplan, and M. Sharir. Union of random minkowski sums and network vulnerability analysis. *Discrete Comput. Geom.*, 52(3):551–582, 2014.

3 P.K. Agarwal and M. Sharir. Pipes, cigars, and kreplach: the union of minkowski sums in three dimensions. *Discrete Comput. Geom.*, 24(4):645–657, Jan 2000.

4 B. Aronov, A. Efrat, V. Koltun, and M. Sharir. On the union of κ-round objects in three and four dimensions. *Discrete Comput. Geom.*, 36(4):511–526, 2006.

5 B. Aronov and M. Sharir. Triangles in space or building (and analyzing) castles in the air. *Combinatorica*, 10(2):137–173, 1990.

6 B. Aronov and M. Sharir. Castles in the air revisited. *Discrete Comput. Geom.*, 12(2):119–150, 1994.

7 B. Aronov and M. Sharir. On translational motion planning of a convex polyhedron in 3-space. *SIAM J. Comput.*, 26(6):1785–1803, 1997.

8 B. Aronov, M. Sharir, and B. Tagansky. The union of convex polyhedra in three dimensions. *SIAM J. Comput.*, 26(6):1670–1688, 1997.

9 F. Aurenhammer, R. Klein, and D. T. Lee. *Voronoi diagrams and Delaunay triangulations.* World Scientific, 2013.

10 J.-D. Boissonnat, M. Sharir, B. Tagansky, and M. Yvinec. Voronoi diagrams in higher dimensions under certain polyhedral distance functions. *Discrete Comput. Geom.*, 19(4):485–519, 1998.

11 H.-C. Chang, S. Har-Peled, and B. Raichel. From proximity to utility: A voronoi partition of pareto optima. *Discrete Comput. Geom.*, 56(3):631–656, 2016.

12 E. Ezra. On the union of cylinders in three dimensions. *Discrete Comput. Geom.*, 45(1):45–64, 2011.

13 E. Ezra and M. Sharir. On the union of fat tetrahedra in three dimensions. *J. ACM*, 57(1):2:1–2:23, 2009.

14 M. J. Golin and H.-S. Na. On the average complexity of 3d-voronoi diagrams of random points on convex polytopes. *Comput. Geom: Theory Appls.*, 25(3):197–231, 2003.

15 R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1994.

16 S. Har-Peled and B. Raichel. On the complexity of randomly weighted multiplicative voronoi diagrams. *Discrete Comput. Geom.*, 53(3):547–568, 2015.

17 J. Matoušek. *Lectures on Discrete Geometry.* Springer-Verlag, Berlin, Heidelberg, 2002.

18 J. Pach P. K. Agarwal and M. Sharir. State of the union (of geometric objects). In J. Pach J. Goodman and R. Pollack, editors, *Surveys on Discrete and Computational Geometry*, pages 9–48. Amer. Math. Soc., Providence, RI, 2008.

19 J. Pach, I. Safruti, and M. Sharir. The union of congruent cubes in three dimensions. *Discrete Comput. Geom.*, 30(1):133–160, 2003.

20 R. Schneider and J. A. Wieacker. Integral geometry. In P. M. Gruber and J. M. Wills, editors, *Handbook of Convex Geometry*, volume B, pages 1349–1390. North-Holland, Amsterdam, 1993.

21 W. Weil and J. A. Wieacker. Stochastic geometry. In P. M. Gruber and J. M. Wills, editors, *Handbook of Convex Geometry*, volume B, pages 1393–1438. North-Holland, Amsterdam, 1993.

Noise-Tolerant Testing of High Entanglement of Formation

Rotem Arnon-Friedman[1]
ETH Zürich, Switzerland
http://rotemaf.info
rotema@itp.phys.ethz.ch

Henry Yuen[2]
UC Berkeley, USA
http://www.henryyuen.net
hyuen@cs.berkeley.edu

Abstract

In this work we construct tests that allow a classical user to certify high dimensional entanglement in uncharacterized and possibly noisy quantum devices. We present a family of non-local games $\{G_n\}$ that for all n certify states with entanglement of formation $\Omega(n)$. These tests can be derived from *any* bipartite non-local game with a classical-quantum gap. Furthermore, our tests are noise-tolerant in the sense that fault tolerant technologies are not needed to play the games; entanglement distributed over noisy channels can pass with high probability, making our tests relevant for realistic experimental settings. This is in contrast to, e.g., results on self-testing of high dimensional entanglement, which are only relevant when the noise rate goes to zero with the system's size n. As a corollary of our result, we supply a lower-bound on the entanglement cost of any state achieving a quantum advantage in a bipartite non-local game. Our proof techniques heavily rely on ideas from the work on classical and quantum parallel repetition theorems.

2012 ACM Subject Classification Theory of computation → Quantum complexity theory

Keywords and phrases device independence, quantum games, entanglement testing, noise tolerance

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.11

Related Version Full version hosted on arXiv at https://arxiv.org/abs/1712.09368.

Acknowledgements We thank Valerio Scarani for helpful pointers to the literature, Thomas Vidick for feedback on an earlier draft, and anonymous referees for helpful comments and pointing us to the work of [JPPG+10]. Work on this project initiated when RAF was visiting UC Berkeley.

1 Introduction

Non-local games offer a powerful method to experimentally study the properties and behavior of uncharacterized quantum systems. In a non-local game, an experimenter can play a game with two non-communicating players (representing spatially separated quantum systems) via classical interaction only. Based on the outcome of the game, the experimenter draws conclusions about, e.g., whether the players used an entangled quantum state to win the

[1] RAF is supported by the Swiss National Science Foundation (grant No. 200020-135048) via the National Centre of Competence in Research "Quantum Science and Technolog" and by the US Air Force Office of Scientific Research (grant No. FA9550-16-1-0245)
[2] HY is supported by ARO Grant W911NF-12-1-0541 and NSF Grant CCF-1410022.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 11; pp. 11:1–11:12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

game. This idea dates back to John Bell's seminal paper [7], in which he presents a game to test the non-classicality of nature. Today, such games are not only relevant for our understanding of the foundations of quantum physics but are at the heart of device-independent quantum information processing, where a classical user can certify that an unknown quantum device is performing a desired computational or cryptographic task (such as, e.g., device-independent quantum key distribution [4, 37, 45, 34, 2] or delegated quantum computation [42, 24, 22, 35, 17]).

In this work we ask the following question:

Is it possible to classically test for high dimensional entanglement, even in the presence of noise?

Whereas Bell's original test is a classical method to certify the *presence* of entanglement, we are instead interested in non-local games that would allow us to quantify the *amount*. In particular, we are interested in certifying the amount of entanglement of *noisy quantum systems*.

Designing noise-tolerant tests for high dimensional entanglement is an important and timely challenge for both computer science and physics. First, our understanding of complexity theory indicates that unless $\mathsf{BQP} \subseteq \mathsf{BPP}$ (i.e., quantum computers are classically simulable), general quantum computations must involve highly entangled states. Thus if we hope to achieve super-classical speedups in quantum computers, at the very least we must be able to generate high dimensional entanglement.

Second, we are seeing increasingly sophisticated experiments involving quantum information, from loophole-free Bell tests [26, 43, 23] to small scale quantum computers [10, 29]. However, full-fledged quantum fault tolerance appears to be a faraway prospect; in the near-term, our explorations of complex quantum states will be done using noisy gates and little (if any) error correction. Despite this obstacle, researchers have been enthusiastically proposing uses of noisy quantum computers, from approximate optimization to investigation of exotic physics phenomena. Interesting questions will emerge in tandem with these efforts, namely: how can one verify that a noisy quantum computer has succeeded in these proposed experiments? Finding noise-tolerant tests to certify high dimensional entanglement is a prerequisite step towards verifying other complex quantum behavior in this noisy regime.

1.1 What do we mean by certifying entanglement?

There are a variety of ways to formulate this task; our work is most directly motivated by recent work on *self-tests*, which are games that certify the presence of entanglement *of a specific form*. The works of [33, 14, 16, 20, 35] construct families of games $\{G_n\}$ where any optimal quantum strategy for G_n must use a large amount of entanglement, e.g., a tensor product of n EPR pairs. These self-testing results are also *robust*, in that near-optimal strategies must use states that are near a specific highly entangled state. However, these tests will also reject a natural class of highly entangled states such as $\sigma^{\otimes n}$ where σ has fidelity $1-\nu$ with a single EPR pair. Here, think of ν as a small (but fixed) noise parameter that represents the level of imperfection of a state preparation process.

Thus, even though $|EPR\rangle\langle EPR|^{\otimes n}$ can be used to pass the tests of [33, 14, 16, 20, 35] with high probability, the "similar-looking" state $\sigma^{\otimes n}$ will fail with high probability. A key observation we wish to emphasize in this paper is that robustness of a self-test is *not* equivalent to noise tolerance!

More formally, the robust self-tests in the above works show the following: let $\mathrm{qval}(G_n)$ denote the optimal quantum winning probability for the game G_n. Then there exists a

function $f(n,\varepsilon)$ and an *ideal state* ρ_n^* such that for all ε, any quantum strategy that achieves a winning probability of at least $\text{qval}(G_n)-\varepsilon$ must use a state ρ that is $f(n,\varepsilon)$-close to ρ_n^*. In these works, ρ_n^* is a state whose entanglement grows with n (like a maximally entangled state on n qubits). "Closeness" can be defined in terms of the fidelity of the two states up to local isometries acting on each of the players' systems.

Given a game G_n as above, an experiment to test the entanglement of an unknown state ρ can be the following: play the game G_n using ρ, and check whether the game is won.[3] In order to obtain a non-trivial guarantee about ρ, we require that $f(n,\varepsilon)<1$; one can think of this function as specifying the amount of experimental imperfection/noise that can be tolerated by the *test* itself. In the works of [33, 14, 16, 20], the function $f(n,\varepsilon)$ scales as $a\cdot n^b\cdot\varepsilon^c$ for constants a,b,c. Thus we get no guarantees about ρ unless ε scales as $1/\operatorname{poly}(n)$. In other words, as we increase the amount of entanglement we want to certify, the test becomes less tolerant of noise!

The strongest self-testing result (in this context) is presented in the work of Natarajan and Vidick [35]. There, a self-test for n EPR pairs is given where the associated function is $f(n,\varepsilon)=O(\sqrt{\varepsilon})$. While the closeness parameter is independent of the parameter n, such $f(n,\varepsilon)$ still requires that, in order to pass the test with high probability, the players share a state ρ that is *globally* $O(\sqrt{\varepsilon})$-close to $|EPR\rangle\langle EPR|^{\otimes n}$. Using a state like $\sigma^{\otimes n}$ where σ has $1-\nu$ fidelity with a single EPR pair would fail their test with high probability, because $\sigma^{\otimes n}$ has *exponentially small* fidelity $(1-\nu)^n\approx e^{-n/\nu}$ with $|EPR\rangle\langle EPR|^{\otimes n}$.

In this paper we seek an entanglement test that is both *sound* — meaning that any strategy that passes the test with good probability must have high entanglement — and also *noise tolerant*, meaning that they do not reject noisy implementations of an ideal strategy. The self-tests above are sound, but they are not noise tolerant. Part of the difficulty stems from the fact that it is not even clear how one should *formulate* the soundness guarantee of a desired noise-tolerant self-testing result.

1.2 Noise model

As discussed above, we wish to define a testing procedure that can also certify entanglement in noisy entangled states. While our work can be used to certify different types of noisy states, we briefly discuss a specific noise model here for the sake of concreteness. The noise model that we have in mind produces a state of the form $\sigma^{\otimes n}$ where each σ has fidelity $1-\nu$ with some optimal state defined via the considered non-local game. Such a state can be produced, e.g., by sending many copies of the optimal state via noisy channels.

We emphasize that by saying that this is the noise model that we consider we merely mean that we require that our tests will be able to certify the entanglement of $\sigma^{\otimes n}$. However, we do not assume that all of the states on which the procedure is applied must have this form (i.e., the soundness part of the statement is independent of the considered noise model).

1.3 Results and contributions

In this work, instead of trying to certify the presence of a specific state like in self-testing statements, we address the question of certifying an entanglement measure. This allows us to sidestep the difficulty of formulating a noise-tolerant self-testing result.

[3] In an experiment one actually needs to prepare many identical and independent copies of ρ and play the game G_n many times. Then the average winning probability can be calculated, and high amount of entanglement is certified (with high probability) if the average winning probability is at least $\text{qval}(G_n)-\varepsilon$.

We present a family of simple non-local games $\{G_n\}$ where each game G_n certifies that the shared state of the players has $\Omega(n)$ bits of *entanglement of formation*. The entanglement of formation, denoted by $E_F(\rho)$, is a well-studied entanglement measure for bipartite mixed states that, in the case of pure states, is equal to the entanglement entropy. As the name suggests, the entanglement of formation captures, roughly speaking, the amount of entanglement needed in order to produce a given state ρ. It is also closely related to another important, perhaps more well known, entanglement measure which will be of use below – the *entanglement cost* $E_C(\rho)$. The entanglement cost of a mixed state roughly describes how many EPR pairs are needed to create ρ via local operations and classical communication [9]. We provide a more thorough discussion of the entanglement measures relevant for our work in Section 1.4.

The family of non-local games that we consider are the so called *threshold games*. Before stating our main result, we define these games. Let G be a two-player non-local game with classical value[4] $\mathrm{cval}(G)$ and quantum value $\mathrm{qval}(G)$. Given an integer $n \geq 1$ and a noise threshold $0 \leq \nu < \mathrm{qval}(G) - \mathrm{cval}(G)$, define the threshold game $G^n_{\mathrm{qval}(G)-\nu}$ to be a game where the two-players now play n independent instances of G in parallel, and win if they win at least $\mathrm{qval}(G) - \nu$ fraction of instances of G.

The main theorem of this paper is as follows:

▶ **Theorem 1** (Main theorem). *Let G be a two-player game with a classical-quantum gap: i.e., $\Delta := \mathrm{qval}(G) - \mathrm{cval}(G) > 0$. Let $0 \leq \nu < \Delta$ be a noise parameter.*

Completeness (Noise tolerance). *Let $n \geq 1$ be an integer. Consider a quantum strategy for G that succeeds with probability $\mathrm{qval}(G) - \eta$ for $0 \leq \eta < \nu$. Playing this strategy n times independently in parallel in the threshold game $G^n_{\mathrm{qval}(G)-\nu}$ succeeds with probability at least $1 - \exp(-(\nu - \eta)^2 n/3)$.*[5]

Soundness (Entanglement certification). *There exist constants $0 < c_1, c_2 < 1$ such that for sufficiently large $n > \frac{1}{c_1}$, any strategy that wins the threshold game $G^n_{\mathrm{qval}(G)-\nu}$ with probability $\kappa \geq \exp(-c_1 n)$ must use a quantum state ρ such that its entanglement of formation satisfies $E_F(\rho) \geq c_2 \kappa^2 n$.*

The constants c_1, c_2 depend only on Δ, ν, and the number of possible answers in G.

To gain a better understanding of our theorem we now give an example. Consider the famous CHSH game, which has classical value $\mathrm{cval}(CHSH) = 3/4$ and quantum value $\mathrm{qval}(CHSH) \approx 0.854$. Any strategy for winning a single instance of CHSH with probability $\mathrm{qval}(CHSH) - \eta$ for some parameter $0 \leq \eta < 0.1$ must use some entangled state σ. An "honest" strategy for playing the threshold game $CHSH^n_{.854-2\eta}$ would be to play each instance of $CHSH$ independently using $\sigma^{\otimes n}$ as the entangled resource state. Via a simple Chernoff-Hoeffding bound it is easy to see that this strategy will pass $CHSH^n_{.854-2\eta}$ with overwhelming probability. Thus this game is noise-tolerant. The entanglement of formation of $\sigma^{\otimes n}$ is indeed $\Omega(n)$.

But what about other strategies? Is there a state with entanglement of formation $o(n)$ that can be used to win $CHSH^n_{.854-2\eta}$ sufficiently well? Theorem 1 shows that this is not possible.

---

[4] The classical value of a game is the maximum winning probability when the players employ classical strategies, i.e., do not use entanglement. Similarly, the quantum value of a game is the optimal winning probability when using quantum strategies.

[5] Alternatively, a simpler (but slightly weaker) statement is that playing a strategy the succeeds with probability $\mathrm{qval}(G) - \nu$ in G n times independently in parallel succeeds in the threshold game $G^n_{\mathrm{qval}(G)-\nu}$ with probability $\frac{1}{2}$. This is sufficient for an experiment certifying entanglement.

We list several features of Theorem 1:

1. It holds for *any* two-player game G. In other words, any game with a classical-quantum gap can be "lifted" to another game that tests for large entanglement in a noise-tolerant manner.
2. The players are able to pass our test with high probability by holding a tensor product of noisy few-qubit states (such as $\sigma^{\otimes n}$ where σ has fidelity $1-\nu$ with an EPR pair for any amount). The theorem gives non-trivial guarantees for any $0 \leq \nu < \mathrm{qval}(G) - \mathrm{cval}(G)$, i.e., it is robust to *any* amount of noise up to the classical limit.
3. It gives non-trivial guarantees even for strategies whose success probability is far from optimal; for any constant κ, Theorem 1 still guarantees that $E_F(\rho) \in \Omega(n)$.[6]

Theorem 1 thus shows that by playing the simple threshold game $G^n_{\mathrm{qval}(G)-\nu}$ with an uncharacterized device we can classically test for large amounts of entanglement (as measured by the entanglement of formation), even when the device is highly noisy, as current devices are. As far as we are aware, previous results [33, 14, 16, 35, 18] cannot be used to derive conclusions which are quantitively strong as Theorem 1, even when considering more complex games and proof techniques.[7]

Our main theorem presented above can be easily used to derive another quantitive relation between the advantage in a non-local game G and the *entanglement cost* required to achieve this advantage. Specifically, we prove the following.

▶ **Theorem 2.** *Let G be a two-player game with a classical-quantum gap: i.e., $\Delta := \mathrm{qval}(G) - \mathrm{cval}(G) > 0$. Let $0 \leq \nu < \Delta$ be a noise parameter. Then, for any state σ that can be used to win G with probability at least $\mathrm{qval}(G) - \nu$, its entanglement cost satisfies $E_C(\sigma) \geq c_2/4$, where c_2 is the constant from Theorem 1.*

Put in other words: the minimum entanglement cost[8] needed to obtain a super-classical success probability in a non-local game only depends on the classical-quantum gap as well as the number of possible answers in the game.

As we explain in Section 1.4, even given the full description of a state σ, calculating $E_C(\sigma)$ is not easy and no "single letter" formula is known to describe it. Theorem 2 gives a simple lower bound on $E_C(\sigma)$ in terms of σ's advantage in any non-local game G.

The only lower-bound with a similar flavour which was known before is the one given in [46]. There, a (tight) relation between $E_F(\sigma)$ and σ's winning probability in the CHSH game was derived. Self-testing results can, of course, also be used to achieve similar bounds (by taking into account the continuity of the considered entanglement measures), but so far most of the results are non-trivial for a very limited amount of noise and only apply to specific two-player games. In contrast, Theorem 2 holds for any non-local game and amount of noise.

1.4 Why entanglement of formation?

In this section we motivate and explain the relations between the entanglement measures certified by our tests in Theorem 1 and Theorem 2.

[6] However, the constants c_1 and c_2 are probably not optimal and can be improved.

[7] This is not to say that our work supersedes the mentioned works; these derived self-testing statements which certify the *state* and not just its *entanglement* as we do here.

[8] For any σ, $E_F(\sigma) \geq E_C(\sigma)$. Thus, Theorem 2 could have been phrased in terms of the entanglement of formation as well.

Myriad entanglement measurements have been studied by researchers, each possessing various properties [38, 28]. For pure bipartite states $|\psi\rangle^{\mathsf{AB}}$, the coarsest quantity describing entanglement is the *entanglement rank*, which is simply the Schmidt rank of $|\psi\rangle$. However, this is not a very useful measure of entanglement as one can have a state arbitrarily close to a product state, yet have high entanglement rank.

A more natural measure of entanglement is the *entanglement entropy* $E(\psi)$, which is the von Neumann entropy of the reduced density matrix of $|\psi\rangle$ on system A or equivalently B [8, 39]. In fact, the entanglement entropy is the *unique* entanglement measure for pure bipartite states that satisfies a few natural axioms, such as monotonicity under local operations and classical communication (LOCC) and asymptotic continuity [28].

For mixed states the situation is more complicated — there is no clear "best" entanglement measure. The most natural and operational entanglement measures are considered to be the *entanglement cost* E_C and the *distillable entanglement* E_D. In fact, for any entanglement measure M satisfying some natural properties we have that $E_D \leq M \leq E_C$ [28]. Thus the entanglement cost and distillable entanglement are in a sense "extremal" entanglement measures. For pure states, both E_C and E_D are equal to the entanglement entropy.

In the following we focus on E_C. Informally, the entanglement cost of a bipartite quantum state ρ_{AB} describes the number of maximally entangled states required to produce ρ using only LOCC. As LOCC cannot increase entanglement, the pre-shared maximally entangled states describe the sole source of entanglement in such a process and hence quantify how entangled ρ is in a meaningful way.[9]

Formally, the entanglement cost is defined as the following asymptotic quantity:

$$E_C(\rho) = \inf\left\{ r : \lim_{n\to\infty} \left(\inf_{\Lambda} \|\rho^{\otimes n} - \Lambda(\Phi^+_{2^{rn}})\|_1 \right) = 0 \right\} ,$$

where the infimum ranges over all LOCC maps Λ and $\Phi^+_{2^{rn}}$ is the maximally entangled state of rank 2^{rn}. That is, it is the maximal possible rate r at which one can convert $\Phi^+_{2^{rn}}$ into $\rho^{\otimes n}$ with vanishing error in the limit $n \to \infty$.

Computing $E_C(\rho)$ is considered to be a difficult task in general. Due to this reason one usually considers a closely related entanglement measure called the *entanglement of formation*. It is formally defined as follows [9]:

$$E_F(\rho) = \inf\left\{ \sum_i p_i E(\Psi_i) : \rho = \sum_i p_i |\Psi_i\rangle\langle\Psi_i| \right\} .$$

That is, $E_F(\rho)$ is the minimum average entanglement entropy over all pure-state decompositions of ρ.

The entanglement of formation derives its relevance from its relation to the entanglement cost $E_C(\rho)$ discussed above. It describes the rate in which maximally entangled states are converted to ρ using a specific type of LOCC protocols [51] (whereas $E_C(\rho)$ is the minimum over all LOCC protocols). Furthermore, [25] showed that the entanglement cost is equal to the *regularised* entanglement of formation:

$$E_C(\rho) = E_F^{\infty}(\rho) = \lim_{n\to\infty} (E_F(\rho^{\otimes n})/n).$$

[9] Another way of thinking about the operational meaning of entanglement cost is by considering the task of entanglement dilution. There, the goal is to start with initial noiseless entanglement and dilute it to create a target state ρ using LOCC.

For some time it was conjectured that the entanglement of formation is additive and hence $E_C(\rho) = E_F(\rho)$. Today it is known that this is not the case and that the limit in the above equation is needed in general [11].

It is not known how to compute $E_F^\infty(\rho)$ for general ρ, in part because of the infinite limit. The "single-letter" quantity $E_F(\rho)$ does not appear to be much easier to compute because of the minimisation over all possible decompositions of ρ. To date, it can be done only for states with high symmetry [44, 48] or of low dimension [49, 50, 3]. One can imagine that the task of calculating or bounding $E_F(\rho)$ only becomes harder if one does not have full information about ρ as in the scenario considered in the current work.

In the light of the above, one can see our work as giving a way to lower bound those complex entanglement measures for an unknown state ρ in a device-independent manner. Of course, this is not a general method that works for all states ρ, but rather it works for any state ρ that can be used to gain an advantage in non-local games (or, in other words, violate some Bell inequality). Specifically, Theorem 1 gives a lower bound on E_F for high dimensional (while perhaps noisy) states that can be used to pass the threshold game $G^n_{\mathrm{qval}(G)-\nu}$ for some two-player game G. Theorem 2 gives a lower bound on E_C for any state achieving a quantum advantage in a two-player game G. In particular, for any given state one can choose the game G such that the lower bounds on E_F and E_C are maximal.

1.5 Proof technique

The proof idea is simple: if the entanglement of formation of the players' shared state in the threshold game $G^n_{\mathrm{qval}(G)-\nu}$ is $o(n)$ and the players win with non-negligible probability, then this strategy can be transformed into a strategy for the original game G that uses *no* entanglement, yet still wins with probability strictly greater than $\mathrm{cval}(G)$, which would be a contradiction.

This is argued as follows. Consider a two-player game G where the first player receives a question x and produces answer a, and the second player receives question y and responds with answer b. The players win if $V(x, y, a, b) = 1$ for some predicate V. Let $\mathrm{qval}(G) > \mathrm{cval}(G)$.

Now suppose there is a quantum strategy that wins $G^n_{\mathrm{qval}(G)-\nu}$ with decent probability. A simple probabilistic argument implies that conditioned on an event E of winning roughly $\mathrm{qval}(G) - \nu$ fraction of some subset $S \subseteq [n]$ of instances, the players will win the j'th instance with probability close to $\mathrm{qval}(G)$, for an average $j \in [n]$. Another way of phrasing this statement is: Let $(\mathbf{X}_j, \mathbf{Y}_j)$ denote the questions to the two players in the j'th instance of G, and let $(\mathbf{A}_j, \mathbf{B}_j)$ denote their answers. Let $\mathsf{P}_{\mathbf{X}_j\mathbf{Y}_j\mathbf{A}_j\mathbf{B}_j|E}$ denote the joint distribution of questions and answers of the j'th coordinate in this hypothetical strategy, conditioned on the event E. Then sampling a tuple $(\mathbf{X}_j, \mathbf{Y}_j, \mathbf{A}_j, \mathbf{B}_j)$ from $\mathsf{P}_{\mathbf{X}_j\mathbf{Y}_j\mathbf{A}_j\mathbf{B}_j|E}$ will satisfy the game predicate V with probability $\mathrm{qval}(G) - \varepsilon > \mathrm{cval}(G)$.

Next, we will prove the following three statements (roughly speaking): (1) $\mathsf{P}_{\mathbf{X}_j\mathbf{Y}_j|E} \approx \mathsf{P}_{\mathbf{X}_j\mathbf{Y}_j}$, (2) $\mathsf{P}_{\mathbf{A}_j|\mathbf{X}_j\mathbf{Y}_jE} \approx \mathsf{P}_{\mathbf{A}_j|\mathbf{X}_jE}$, and (3) $\mathsf{P}_{\mathbf{B}_j|\mathbf{X}_j\mathbf{Y}_j\mathbf{A}_jE} \approx \mathsf{P}_{\mathbf{B}_j|\mathbf{Y}_jE}$, where "$\approx$" denotes closeness in statistical distance. Notice that without the conditioning event E, the first item would be trivial and the second item would follow exactly from the non-signaling condition between the players. To prove the third item, we use the fact that the hypothetical strategy for the threshold game uses $o(n)$ bits of entanglement; intuitively this implies that each instance of G can only use $o(1)$ bits of entanglement.

Putting these three items together, we obtain a classical strategy for G: the first player receives question $\mathbf{X}_j$, and samples an answer $\mathbf{A}_j$ from the distribution $\mathsf{P}_{\mathbf{A}_j|\mathbf{X}_jE}$. The second player receives question $\mathbf{Y}_j$ and samples from $\mathsf{P}_{\mathbf{B}_j|\mathbf{Y}_jE}$. The joint distribution of their questions and answers will be close to $\mathsf{P}_{\mathbf{X}_j\mathbf{Y}_j\mathbf{A}_j\mathbf{B}_j|E}$, but that implies that they will win G with probability $\mathrm{qval}(G) - \varepsilon > \mathrm{cval}(G)$, which is a contradiction.

The proof strategy and the techniques used are heavily inspired by the proofs of the *parallel repetition theorem* in classical complexity theory [41, 27, 40], and subsequently the work on the *quantum parallel repetition problem*. This problem asks for a bound on $\mathrm{qval}(G^n)$ if $\mathrm{qval}(G) < 1$, where G^n is like the threshold game except we demand that the players win *all* instances of G. It is conjectured that $\mathrm{qval}(G^n)$ decays exponentially with n, although the best general upper bound is that $\mathrm{qval}(G^n)$ decays polynomially with n when $\mathrm{qval}(G) < 1$ [52]. Nearly all of the works that study the quantum parallel repetition problem [30, 15, 5, 6] share the proof strategy of transforming a "too-good-to-be-true" strategy for the repeated game G^n into a "too-good-to-be-true" strategy for the single game G, namely a quantum strategy with success probability better than $\mathrm{qval}(G)$, a contradiction. These works all use information-theoretic machinery in the proof, and in this work we use the same tools.

The full proof can be found at `https://arxiv.org/abs/1712.09368`.

1.6 Related work

Our work is the first that addresses directly the question of certifying the entanglement of formation of high dimensional states in a noise-tolerant way (while the case of a single CHSH game was already considered in [46] as mentioned above).

Any robust self-testing result can be used to certify any continuous entanglement measures (e.g. the entanglement of formation); but as explained before, such results cannot accommodate the kinds of noise considered here. In addition to the self-testing results mentioned before [33, 14, 16, 20, 35, 18], the only other self-testing result that certifies asymptotically growing amounts of entanglement is from the work of Reichardt, Unger and Vazirani [42], who show how to verify quantum computations using classical resources only. At the heart of their result is a *sequential* protocol where the experimenter plays many rounds of the CHSH game with the two players in order to certify the presence of many EPR pairs. However, like the other self-testing results, the protocol of [42] is also not noise-tolerant in the sense considered here.

If one cares just about certifying high *entanglement rank* of a state (rather than certifying an entanglement measure such as E_F, or precisely characterizing the state as in self-testing), then we can combine the following two independent results to address the question of noise-tolerant, device-independent testing of asymptotically growing amounts of entanglement: The work of [40] shows that the *classical* value of a threshold game $G^n_{\mathrm{cval}(G)+\delta}$ decays exponentially fast with n (if $\mathrm{cval}(G) < 1$). The work of [31] shows that the maximum quantum success probability in a game F using dimension-d entanglement is at most $d\,\mathrm{cval}(F)$. Letting F be a threshold game, we obtain that d must be exponentially large in any quantum strategy whose winning probability is say at least a small constant. Since the threshold game is noise-tolerant (i.e. it can be won with high probability with noisy strategies), this gives a noise-tolerant test for *entanglement rank*. This same argument can be modified to show that the 1/2-*Rényi entropy* of the state[10] must be linear in n.

Our test lower bounds a stronger entanglement measure, the entanglement of formation, which in the pure state case is the entanglement entropy and therefore a lower bound on the 1/2-Rényi entropy. There can be arbitrarily large gaps between the von Neumann entropy and the 1/2-Rényi entropy of a pure state.

[10] The 1/2-Rényi entropy of a pure state $|\psi\rangle$ is $2\log(\sum_i \lambda_i^{1/2})$ where λ_i are the eigenvalues of the reduced density matrix of $|\psi\rangle$ on either side.

The broader goal of certifying the dimension of a quantum system in a device-independent manner has been heavily studied under the heading of *dimension witnesses*. Much of the work on dimension witnesses has focused on finding Bell inequalities such that achieving the optimal violation requires an entangled state of a certain dimension [12, 36, 13]. Many of these works construct and design dimension witnesses using a combination of analytical and numerical techniques.

1.7 Future work

Some open problems and future directions include:

1. Quantitatively improve our results. The constants c_1, c_2 in Theorem 1 are small; for the CHSH game, the constant c_1 is on the order of 10^{-6} and thus in order for our Theorem to give any guarantees, $\sim 10^6$ CHSH games would have to be played. Even though recent experiments are capable of producing such a large amount of states (in [32], for example, order of 10^{10} signals were produced), an improvement of the constants can lead to the ability of certifying *much* more entanglement in such experiments. Our analysis is likely far from tight and significant quantitative improvements can probably be gained by tailoring the analysis to a specific game, such as the CHSH game.
2. To get a non-trivial bound on the entanglement of formation, this requires that the success probability κ is at least $\sim 1/\sqrt{n}$. Can this dependence on κ be improved?
3. Can one prove a version of Theorem 1 for some non-local games G that allows one to lower bound other measures of entanglement, such as distillable entanglement[11] or quantum conditional entropy? The results of [47, 21] indicate that this cannot be done for arbitrary amount of noise for all games since there are Bell inequalities that can be violated while using states with un-distillable entanglement or positive conditional entropy.
4. Can one prove a self-testing result for a growing number of EPR pairs that is also noise-tolerant in the sense described above? A concrete goal would be to characterize all near-optimal strategies for the threshold game $CHSH^n_{.854-\nu}$. The results of [19] hint that by sticking to the current measures of distance considered in self-testing results any characterization of near-optimal strategies for $CHSH^n_{.854-\nu}$, in the regime of high amount of noise, must include also non-entangled states. Hence, we do not expect self-testing results (as they are phrased today) to allow for certification of entanglement in the presence of arbitrary noise using threshold games.

References

1 Rotem Arnon-Friedman and Jean-Daniel Bancal. Device-independent certification of one-shot distillable entanglement. *arXiv*, 2017.

2 Rotem Arnon-Friedman, Renato Renner, and Thomas Vidick. Simple and tight device-independent security proofs. *arXiv preprint arXiv:1607.01797*, 2016.

3 Koenraad Audenaert, Frank Verstraete, and Bart De Moor. Variational characterizations of separability and entanglement of formation. *Physical Review A*, 64(5):052304, 2001.

4 Jonathan Barrett, Lucien Hardy, and Adrian Kent. No signaling and quantum key distribution. *Physical Review Letters*, 95(1):010503, 2005.

[11] In a related work by Jean-Daniel Bancal together with one of the current authors a device-independent protocol certifying a lower bound on the one shot distillable entanglement is given. The considered setting and type of statement are different than the ones presented here. For further details see [1].

5 Mohammad Bavarian, Thomas Vidick, and Henry Yuen. Parallel repetition via fortification: analytic view and the quantum case. *arXiv preprint arXiv:1603.05349*, 2016.

6 Mohammad Bavarian, Thomas Vidick, and Henry Yuen. Hardness amplification for entangled games via anchoring. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 303–316, 2017.

7 John S Bell. On the Einstein-Podolsky-Rosen paradox. *Physics*, 1(3), 1964.

8 Charles H Bennett, Herbert J Bernstein, Sandu Popescu, and Benjamin Schumacher. Concentrating partial entanglement by local operations. *Physical Review A*, 53(4):2046, 1996.

9 Charles H Bennett, David P DiVincenzo, John A Smolin, and William K Wootters. Mixed-state entanglement and quantum error correction. *Physical Review A*, 54(5):3824, 1996.

10 Sergio Boixo, Sergei V Isakov, Vadim N Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, John M Martinis, and Hartmut Neven. Characterizing quantum supremacy in near-term devices. *arXiv preprint arXiv:1608.00263*, 2016.

11 Fernando GSL Brandao and Michał Horodecki. On Hastings' counterexamples to the minimum output entropy additivity conjecture. *Open Systems & Information Dynamics*, 17(01):31–52, 2010.

12 Nicolas Brunner, Stefano Pironio, Antonio Acin, Nicolas Gisin, André Allan Méthot, and Valerio Scarani. Testing the dimension of Hilbert spaces. *Physical review letters*, 100(21):210503, 2008.

13 Yu Cai, Jean-Daniel Bancal, Jacquiline Romero, and Valerio Scarani. A new device-independent dimension witness and its experimental implementation. *Journal of Physics A: Mathematical and Theoretical*, 49(30):305301, 2016.

14 Rui Chao, Ben W Reichardt, Chris Sutherland, and Thomas Vidick. Test for a large amount of entanglement, using few measurements. *arXiv preprint arXiv:1610.00771*, 2016.

15 Kai-Min Chung, Xiaodi Wu, and Henry Yuen. Parallel repetition for entangled k-player games via fast quantum search. In *the 30th Conference on Computational Complexity (CCC)*, pages 512–536, 2015.

16 Andrea Coladangelo. Parallel self-testing of (tilted) EPR pairs via copies of (tilted) CHSH and the magic square game. *Quantum Information and Computation*, 17(9-10):831–865, 2017.

17 Andrea Coladangelo, Alex Grilo, Stacey Jeffery, and Thomas Vidick. Verifier-on-a-leash: new schemes for verifiable delegated quantum computation, with quasilinear resources. *arXiv preprint arXiv:1708.07359*, 2017.

18 Andrea Coladangelo and Jalex Stark. Robust self-testing for linear constraint system games. *arXiv preprint arXiv:1709.09267*, 2017.

19 Tim Coopmans. Robust self-testing of (almost) all pure two-qubit states. Master's thesis, Universiteit van Amsterdam, 2017.

20 Matthew Coudron and Anand Natarajan. The parallel-repeated magic square game is rigid. *arXiv preprint arXiv:1609.06306*, 2016.

21 Nicolai Friis, Sridhar Bulusu, and Reinhold A Bertlmann. Geometry of two-qubit states with negative conditional entropy. *Journal of Physics A: Mathematical and Theoretical*, 50(12):125301, 2017.

22 Alexandru Gheorghiu, Elham Kashefi, and Petros Wallden. Robustness and device independence of verifiable blind quantum computing. *New Journal of Physics*, 17(8):083040, 2015.

23 Marissa Giustina, Marijn AM Versteegh, Sören Wengerowsky, Johannes Handsteiner, Armin Hochrainer, Kevin Phelan, Fabian Steinlechner, Johannes Kofler, Jan-Åke Larsson, Carlos Abellán, et al. Significant-loophole-free test of Bell's theorem with entangled photons. *Physical review letters*, 115(25):250401, 2015.

24 Michal Hajdušek, Carlos A Pérez-Delgado, and Joseph F Fitzsimons. Device-independent verifiable blind quantum computation. *arXiv preprint arXiv:1502.02563*, 2015.

25 Patrick M Hayden, Michal Horodecki, and Barbara M Terhal. The asymptotic entanglement cost of preparing a quantum state. *Journal of Physics A: Mathematical and General*, 34(35):6891, 2001.

26 Bas Hensen, H Bernien, AE Dréau, A Reiserer, N Kalb, MS Blok, J Ruitenberg, RFL Vermeulen, RN Schouten, C Abellán, et al. Loophole-free Bell inequality violation using electron spins separated by 1.3 kilometres. *Nature*, 526(7575):682–686, 2015.

27 Thomas Holenstein. Parallel repetition: Simplification and the no-signaling case. *Theory of Computing*, 5(8):141–172, 2009. doi:10.4086/toc.2009.v005a008.

28 Ryszard Horodecki, Paweł Horodecki, Michał Horodecki, and Karol Horodecki. Quantum entanglement. *Reviews of modern physics*, 81(2):865, 2009.

29 IBM. IBM quantum experience. URL: https://www.research.ibm.com/ibm-q/.

30 Rahul Jain, Attila Pereszlényi, and Penghui Yao. A parallel repetition theorem for entangled two-player one-round games under product distributions. In *Proceedings of Conference on Computational Complexity (CCC)*, pages 209–216, 2014.

31 Marius Junge, Carlos Palazuelos, David Pérez-García, Ignacio Villanueva, and Michael M Wolf. Unbounded violations of bipartite Bell inequalities via operator space theory. *Communications in Mathematical Physics*, 300(3):715–739, 2010.

32 Yang Liu, Xiao Yuan, Ming-Han Li, Weijun Zhang, Qi Zhao, Jiaqiang Zhong, Yuan Cao, Yu-Huai Li, Luo-Kan Chen, Hao Li, et al. High speed self-testing quantum random number generation without detection loophole. In *Frontiers in Optics*, pages FTh2E–1. Optical Society of America, 2017.

33 Matthew McKague. Self-testing in parallel. *New Journal of Physics*, 18(4):045013, 2016.

34 Carl A Miller and Yaoyun Shi. Robust protocols for securely expanding randomness and distributing keys using untrusted quantum devices. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 417–426. ACM, 2014.

35 Anand Natarajan and Thomas Vidick. A quantum linearity test for robustly verifying entanglement. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1003–1015. ACM, 2017.

36 Károly F Pál and Tamás Vértesi. Quantum bounds on Bell inequalities. *Physical Review A*, 79(2):022120, 2009.

37 Stefano Pironio, Antonio Acín, Nicolas Brunner, Nicolas Gisin, Serge Massar, and Valerio Scarani. Device-independent quantum key distribution secure against collective attacks. *New Journal of Physics*, 11(4):045021, 2009.

38 Martin B Plenio and Shashank Virmani. An introduction to entanglement measures. *arXiv preprint quant-ph/0504163*, 2005.

39 Sandu Popescu and Daniel Rohrlich. Thermodynamics and the measure of entanglement. *Physical Review A*, 56(5):R3319, 1997.

40 Anup Rao. Parallel repetition in projection games and a concentration bound. *SIAM Journal on Computing*, 40(6):1871–1891, 2011.

41 Ran Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998.

42 Ben W Reichardt, Falk Unger, and Umesh Vazirani. Classical command of quantum systems. *Nature*, 496(7446):456–460, 2013.

43 Lynden K Shalm, Evan Meyer-Scott, Bradley G Christensen, Peter Bierhorst, Michael A Wayne, Martin J Stevens, Thomas Gerrits, Scott Glancy, Deny R Hamel, Michael S Allman, et al. Strong loophole-free test of local realism. *Physical review letters*, 115(25):250402, 2015.

44 Barbara M Terhal and Karl Gerd H Vollbrecht. Entanglement of formation for isotropic states. *Physical Review Letters*, 85(12):2625, 2000.

45 Umesh Vazirani and Thomas Vidick. Fully device-independent quantum key distribution. *Physical review letters*, 113(14):140501, 2014.

46 Frank Verstraete and Michael M Wolf. Entanglement versus Bell violations and their behavior under local filtering operations. *Physical review letters*, 89(17):170401, 2002.

47 Tamás Vértesi and Nicolas Brunner. Disproving the Peres conjecture: Bell nonlocality from bipartite bound entanglement. *arXiv preprint arXiv:1405.4502*, 2014.

48 Karl Gerd H Vollbrecht and Reinhard F Werner. Entanglement measures under symmetry. *Physical Review A*, 64(6):062307, 2001.

49 William K Wootters. Entanglement of formation of an arbitrary state of two qubits. *Physical Review Letters*, 80(10):2245, 1998.

50 William K Wootters. Entanglement of formation and concurrence. *Quantum Information & Computation*, 1(1):27–44, 2001.

51 William K. Wootters. Entanglement of formation. In Prem Kumar, Giacomo M D'Ariano, and Osamu Hirota, editors, *Quantum Communication, Computing and Measurement 2*, pages 69–74. Springer, 2002.

52 Henry Yuen. A parallel repetition theorem for all entangled games. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 77:1–77:13, 2016.

A Complete Dichotomy for Complex-Valued Holant^c

Miriam Backens[1]
Department of Computer Science, University of Oxford, UK
miriam.backens@cs.ox.ac.uk

Abstract

Holant problems are a family of counting problems on graphs, parametrised by sets of complex-valued functions of Boolean inputs. Holant^c denotes a subfamily of those problems, where any function set considered must contain the two unary functions pinning inputs to values 0 or 1. The complexity classification of Holant problems usually takes the form of dichotomy theorems, showing that for any set of functions in the family, the problem is either #P-hard or it can be solved in polynomial time. Previous such results include a dichotomy for real-valued Holant^c and one for Holant^c with complex symmetric functions, i.e. functions which only depend on the Hamming weight of the input.

Here, we derive a dichotomy theorem for Holant^c with complex-valued, not necessarily symmetric functions. The tractable cases are the complex-valued generalisations of the tractable cases of the real-valued Holant^c dichotomy. The proof uses results from quantum information theory, particularly about entanglement. This full dichotomy for Holant^c answers a question that has been open for almost a decade.

2012 ACM Subject Classification Theory of computation → Complexity classes, Theory of computation → Problems, reductions and completeness

Keywords and phrases computational complexity, counting complexity, Holant problems, dichotomy, entanglement

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.12

Related Version A full version of the paper is available at [1], arXiv:1704.05798.

Acknowledgements I would like to thank Pinyan Lu for pointing out a flaw in the original statement of the main theorem. Many thanks also to Ashley Montanaro, William Whistler and Leslie Ann Goldberg for helpful comments on earlier versions of this paper.

1 Introduction

Holant problems are a framework for the analysis of counting problems defined on graphs. They encompass and generalise other counting complexity frameworks like counting constraint satisfaction problems (#CSP) [9, 10] and counting graph homomorphisms [10, 4].

A Holant instance is defined by assigning a function from a specified set to each vertex of a graph, with the edges incident on that vertex corresponding to inputs of the function. The counting problem is a sum-of-products computation: multiplying all the function values

[1] The research leading to these results has received funding from EPSRC via grant EP/L021005/1 and from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) ERC grant agreement no. 334828. The paper reflects only the author's views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein. No new data were created during this study. The majority of this research was done at the School of Mathematics, University of Bristol, UK.

and then summing over the different assignments of input values to the edges [10]. A more rigorous definition can be found in Section 2. In this work, we consider only complex-valued functions of Boolean inputs. Throughout, all numbers are assumed to be algebraic [4].

Problems expressible in the Holant framework include counting matchings or counting perfect matchings, counting vertex covers [10], and counting Eulerian orientations [14]. A Holant problem can also be thought of as the problem of contracting a tensor network; from that perspective, each function corresponds to a tensor with one index for each input [7].

The main goal in the analysis of Holant problems is the derivation of dichotomy theorems, showing that all problems in a certain family are either polynomial time solvable or #P-hard. Families of Holant problems are often defined by assuming that the function sets contain specific functions, which are said to be 'freely available'. As an example, the problem #CSP($\mathcal{F}$) for a function set $\mathcal{F}$ effectively corresponds to the Holant problem $\textsc{Holant}\left(\mathcal{F} \cup \{=_n |\ n \in \mathbb{N}_{\geq 1}\}\right)$, where $(=_1) : \{0,1\} \to \mathbb{C}$ is the function that is 1 on both inputs, and, for $n \geq 2$, $(=_n) : \{0,1\}^n \to \mathbb{C}$ is the function satisfying:

$$(=_n)(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{if } x_1 = x_2 = \ldots = x_n \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The problem $\textsc{Holant}^c\left(\mathcal{F}\right)$ is the Holant problem where the unary functions pinning edges to values 0 or 1 are available in addition to the elements of $\mathcal{F}$:

$$\textsc{Holant}^c\left(\mathcal{F}\right) = \textsc{Holant}\left(\mathcal{F} \cup \{\delta_0, \delta_1\}\right), \tag{2}$$

with $\delta_0(0) = 1$, $\delta_0(1) = 0$ and $\delta_1(0) = 0$, $\delta_1(1) = 1$ [9]. Another important family is $\textsc{Holant}^*$, in which all unary functions are freely available [6].

Known Holant dichotomies include a full dichotomy for $\textsc{Holant}^*$ [6], dichotomies for $\textsc{Holant}^c$ with symmetric functions, i.e. where all functions in the sets considered depend only on the Hamming weight of the input [9], and a dichotomy for real-valued $\textsc{Holant}^c$, where functions need not be symmetric but must take values in $\mathbb{R}$ instead of $\mathbb{C}$ [11]. There is also a dichotomy for symmetric $\textsc{Holant}$ [5] and a dichotomy for non-negative real-valued $\textsc{Holant}$ [18]. Both existing results about $\textsc{Holant}^c$ are proved via dichotomies for #CSP problems with complex-valued, not necessarily symmetric functions: in the first case, a dichotomy for general #CSP problems, and in the second case, a dichotomy for $\#\mathrm{CSP}_2^c$, a subfamily of #CSP in which each variable must appear an even number of times and variables can be pinned to 0 or 1.

While many dichotomies have been derived for functions taking values in some smaller set, we consider complex-valued functions to be the natural setting for Holant problems. This is motivated in part by connecting Holant problems to quantum computation, where complex numbers naturally arise: the problem of strongly classically simulating a quantum circuit with fixed input and output states can immediately be expressed as a Holant problem. The second justification for considering complex numbers is that many tractable sets find a more natural expression over $\mathbb{C}$. An example of this are the 'affine functions' (see Section 3.1): they were originally discovered as several distinct tractable sets for a smaller codomain, but their definition is vastly more straightforward when expressed in terms of complex values [10]. Thirdly, some problems parametrised in terms of real values are naturally connected by *complex-valued* holographic transformations (defined in Section 2.1): for example, the problem of counting Eulerian orientations on 4-regular graphs is expressed in the Holant framework in this way [5, 14].

We therefore build on the existing work to derive a $\textsc{Holant}^c$ dichotomy for complex-valued, not necessarily symmetric functions. In the process, we employ notation and results from quantum information theory. This approach was first used in a recent paper [2]

to derive a dichotomy for Holant^+, in which four unary functions are freely available, including the ones available in Holant^c: $\text{Holant}^+(\mathcal{F}) = \text{Holant}(\mathcal{F} \cup \{\delta_0, \delta_1, \delta_+, \delta_-\})$, where $\delta_+(x) = 1$ for both inputs (i.e. it is the same as the unary equality function[2]) and $\delta_-(x) = (-1)^x$.

A core part of quantum theory, and also of the quantum approach to Holant problems, is the notion of entanglement. A pure[3] quantum state of n qubits, the quantum equivalents of bits, is represented by a vector in the space $(\mathbb{C}^2)^{\otimes n}$, which consists of n tensor copies of $\mathbb{C}^2$. Such a vector is called *entangled* if it cannot be written as a tensor product of vectors from each copy of $\mathbb{C}^2$.

An n-ary function $f : \{0,1\}^n \to \mathbb{C}$ can be considered as a vector in $\mathbb{C}^{2^n}$ by treating each input as an element of an orthonormal basis for that space and using the function values as coefficients in a linear combination of those basis vectors (cf. Section 2). This vector space $\mathbb{C}^{2^n}$ is isomorphic to $(\mathbb{C}^2)^{\otimes n}$, allowing functions to be brought into correspondence with quantum states. We thus call a function entangled if the associated vector is entangled. Identifying this property in Holant problems lets us apply some of the large body of work on quantum entanglement [19, 13, 12, 17] to Holant problems. The resulting complexity classification of Holant problems remains non-quantum, we simply employ a different set of mathematical tools in their analysis.

In the Holant^+ dichotomy, it was shown how to construct a gadget for an entangled ternary function, given an n-ary entangled function with $n \geq 3$ and using the freely-available unary functions. Furthermore, in most cases it was shown to be possible to realise a ternary symmetric function from this [2]. We show how to adapt those constructions to the Holant^c framework, where only two unary functions are freely available. This does not always work, yet if the construction fails, it is always the case that either the problem is tractable by the Holant^* dichotomy or it is equivalent to $\#\text{CSP}_2^c$ using techniques from [11]. With these adaptations, we therefore extend the dichotomy theorem for real-valued Holant^c to arbitrary complex-valued functions.

In the following, Section 2 contains the formal definition of Holant problems and an overview over common strategies used in classifying their complexity. We recap existing results in Section 3. The new dichotomy and its constituent lemmas are proved in Section 4. Section 5 contains the conclusions and outlook.

2 Holant problems

Holant problems are a framework for counting complexity problems defined on graphs, first introduced in the conference version of [10]. Let $G = (V, E)$ be a graph with vertices V and edges E, which may contain self-loops and multiple edges between the same pair of vertices, and let $\mathcal{F}$ be a set of complex-valued functions of Boolean inputs. Throughout, when we refer to complex numbers we mean algebraic complex numbers. Let π be a function that assigns to each degree-n vertex v in the graph an n-ary function $f_v \in \mathcal{F}$ and also assigns one edge incident on the vertex to each input of the function. This determines a complex value

[2] The availability of the function δ_+ is indeed important to the Holant^+ dichotomy proof. This is a difference between the Holant framework and #CSP, where a constraint equal to δ_+ would have no effect.

[3] There are also *mixed* quantum states, which have a different mathematical representation, and which are not considered here.

associated with the tuple $(\mathcal{F}, G, \pi)$, called the *Holant* and defined as follows:

$$\text{Holant}_{(\mathcal{F},G,\pi)} = \sum_{\sigma:E\to\{0,1\}} \prod_{v\in V} f\left(\sigma|_{E(v)}\right). \tag{3}$$

Here, σ is an assignment of a Boolean value to each edge in the graph and $\sigma|_{E(v)}$ is the restriction of σ to the edges incident on vertex v. The tuple $(\mathcal{F}, G, \pi)$ is called a *signature grid*.

The associated counting problem is Holant $(\mathcal{F})$: given a signature grid $(\mathcal{F}, G, \pi)$ for the fixed set of functions $\mathcal{F}$, find $\text{Holant}_{(\mathcal{F},G,\pi)}$.

It is often useful to think of the functions, also called signatures, as vectors or tensors [7]. The n-ary functions can be put in one-to-one correspondence with vectors in $\mathbb{C}^{2^n}$ as follows: pick an orthonormal basis for $\mathbb{C}^{2^n}$ and label its elements $\{|x\rangle\}_{x\in\{0,1\}^n}$, i.e. each basis vector is labelled by one of the 2^n n-bit strings.[4] Then assign to each $f : \{0,1\}^n \to \mathbb{C}$ the vector $|f\rangle = \sum_{x\in\{0,1\}^n} f(x)\,|x\rangle$. Conversely, any vector $|\psi\rangle \in \mathbb{C}^{2^n}$ corresponds to an n-ary function $\psi : \{0,1\}^n \to \mathbb{C} :: x \mapsto \langle x|\psi\rangle$, where $\langle\cdot|\cdot\rangle$ denotes the inner product of two vectors.[5] The product of two functions of disjoint sets of variables corresponds to the tensor product of the associated vectors, i.e. if $h(x_1,\ldots,x_n,y_1,\ldots,y_m) = f(x_1,\ldots,x_n)g(y_1,\ldots,y_m)$ then $|h\rangle = |f\rangle \otimes |g\rangle$. Where no confusion arises, we drop the tensor product symbol and sometimes even combine labels into a single 'ket' $|\cdot\rangle$: for example, instead of writing $|0\rangle \otimes |0\rangle$, we may write $|0\rangle\,|0\rangle$ or $|00\rangle$. If g is a unary signature, we sometimes write $\langle g|_l\,|f\rangle$ to indicate that the l-th input of f is connected to a vertex with signature g.

The vector perspective is particularly useful for bipartite Holant problems, which arise on bipartite graphs if we assign functions from two different signature sets to the vertices in the two different partitions. Then the Holant becomes the inner product between two vectors corresponding to the two partitions. Formally: let $G = (V, W, E)$ be a bipartite graph with vertex partitions V and W, and let $\mathcal{F}, \mathcal{G}$ be two sets of signatures. Suppose π is a function that assigns elements of $\mathcal{F}$ to vertices from V and elements of $\mathcal{G}$ to vertices from W and otherwise acts as described above. Then:

$$\text{Holant}_{(\mathcal{F}|\mathcal{G},G,\pi)} = \left(\bigotimes_{v\in V}(|f_v\rangle)^T\right)\left(\bigotimes_{w\in W}|g_w\rangle\right), \tag{4}$$

where we assume the two tensor products are arranged so that the appropriate components of the two vectors meet. The bipartite Holant problem over signature sets $\mathcal{F}$ and $\mathcal{G}$ is denoted by Holant $(\mathcal{F} \mid \mathcal{G})$.

Any Holant instance can be made bipartite without changing the value of the Holant by inserting an additional vertex in the middle of each edge and assigning it the binary equality signature $=_2$. Thus, Holant $(\mathcal{F}) \equiv_T$ Holant $(\mathcal{F} \mid \{=_2\})$, i.e. the two problems have the same complexity.

In the following, we use the function and vector perspectives on signatures interchangeably.

2.1 Complexity classification

Most complexity results about the Holant problem take the form of *dichotomies*, showing that for all signature sets in a specific family, the problem is either #P-hard or in FP. Such a dichotomy is not expected to be true for all counting complexity problems: if FP $\neq$ #P, then there are problems in #P $\setminus$ FP which are not #P-hard [6].

[4] The $|\cdot\rangle$ notation for vectors is the Dirac or bra-ket notation commonly used in quantum theory.

[5] Strictly speaking, this notation refers to the complex inner product, i.e. $\langle x|$ is the conjugate transpose of $|x\rangle$, but the distinction is irrelevant if all coefficients of $|x\rangle$ are real.

We write $A \leq_T B$ if there exists a polynomial-time reduction from problem B to problem A and $A \equiv_T B$ if $(A \leq_T B) \wedge (B \leq_T A)$. A number of polynomial-time reduction techniques are commonly used in Holant problems.

The technique of *holographic reductions* is the origin of the name Holant. Let M be a 2 by 2 invertible complex matrix and define $M \circ f = M^{\otimes \operatorname{arity}(f)} |f\rangle$, where $M^{\otimes 1} = M$ and $M^{\otimes n+1} = M \otimes M^{\otimes n}$. Furthermore, let $M \circ \mathcal{F} = \{M \circ f \mid f \in \mathcal{F}\}$. This is called a *holographic transformation*. Let $\mathcal{F}$ and $\mathcal{G}$ be two signature sets. Then:

$$\textsc{Holant}\left(\mathcal{F} \mid \mathcal{G}\right) \equiv_T \textsc{Holant}\left(M \circ \mathcal{F} \mid (M^{-1})^T \circ \mathcal{G}\right) \tag{5}$$

and, in fact, $\text{Holant}_{(\mathcal{F}|\mathcal{G},G,\pi)} = \text{Holant}_{(M\circ\mathcal{F}|(M^{-1})^T\circ\mathcal{G},G,\pi')}$; this is Valiant's Holant Theorem [20].

A second technique is that of *gadgets*. Consider a subgraph of some signature grid, which is connected to the larger graph by n edges. This subgraph can be replaced by a single degree-n vertex with an appropriate signature without changing the value of the overall Holant. Thus, if there exists some subgraph with signatures taken from $\mathcal{F}$ such that the effective signature for that subgraph is g, then [6]:

$$\textsc{Holant}\left(\mathcal{F} \cup \{g\}\right) \leq_T \textsc{Holant}\left(\mathcal{F}\right). \tag{6}$$

We say g is *realisable* over $\mathcal{F}$. As multiplying a signature by a non-zero constant does not change the complexity of a Holant problem, we also consider g realisable if we can construct a gadget with effective signature cg for some $c \in \mathbb{C} \setminus \{0\}$. In bipartite signature grids, we may distinguish between left-hand side (LHS) gadgets and right-hand side (RHS) gadgets, which can be used as if they are signatures for the left and right partitions, respectively.

Finally, there is the technique of *polynomial interpolation*. Let $\mathcal{F}$ be a set of signatures and suppose g is a signature that cannot be realised over $\mathcal{F}$. If, given any signature grid over $\mathcal{F} \cup \{g\}$, it is possible to set up a family of signature grids over $\mathcal{F}$ such that the Holant for the original problem instance can be determined efficiently from the Holant values of the family by solving a linear system, then g is said to be *interpolatable* over $\mathcal{F}$. We do not directly use polynomial interpolation here, though the technique is employed by many of the results we build upon. A rigorous definition of polynomial interpolation can be found in [10].

2.2 Properties of signatures

A signature is called *symmetric* if its value as a function depends only on the Hamming weight of the inputs – in other words, it is invariant under any permutation of the inputs. Symmetric functions are often written in the short-hand notation $f = [f_0, f_1, \ldots, f_n]$, where f_k is the value f takes on inputs of Hamming weight k.

A signature is *degenerate* if it can be written as a tensor product of unary signatures. Conversely, using language from quantum theory, a signature is *entangled* if it cannot be written as a tensor product of unary signatures. This corresponds to the notion of *non-degenerate* signatures in the Holant literature. For example, $|01\rangle + |11\rangle$ is not entangled because it can be rewritten as $(|0\rangle + |1\rangle) \otimes |1\rangle$. On the other hand, the binary equality signature $|00\rangle + |11\rangle$ is entangled. If $k \geq 2$, a k-ary signature can be partially decomposable into a tensor product, e.g. $|0\rangle \otimes (|00\rangle + |11\rangle)$. We say a signature is *genuinely entangled* if there is no way of decomposing it as a tensor product of signatures of any arity. A genuinely entangled signature of arity at least 3 is said to be *multipartite entangled* (as opposed to the bipartite entanglement in a signature of arity 2). A non-genuinely entangled signature has multipartite entanglement if it has a tensor factor corresponding to a genuinely entangled signature of arity at least 3 and a set of signatures has multipartite entanglement if it contains a signature that does.

Among genuinely entangled ternary signatures, we distinguish two types, also known as 'entanglement classes' [12]. Each entanglement class contains signatures that are related via local holographic transformations, i.e. two states $|f\rangle, |g\rangle$ are in the same entanglement class if and only if there exist some 2 by 2 invertible matrices A, B, C such that $(A \otimes B \otimes C)|f\rangle = |g\rangle$.

In quantum theory, the two entanglement classes are named after their representative states: the ternary equality signature $|000\rangle + |111\rangle$, called the GHZ-state, and the ternary perfect matching signature $|001\rangle + |010\rangle + |100\rangle$, called the W state. We say that a signature has GHZ type if it is equivalent to the GHZ state under local holographic transformations and that a signature has W type if it is equivalent to the GHZ state under local holographic transformations. In the Holant literature, GHZ-type signatures are called the *generic case* and W type signatures are called the *double-root case* [9].

The two types of ternary genuinely entangled signatures can be distinguished as follows [17]. Let f be a ternary signature and write:

$$|f\rangle = \sum_{k,\ell,m \in \{0,1\}} a_{k\ell m} |k\ell m\rangle, \tag{7}$$

where $a_{k\ell m} \in \mathbb{C}$ for all $k, \ell, m \in \{0, 1\}$. Then $|f\rangle$ has GHZ type if the following polynomial in the coefficients is non-zero:

$$(a_{000}a_{111} - a_{010}a_{101} + a_{001}a_{110} - a_{011}a_{100})^2 - 4(a_{010}a_{100} - a_{000}a_{110})(a_{011}a_{101} - a_{001}a_{111}). \tag{8}$$

The signature $|f\rangle$ has W type if the above polynomial is zero, and furthermore each of the following three expressions is satisfied:

$$(a_{000}a_{011} \neq a_{001}a_{010}) \vee (a_{101}a_{110} \neq a_{100}a_{111}) \tag{9}$$

$$(a_{001}a_{100} \neq a_{000}a_{101}) \vee (a_{011}a_{110} \neq a_{010}a_{111}) \tag{10}$$

$$(a_{011}a_{101} \neq a_{001}a_{111}) \vee (a_{010}a_{100} \neq a_{000}a_{110}). \tag{11}$$

If the polynomial (8) is zero and at least one of the above expressions evaluates to false, then the signature is not genuinely entangled.

There are many other classes of entangled signatures for higher arities [21, 15, 16, 3], but those are not directly relevant to this paper.

Given a set of signatures that contains multipartite entanglement in the Holant^c framework, we can assume without loss of generality that we have a genuinely multipartite-entangled signature. To see this, consider a non-zero signature $|\psi\rangle$ that has multipartite entanglement, and suppose $|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$. Then at least one of the tensor factors must have multipartite entanglement, assume this is $|\psi_1\rangle$. Now, $|\psi\rangle$ is non-zero, so $\langle x|\psi_2\rangle$ must be non-zero for some bit string x. Thus we can realise $|\psi_1\rangle$ by connecting all inputs associated with $|\psi_2\rangle$ to $|0\rangle$ or $|1\rangle$, as appropriate.

3 Existing results

It is difficult to determine the complexity of the general Holant problem. Thus, all existing dichotomies make use of one or more simplifying assumptions: either they assume the availability of certain signatures in all signature sets considered [7, 6, 2], or they only consider signature sets containing functions taken from more restricted families, e.g. symmetric functions [9, 5] or functions taking only real [11] or even non-negative real values [18].

Among others, the following variants of the Holant problem have been considered:

- Holant* $(\mathcal{F})$ = Holant $(\mathcal{F} \cup \mathcal{U})$, where $\mathcal{U}$ is the set of all unary signatures [6],

- $\text{Holant}^+(\mathcal{F}) = \text{Holant}(\mathcal{F} \cup \{\delta_0, \delta_1, \delta_+, \delta_-\})$, where $\delta_+(x) = 1$ and $\delta_-(x) = (-1)^x$ [2], and
- $\text{Holant}^c(\mathcal{F}) = \text{Holant}(\mathcal{F} \cup \{\delta_0, \delta_1\})$ [9, 11].

Several variants of complex-weighted Boolean counting constraint satisfaction problems (#CSP) have also been expressed in the Holant framework. These include:

- $\#\text{CSP}(\mathcal{F}) = \text{Holant}(\mathcal{F} \mid \mathcal{G})$, where $\mathcal{G} = \{=_n \mid n \in \mathbb{N}_{\geq 1}\}$ is the set containing all equality signatures [10], and
- $\#\text{CSP}_2^c(\mathcal{F}) = \text{Holant}(\mathcal{F} \mid \{\delta_0, \delta_1\} \cup \{=_{2n} \mid n \in \mathbb{N}_{\geq 1}\})$ [11].

The $\#\text{CSP}_2^c$ problems assume availability of the signatures pinning inputs to 0 or 1, respectively, as well as equality signatures of even arity.

Existing results include full dichotomies for Holant^* [6], Holant^+ [2], #CSP [10], and $\#\text{CSP}_2^c$ [11]. There are also dichotomies for Holant^c with symmetric complex-valued signatures [6], Holant^c with arbitrary real-valued signatures [11], Holant with symmetric complex-valued signatures [5], and Holant with arbitrary non-negative real-valued signatures [18].

3.1 Preliminary definitions

The following definitions will be used throughout the dichotomy theorems. Write:

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}, \qquad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \text{and} \qquad K = \begin{pmatrix} 1 & 1 \\ i & -i \end{pmatrix}, \tag{12}$$

where $i^2 = -1$. Then let:

- $\mathcal{T}$ be the set of all unary and binary signatures,
- $\mathcal{E}$ be the set of all signatures that are non-zero only on two inputs x and $\bar{x}$, where $\bar{x}$ denotes the bit-wise complement of x, also called *generalised equality signatures*,
- $\mathcal{M}$ be the set of all signatures that are non-zero only on inputs of Hamming weight at most 1,
- $\mathcal{A}$ be the set of all *affine signatures*, i.e. functions of the form $f(x) = ci^{l(x)}(-1)^{q(x)}\chi$, where $c \in \mathbb{C}$, $l(x)$ is a linear Boolean function, $q(x)$ is a quadratic Boolean function, and χ is the indicator function for an affine space, and
- $\mathcal{L}$ be the set of all signatures f with the property that, for any bit string x in the support of f:

$$\left(\bigotimes_{j=1}^{\text{arity}(f)} T^{x_j} \right) |f\rangle \in \mathcal{A}, \tag{13}$$

where x_j is the j-th bit of x. Elements of $\mathcal{L}$ are called *local affine signatures*.

Denote by $\langle \mathcal{F} \rangle$ the closure of the signature set $\mathcal{F}$ under tensor products. It is straightforward to see that $\mathcal{A} = \langle \mathcal{A} \rangle$ and $\mathcal{L} = \langle \mathcal{L} \rangle$, i.e. these signature sets are already closed under tensor products. If n is a positive integer, we denote by $[n]$ the set $\{1, 2, \ldots, n\}$.

3.2 Dichotomies for Holant variants

The Holant dichotomies generally build upon each other. Dichotomies with fewer freely-available signatures refer to dichotomies for problems with more freely-available signatures, as all tractable cases of the latter must also be tractable cases of the former: removing signatures can never make the problem harder.

▶ **Theorem 1** (Theorem 2.2, [6]). *Let $\mathcal{F}$ be any set of complex valued functions in Boolean variables. The problem* $\text{Holant}^*(\mathcal{F})$ *is polynomial time computable if:*

- $\mathcal{F} \subseteq \langle \mathcal{T} \rangle$, *or*
- $\mathcal{F} \subseteq \langle O \circ \mathcal{E} \rangle$, *where O is a complex orthogonal 2 by 2 matrix, or*
- $\mathcal{F} \subseteq \langle K \circ \mathcal{E} \rangle$, *or*
- $\mathcal{F} \subseteq \langle K \circ \mathcal{M} \rangle$ *or* $\mathcal{F} \subseteq \langle KX \circ \mathcal{M} \rangle$.

In all other cases, $\text{Holant}^*(\mathcal{F})$ *is #P-hard.*

▶ **Theorem 2** (Theorem 6, [9]). *Let $\mathcal{F}$ be a set of complex symmetric signatures.* $\text{Holant}^c(\mathcal{F})$ *is #P-hard unless $\mathcal{F}$ satisfies one of the following conditions, in which case it is tractable:*

- $\text{Holant}^*(\mathcal{F})$ *is tractable, or*
- *there exists a 2 by 2 matrix $S \in \mathcal{S}$ such that $\mathcal{F} \subseteq S \circ \mathcal{A}$, where:*

$$\mathcal{S} = \left\{ S \,\middle|\, (S^T)^{\otimes 2}(=_2), S^T\delta_0, S^T\delta_1 \in \mathcal{A} \right\}. \tag{14}$$

▶ **Theorem 3** (Theorem 4.1, [11]). *A $\#\text{CSP}_2^c(\mathcal{F})$ problem has a polynomial time algorithm if one of the following holds: $\mathcal{F} \subseteq \langle \mathcal{E} \rangle$, $\mathcal{F} \subseteq \mathcal{A}$, $\mathcal{F} \subseteq T \circ \mathcal{A}$, or $\mathcal{F} \subseteq \mathcal{L}$. Otherwise, it is #P-hard.*

The preceding results all apply to complex-valued signatures, but the following theorem is restricted to real-valued ones.

▶ **Theorem 4** (Theorem 5.1, [11]). *Let $\mathcal{F}$ be a set of real-valued signatures. Then* $\text{Holant}^c(\mathcal{F})$ *is #P-hard unless $\mathcal{F}$ is a tractable family for* Holant^* *or* $\#\text{CSP}_2^c$.

3.3 Complexity results for ternary signatures

In addition to the above-mentioned Holant dichotomies, there are also some dichotomies specific to symmetric signatures on three-regular graphs. For signature sets containing a ternary GHZ-type signature, there is furthermore a direct relationship to #CSP, which allows a more general complexity classification. When deriving the Holant^c dichotomy, our general approach will be to attempt to construct a gadget for a genuinely entangled ternary signature and then use the following results.

▶ **Theorem 5** (Theorem 3.4, [9]). $\text{Holant}([y_0, y_1, y_2] | [x_0, x_1, x_2, x_3])$ *is #P-hard unless the signatures $[x_0, x_1, x_2, x_3]$ and $[y_0, y_1, y_2]$ satisfy one of the following conditions, in which case the problem is in FP:*

- $[x_0, x_1, x_2, x_3]$ *is degenerate, or*
- *there is a 2 by 2 matrix M such that:*
  - $[x_0, x_1, x_2, x_3] = M \circ [1, 0, 0, 1]$ *and* $(M^T)^{-1} \circ [y_0, y_1, y_2]$ *is in* $\mathcal{A} \cup \langle \mathcal{E} \rangle$,
  - $[x_0, x_1, x_2, x_3] = M \circ [1, 1, 0, 0]$ *and* $(M^T)^{-1} \circ [y_0, y_1, y_2]$ *is of the form* $[0, *, *]$,
  - $[x_0, x_1, x_2, x_3] = M \circ [0, 0, 1, 1]$ *and* $(M^T)^{-1} \circ [y_0, y_1, y_2]$ *is of the form* $[*, *, 0]$,

with $*$ denoting an arbitrary complex number.

The signature $|000\rangle + |111\rangle$ is invariant under holographic transformations of the form $\left(\begin{smallmatrix} 1 & 0 \\ 0 & \omega \end{smallmatrix}\right)$, where $\omega^3 = 1$. Therefore, a binary signature is considered to be ω-normalised if $y_0 = 0$, or there does not exist a primitive $(3t)$-th root of unity λ, where $gcd(t, 3) = 1$, such that $y_2 = \lambda y_0$. Similarly, a unary signature $[a, b]$ is ω-normalised if $a = 0$, or there does not exist a primitive $(3t)$-th root of unity λ, where $gcd(t, 3) = 1$, such that $b = \lambda a$ [9].

▶ **Theorem 6** (Theorem 4.1, [9]). *Let $\mathcal{G}_1, \mathcal{G}_2$ be two sets of signatures and let $[y_0, y_1, y_2]$ be a ω-normalised and non-degenerate signature. In the case of $y_0 = y_2 = 0$, further assume that $\mathcal{G}_1$ contains a unary signature $[a, b]$ which is ω-normalised and satisfies $ab \neq 0$. Then:*

$$\text{Holant}\left(\{[y_0, y_1, y_2]\} \cup \mathcal{G}_1 \mid \{[1, 0, 0, 1]\} \cup \mathcal{G}_2\right) \equiv_T \#\text{CSP}(\{[y_0, y_1, y_2]\} \cup \mathcal{G}_1 \cup \mathcal{G}_2). \quad (15)$$

More specifically, $\text{Holant}\left(\{[y_0, y_1, y_2]\} \cup \mathcal{G}_1 \mid \{[1, 0, 0, 1]\} \cup \mathcal{G}_2\right)$ *is #P-hard unless*

$$\{[y_0, y_1, y_2]\} \cup \mathcal{G}_1 \cup \mathcal{G}_2 \subseteq \langle \mathcal{E} \rangle \quad \text{or} \quad \{[y_0, y_1, y_2]\} \cup \mathcal{G}_1 \cup \mathcal{G}_2 \subseteq \mathcal{A}, \quad (16)$$

in which cases the problem is in FP.

The following lemmas show how to realise symmetric genuinely entangled ternary signatures from non-symmetric ones. They do not rely on any unary signatures.

▶ **Lemma 7** (Lemma 18, [2]). *Let $|\psi\rangle$ be a ternary GHZ-type signature, i.e. $|\psi\rangle = (A \otimes B \otimes C)\,|\text{GHZ}\rangle$ for some invertible 2 by 2 matrices A, B, C. Then at least one of the three possible symmetric triangle gadgets constructed from three copies of $|\psi\rangle$ is non-degenerate, unless $|\psi\rangle \in K \circ \mathcal{E}$ and is furthermore already symmetric.*

▶ **Lemma 8** (Lemma 19, [2]). *Let $|\psi\rangle$ be a ternary W-type signature, i.e. $|\psi\rangle = (A \otimes B \otimes C)\,|W\rangle$ for some invertible 2 by 2 matrices A, B, C. If $|\psi\rangle \in K \circ \mathcal{M}$ (or $|\psi\rangle \in KX \circ \mathcal{M}$), assume that we also have a binary entangled signature $|\phi\rangle$ that is not in $K \circ \mathcal{M}$ (or $KX \circ \mathcal{M}$, respectively). Then we can realise a symmetric genuinely entangled ternary signature.*

3.4 Results about 4-ary signatures

Besides the above results about ternary signatures, we will also make use of the following result about realising or interpolating the 4-ary equality signature from a more general 4-ary signature.

▶ **Lemma 9** (Lemma 2.38, [8]). *Suppose $\mathcal{F}$ contains a signature f of arity 4 with:*

$$|f\rangle = a\,|0000\rangle + b\,|0011\rangle + c\,|1100\rangle + d\,|1111\rangle\,, \quad (17)$$

where $M = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ has full rank. Then $\text{Pl-Holant}(\{=_4\} \cup \mathcal{F}) \leq_T \text{Pl-Holant}(\mathcal{F})$.

Here, Pl-Holant refers to the Holant problem for planar graphs; the lemma can also be used in the non-planar setting.

▶ **Lemma 10** (Lemma 5.2, [11]). *Suppose $\mathcal{F}$ contains a 4-ary generalised equality signature f, i.e. $f \in \mathcal{F} \cap \mathcal{E}$ and $arity(f) = 4$. Then* $\text{Holant}\,(\mathcal{F}) \equiv_T \#\text{CSP}_2(\mathcal{F})$*, the counting constraint satisfaction problem in which each variable appears an even number of times.*

4 The dichotomy

Our dichotomy proof uses techniques from the Holant$^+$ dichotomy [2] and from the real-valued Holant^c dichotomy [11], as well as some new results.

The core strategy in the hardness part of the Holant$^+$ dichotomy proof is to realise a symmetric genuinely entangled ternary signature f and a symmetric entangled binary signature g for which $\text{Holant}\,(\{f\} \mid \{g\})$ is known to be #P-hard. The techniques for realising low-arity signatures utilise knowledge from quantum information theory. They rely crucially on having access to the four unary signatures $\delta_0, \delta_1, \delta_+$ and δ_-, and do not seem directly adaptable to the Holant^c setting.

The dichotomy proof for real-valued Holant^c contains arity-reduction techniques that require only δ_0, δ_1, and self-loops. Yet there are two main barriers to extending this result to complex-valued signatures: firstly, some of the hardness results for genuinely entangled ternary signatures in [11] only apply to real values. Secondly, some cases of the dichotomy proof rely on being able to interpolate all unary signatures using techniques that have only been shown to work for real-valued signatures.

In the present work, our strategy is similar to that in the Holant$^+$ dichotomy: we attempt to realise symmetric genuinely entangled ternary signatures. There are now two cases in which this is not possible: either there is no multipartite entanglement, in which case the problem is in FP, or all genuinely entangled signatures in the closure of $\mathcal{F} \cup \{\delta_0, \delta_1\}$ under gadgets have even arity, in which case the smallest signature that can give hardness has arity 4. For arity reduction, we adapt techniques from [11], modifying them to work for complex values. If all genuinely entangled signatures have even arity, we show analogously to [11] that is is always possible to realise a 4-ary signature of a specific form, which can then be used to realise or interpolate $=_4$. Furthermore, we adapt symmetrisation techniques for genuinely entangled ternary signatures from [2] to work in the Holant^c setting. Thus, we never need to interpolate arbitrary unary signatures. In one subcase, we do require unary signatures other than δ_0 and δ_1, but we give a new construction for realising sufficiently many such signatures by gadgets.

In this extended abstract, we give sketch proofs of the new results; full proofs may be found in the full version of the paper [1].

4.1 Hardness proofs involving a genuinely entangled ternary signature

First, we prove two lemmas that give a complexity classification for Holant^c problems in the presence of any genuinely entangled ternary signature with complex coefficients.

▶ **Lemma 11.** *Let $f \in \mathcal{F}$ be a genuinely entangled ternary signature. Then* Holant$^c(\mathcal{F})$ *is #P-hard unless:*

- Holant$^*(\mathcal{F})$ *is tractable, or*
- $\mathcal{F} \subseteq S \circ \mathcal{A}$ *for some $S \in \mathcal{S}$, as defined in* (14).

In both of those cases, the problem Holant$^c(\mathcal{F})$ *is tractable.*

Proof (sketch). Let $\mathcal{F}' = \mathcal{F} \cup \{\delta_0, \delta_1\}$. We distinguish cases according to whether f is symmetric or not, and according to its entanglement class.

If f is symmetric and has GHZ type, there exists an invertible holographic transformation M that maps f to $=_3$. Transform the Holant problem to bipartite form by adding an extra vertex carrying the signature $=_2$ in the middle of each edge. It is straightforward to see that allowing the signatures δ_0 and δ_1 on both partitions does not affect the complexity, i.e.:

$$\text{Holant}^c(\mathcal{F}) \equiv_T \text{Holant}(\{=_2\} \mid \mathcal{F}') \equiv_T \text{Holant}(\{=_2, \delta_0, \delta_1\} \mid \mathcal{F}'). \tag{18}$$

Apply Valiant's Holant theorem with the holographic transformation M identified above:

$$\text{Holant}(\{=_2, \delta_0, \delta_1\} \mid \mathcal{F}') \equiv_T \text{Holant}\left((M^{-1})^T \circ \{=_2, \delta_0, \delta_1\} \mid M \circ \mathcal{F}'\right). \tag{19}$$

As $(=_3) \in M \circ \mathcal{F}'$ by construction, the problem now has the same form as the LHS of (15), with $[y_0, y_1, y_2] = (M^{-1})^T \circ (=_2)$, $\mathcal{G}_1 = (M^{-1})^T \circ \{\delta_0, \delta_1\}$, and $\mathcal{G}_2 = M \circ \mathcal{F}'$. With a bit of effort, it can be shown that the conditions of Theorem 6 regarding ω-normalisation and unary signatures are always satisfiable by choosing M appropriately; hence:

$$\text{Holant}^c(\mathcal{F}) \equiv_T \#\text{CSP}\left((M^{-1})^T \circ \{=_2, \delta_0, \delta_1\} \cup M \circ \mathcal{F}'\right). \tag{20}$$

Now, as stated in Theorem 6, this problem is #P-hard unless $(M^{-1})^T \circ \{=_2, \delta_0, \delta_1\} \cup M \circ \mathcal{F}'$ is a subset of $\langle \mathcal{E} \rangle$ or a subset of $\mathcal{A}$. With some additional work, these conditions can be shown to correspond to $\text{Holant}^c(\mathcal{F})$ being tractable if $\mathcal{F} \subseteq \langle O \circ \mathcal{E} \rangle$ for some orthogonal 2×2 matrix O, if $\mathcal{F} \subseteq \langle K \circ \mathcal{E} \rangle$, or if $\mathcal{F} \subseteq S \circ \mathcal{A}$ for some $S \in \mathcal{S}$. For all other $\mathcal{F}$ containing a symmetric GHZ-type signature, $\text{Holant}^c(\mathcal{F})$ is #P-hard.

If f is symmetric and has W type, then:

- If $f \notin K \circ \mathcal{M} \cup KX \circ \mathcal{M}$, $\text{Holant}(\{=_2\} \mid \{f\})$ is #P-hard by Theorem 5.
- If $\mathcal{F} \subseteq K \circ \mathcal{M}$ or $\mathcal{F} \subseteq KX \circ \mathcal{M}$, the problem is tractable by the Holant^* dichotomy.
- If $f \in K \circ \mathcal{M}$ but $\mathcal{F} \not\subseteq K \circ \mathcal{M}$, the problem is #P-hard by Lemma 12 below, and analogously with KX instead of K.

If f is not symmetric and $f \notin K \circ \mathcal{M} \cup KX \circ \mathcal{M}$, it is possible to realise a symmetric genuinely entangled ternary signature using Lemmas 7 and 8, so the case reduces to the above.

Finally, if f is not symmetric and $f \in K \circ \mathcal{M}$ (or $f \in KX \circ \mathcal{M}$), then either $\mathcal{F} \subseteq K \circ \mathcal{M}$ (or $\mathcal{F} \subseteq KX \circ \mathcal{M}$) and $\text{Holant}^c(\mathcal{F})$ is in FP, or the problem is hard by Lemma 12 below. This covers all cases. ◀

▶ **Lemma 12.** *Let $f \in \mathcal{F} \cap K \circ \mathcal{M}$ be a genuinely entangled ternary signature, and assume $\mathcal{F} \not\subseteq K \circ \mathcal{M}$. Then $\text{Holant}^c(\mathcal{F})$ is #P-hard. The same holds if $f \in \mathcal{F} \cap KX \circ \mathcal{M}$ and $\mathcal{F} \not\subseteq KX \circ \mathcal{M}$.*

Proof (sketch). We show hardness by either realising a symmetric genuinely entangled ternary signature that is not in $K \circ \mathcal{M} \cup KX \circ \mathcal{M}$ or by realising a symmetric binary entangled signature g such that $\text{Holant}(\{f\} \mid \{g\})$ is #P-hard according to Theorem 5.

The basic approach is the same as in [2], but the techniques need some modification to work in the Holant^c setting. In particular, we show how to realise new unary signatures by gadgets using f, δ_0, δ_1, and self-loops. With these gadgets, we then realise the signatures given above. ◀

These two lemmas show that we can classify the complexity of $\text{Holant}^c(\mathcal{F})$ whenever $\mathcal{F}$ contains a genuinely entangled ternary signature.

4.2 Main theorem

We now have all the components required to prove the main dichotomy for Holant^c. The proof strategy is to realise certain genuinely entangled signatures of low arity. Then:

- If $\mathcal{F} \subseteq \langle \mathcal{T} \rangle$, the problem is known to be tractable.
- If $\mathcal{F}$ contains a genuinely entangled ternary signature, its complexity can be determined by Lemmas 11 and 12.
- If $=_4$ can be realised or interpolated over $\mathcal{F}$, then $\text{Holant}^c(\mathcal{F}) \equiv_T \#\text{CSP}_2^c(\mathcal{F})$ by Lemma 10, so its complexity is determined by Theorem 3.

The arity reduction technique is adapted from that used in the real-valued Holant^c dichotomy [11], with modifications that ensure it works for all complex-valued signatures.

▶ **Theorem 13.** *Let $\mathcal{F}$ be a set of complex-valued signatures. Then $\text{Holant}^c(\mathcal{F})$ is #P-hard unless:*

- *$\mathcal{F}$ is a tractable family for Holant^*,*
- *there exists $S \in \mathcal{S}$ such that $\mathcal{F} \subseteq S \circ \mathcal{A}$, or*
- *$\mathcal{F} \subseteq \mathcal{L}$.*

In all of the exceptional cases, $\text{Holant}^c(\mathcal{F})$ is tractable.

Proof (sketch). If $\mathcal{F}$ is one of the tractable families for Holant*, $\mathcal{F} \subseteq S \circ \mathcal{A}$ for some $S \in \mathcal{S}$, or $\mathcal{F} \subseteq \mathcal{L}$, tractability of Holant$^c(\mathcal{F})$ follows using the same algorithms as employed in the dichotomy proofs for Holant* [6], #CSP [10] (possibly after a holographic transformation), or #CSP$_2^c$ [11]. So assume otherwise. In particular, this implies that $\mathcal{F} \not\subseteq \langle \mathcal{T} \rangle$, i.e. $\mathcal{F}$ has multipartite entanglement.

Without loss of generality, we may focus on genuinely entangled signatures (cf. Section 2.2). So assume that there is some genuinely entangled signature $f \in \mathcal{F}$ of arity $n \geq 3$. If the signature has arity 3, we are done by Lemma 11. Hence assume $n \geq 4$.

As in [11], we now determine the minimum Hamming distance between any pair of bit strings in the support of f, and distinguish cases according to this value. We show that, using f, δ_0, δ_1, and self-loops, it is always possible to realise either a genuinely entangled ternary signature or a 4-ary signature of the form $a\,|0000\rangle + b\,|1100\rangle + c\,|0011\rangle + d\,|1111\rangle$ where $a, b, c, d \in \mathbb{C}$ and $ad - bc \neq 0$. In the former case, we can determine the complexity by Lemmas 11 and 12. In the latter case, we can realise or interpolate the 4-ary equality signature by Lemma 9; then, by Lemma 10, Holant$^c(\mathcal{F}) \equiv_T$ #CSP$_2^c(\mathcal{F})$.

Thus, whenever $\mathcal{F}$ is not one of the tractable families listed in the theorem statement, the problem is #P-hard. ◀

5 Conclusions

Building on the existing dichotomies for real-valued Holant^c and for complex-valued Holant$^+$, we have derived a dichotomy for complex-valued Holant^c. The tractable cases are the complex generalisations of the tractable cases of the real-valued Holant^c dichotomy. The question of a dichotomy for complex-valued, not necessarily symmetric Holant^c had been open since the definition of the family Holant^c in 2009. Several steps in the dichotomy proof use knowledge from quantum information theory, particularly about entanglement. We expect this approach of bringing together Holant problems and quantum information theory to yield further insights into both areas of research in the future. The ultimate goals include a dichotomy for general Holant problems on the one hand, building up on existing results for symmetric functions [5] and non-negative real-valued, not necessarily symmetric functions [18]. On the other hand, we hope to gain further understanding of the complexity of classically simulating quantum circuits.

References

1 Miriam Backens. A complete dichotomy for complex-valued Holant^c. *arXiv:1704.05798 [quant-ph]*, 2017.

2 Miriam Backens. A New Holant Dichotomy Inspired by Quantum Computation. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.16.

3 Miriam Backens. Number of superclasses of four-qubit entangled states under the inductive entanglement classification. *Physical Review A*, 95(2):022329, 2017. doi:10.1103/PhysRevA.95.022329.

4 J. Cai, X. Chen, and P. Lu. Graph Homomorphisms with Complex Values: A Dichotomy Theorem. *SIAM Journal on Computing*, 42(3):924–1029, jan 2013. doi:10.1137/110840194.

5 J. Cai, H. Guo, and T. Williams. A Complete Dichotomy Rises from the Capture of Vanishing Signatures. *SIAM Journal on Computing*, 45(5):1671–1728, 2016. doi:10.1137/15M1049798.

6 J. Cai, P. Lu, and M. Xia. Dichotomy for Holant* Problems of Boolean Domain. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, Proceedings, pages 1714–1728. Society for Industrial and Applied Mathematics, jan 2011. doi:10.1137/1.9781611973082.132.

7 Jin-Yi Cai and Vinay Choudhary. Valiant's Holant Theorem and matchgate tensors. *Theoretical Computer Science*, 384(1):22–32, 2007. doi:10.1016/j.tcs.2007.05.015.

8 Jin-Yi Cai and Zhiguo Fu. Holographic Algorithm with Matchgates is Universal for Planar #CSP over Boolean Domain. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 842–855, New York, NY, USA, 2017. ACM. doi:10.1145/3055399.3055405.

9 Jin-Yi Cai, Sangxia Huang, and Pinyan Lu. From Holant to #CSP and Back: Dichotomy for Holant^c Problems. *Algorithmica*, 64(3):511–533, mar 2012. doi:10.1007/s00453-012-9626-6.

10 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. The complexity of complex weighted Boolean #CSP. *Journal of Computer and System Sciences*, 80(1):217–236, feb 2014. First appeared as "Holant Problems and Counting CSP" in STOC '09. doi:10.1016/j.jcss.2013.07.003.

11 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. Dichotomy for Real Holant^c Problems. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1802–1821. Society for Industrial and Applied Mathematics, 2018. doi:10.1137/1.9781611975031.118.

12 W. Dür, G. Vidal, and J. I. Cirac. Three qubits can be entangled in two inequivalent ways. *Physical Review A*, 62(6):062314, 2000. doi:10.1103/PhysRevA.62.062314.

13 Mariami Gachechiladze and Otfried Gühne. Completing the proof of "Generic quantum nonlocality". *Physics Letters A*, 381(15):1281–1285, apr 2017. doi:10.1016/j.physleta.2016.10.001.

14 Sangxia Huang and Pinyan Lu. A Dichotomy for Real Weighted Holant Problems. *computational complexity*, 25(1):255–304, mar 2016. doi:10.1007/s00037-015-0118-3.

15 L. Lamata, J. León, D. Salgado, and E. Solano. Inductive classification of multipartite entanglement under stochastic local operations and classical communication. *Physical Review A*, 74(5):052336, nov 2006. doi:10.1103/PhysRevA.74.052336.

16 L. Lamata, J. León, D. Salgado, and E. Solano. Inductive entanglement classification of four qubits under stochastic local operations and classical communication. *Physical Review A*, 75(2):022318, feb 2007. doi:10.1103/PhysRevA.75.022318.

17 Dafa Li, Xiangrong Li, Hongtao Huang, and Xinxin Li. Simple criteria for the SLOCC classification. *Physics Letters A*, 359(5):428–437, dec 2006. doi:10.1016/j.physleta.2006.07.004.

18 Jiabao Lin and Hanpin Wang. The Complexity of Holant Problems over Boolean Domain with Non-Negative Weights. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.29.

19 Sandu Popescu and Daniel Rohrlich. Generic quantum nonlocality. *Physics Letters A*, 166(5–6):293–297, 1992. doi:10.1016/0375-9601(92)90711-T.

20 L. Valiant. Holographic Algorithms. *SIAM Journal on Computing*, 37(5):1565–1594, jan 2008. doi:10.1137/070682575.

21 F. Verstraete, J. Dehaene, B. De Moor, and H. Verschelde. Four qubits can be entangled in nine different ways. *Physical Review A*, 65(5):052112, apr 2002. doi:10.1103/PhysRevA.65.052112.

Tight Bounds on Online Checkpointing Algorithms

Achiya Bar-On
Department of Mathematics, Bar-Ilan University, Ramat Gan, Israel
abo1000@gmail.com

Itai Dinur[1]
Computer Science Department, Ben-Gurion University, Beer Sheva, Israel
dinuri@cs.bgu.ac.il

Orr Dunkelman
Computer Science Department, University of Haifa, Haifa, Israel
orrd@cs.haifa.ac.il
https://orcid.org/0000-0001-5799-2635

Rani Hod
Department of Mathematics, Bar-Ilan University, Ramat Gan, Israel
rani.hod@math.biu.ac.il

Nathan Keller[2]
Department of Mathematics, Bar-Ilan University, Ramat Gan, Israel
nkeller@math.biu.ac.il

Eyal Ronen
Computer Science Department, The Weizmann Institute, Rehovot, Israel
eyal.ronen@weizmann.ac.il

Adi Shamir
Computer Science Department, The Weizmann Institute, Rehovot, Israel
adi.shamir@weizmann.ac.il

Abstract

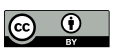
The problem of online checkpointing is a classical problem with numerous applications which had been studied in various forms for almost 50 years. In the simplest version of this problem, a user has to maintain k memorized checkpoints during a long computation, where the only allowed operation is to move one of the checkpoints from its old time to the current time, and his goal is to keep the checkpoints as evenly spread out as possible at all times.

At ICALP'13 Bringmann et al. studied this problem as a special case of an online/offline optimization problem in which the deviation from uniformity is measured by the natural discrepancy metric of the worst case ratio between real and ideal segment lengths. They showed this discrepancy is smaller than $1.59 - o(1)$ for all k, and smaller than $\ln 4 - o(1) \approx 1.39$ for the sparse subset of k's which are powers of 2. In addition, they obtained upper bounds on the achievable discrepancy for some small values of k.

In this paper we solve the main problems left open in the ICALP'13 paper by proving that $\ln 4$ is a tight upper and lower bound on the asymptotic discrepancy for all large k, and by providing tight upper and lower bounds (in the form of provably optimal checkpointing algorithms, some of which are in fact better than those of Bringmann et al.) for all the small values of $k \leq 10$.

2012 ACM Subject Classification Theory of computation → Online algorithms

[1] Supported in part by the Israeli Science Foundation through grant No. 573/16.
[2] The research of Achiya Bar-On, Nathan Keller, and Rani Hod was supported by the European Research Council under the ERC starting grant agreement n. 757731 (LightCrypt) and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 13; pp. 13:1–13:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Keywords and phrases checkpoint, checkpointing algorithm, online algorithm, uniform distribution, discrepancy

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.13

Related Version A full version of the paper is available at `https://arxiv.org/abs/1704.02659`.

1 Introduction and Notation

Most programs perform some irreversible operations, and thus they can only be run in a forward direction. However, in many cases we would like to roll back a computation to an earlier point in time. When the computation is short, we can just rerun the computation from the beginning, but when the computation requires many days, a better strategy is to memorize several copies of the full state of the computation at various times. These memorized states (called *checkpoints*) make it possible to roll the computation back from time T to any earlier time $T' < T$ by restarting the computation from the last available checkpoint which was memorized before T'. This checkpointing technique is extremely useful in many real life applications: For example, when we want to interactively debug a new program we may want to randomly access earlier points in the execution in order to find the source of a problem; in fault tolerant computer systems we may want to undo the effect of faulty hardware; and during lengthy simulations of physical systems we may want to explore the effect of changing some parameter such as the temperature at some earlier point in time without rerunning the simulation from the beginning.

In principle, we can try to memorize the full state of the computation after each step, but for long computations this requires an unrealistic amount of memory. Instead, we assume that we have some bounded amount of memory which suffices to keep k checkpoints. At time T, these checkpoints are spread within the time interval $[0, T]$, dividing it into $k + 1$ subintervals between consecutive checkpoints (where the endpoints 0 and T can be viewed as virtual checkpoints which require no additional memory). As T increases, the last subinterval gets longer, and at some point we may want to relocate one of the old checkpoints by reusing its memory to store the current state of the computation. A checkpointing algorithm can thus be viewed as an infinite pebbling game in which we place k pebbles on the positive side of the time axis, and then repeatedly perform update operations which move one of the pebbles to the right of all the other pebbles.

The first paper dealing with this problem seems to be "Rollback and Recovery Strategies for Computer Programs" [5], published in 1972, while the first paper which tried to solve it optimally was "On the Optimum Checkpoint Interval" [7], published in 1979. Over the years, dozens of academic research papers were published in this area, most notably [8] in 1984, [3] in 1994, and [1, 4] in 2013. However, many of these papers either dealt with concrete applications of the problem in other areas (especially in distributed computing where the notion of a timeline is different), or used other optimization criteria (which make their optimal solutions incomparable with ours). The mathematical problem we are dealing with in this paper was mentioned in [1] and studied in [4] which was published at ICALP'13, and we closely follow their model and notation.

At any time T, we define a *snapshot* as the ordered sequence of current checkpoint locations $S = (T_1, \ldots, T_k)$. Within each snapshot, we refer to the checkpoints by their *freshness index* p, where checkpoint 1 stores the oldest state and checkpoint k stores the newest state. Starting from an initial snapshot $S_k = (t_1, t_2, \ldots, t_k)$, we define for every

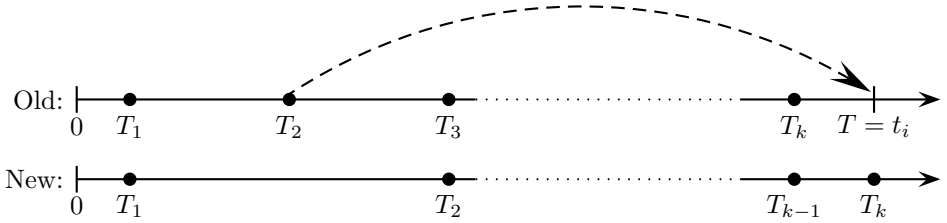


Figure 1 Transition from old to new snapshot for the update action $(t_i, 2)$.

$i \geq k+1$ the i-th *update action* as a pair (t_i, p_i) in which p_i is the freshness index of the checkpoint whose memory we want to reuse by moving it to time t_i. A typical example of how one snapshot is transformed into another snapshot by an update operation is described in Fig. 1. The effect of the i-th update action is to unify the two consecutive subintervals which were separated by the p_i-th oldest active checkpoint at time $T = t_i$, and to create a new subinterval which ends at t_i. Note that with this notation, each update action affects multiple freshness indices within the snapshot; in particular, the freshness index of active checkpoint T_j for $1 \leq j < p_i$ is left unchanged, and is decreased by one for $p_i < j \leq k$. To demonstrate this point, consider a sequence of updates in which $p_i = 1$ for all $i \geq k+1$: it updates the k memory locations in a round robin way since it always updates the oldest active checkpoint by overwriting it with the newest checkpoint, shifting all freshness indices by one. On the other hand, a sequence of updates in which $p_i = k$ for all $i \geq k+1$ keeps updating the same memory location, pushing its associated checkpoint further and further to the right, with no change to the other checkpoints.

In this model, the time complexity of rolling back a computation from time T to time T' is assumed to be proportional to the distance between T' and the last checkpoint that precedes T' in the snapshot at time T, and thus its worst case happens when we decide to roll back to just before the end of the longest subinterval. A *checkpointing algorithm* (t, p) consists of a monotonically increasing and unbounded sequence of update times $t = \{t_i\}_{i=1}^{\infty}$ and a pattern sequence $p = \{p_i\}_{i=k+1}^{\infty}$, forming an initial snapshot and an infinite sequence of update actions; its goal is to make the length of this longest subinterval as short as possible. Clearly, no checkpointing algorithm can make this length shorter than the subinterval length in a perfectly uniform partition of $[0, T]$, which is $T/(k+1)$. We say that a snapshot $S = (T_1, \ldots, T_k)$ of a k-checkpoint algorithm $\textsc{Alg} = (t, p)$ is *q-compliant* at time T if the $k+1$ subintervals defined by S satisfy $T_j - T_{j-1} \leq qT/(k+1)$ for $j = 1, \ldots, k+1$,[3] and that $\textsc{Alg}$ is *q-efficient* if its snapshots are q-compliant at all times $T \geq t_k$. Finally, the *efficiency* of a checkpointing algorithm is defined as the smallest q for which it is q-efficient.

Notice that the problem of efficient checkpointing can be viewed as a special case of an online/offline optimization problem: If we knew in advance the time T at which we would like to roll back the computation, we could make each subinterval as small as $T/(k+1)$. However, in the online version of the problem, we do not know T in advance, and thus we have to position the checkpoints so that they will be roughly equally spaced at all times. The efficiency of the solution is the ratio between what we can achieve in the online and offline cases, respectively, and the goal of the *online checkpointing problem* is to find the smallest possible efficiency q_k achievable by the best k-checkpoint algorithm for any given k.

[3] We write $T_0 = 0$ and $T_{k+1} = T$ for convenience.

Clearly, $q_k \geq 1$ for all $k \geq 2$, and cannot be too close to 1 since any snapshot in which all subintervals have roughly the same length will be transformed by the next update operation to a snapshot in which one of the subintervals will be the union of two previous subintervals, and thus will be about twice as long as the other subintervals.[4] On the other hand, there is a very simple subinterval doubling algorithm from [1, Section 3.1] which is 2-efficient: Assuming WLOG that k is even, the algorithm starts with the snapshot $(1, 2, 3, \ldots, k)$, and performs the sequence of update actions $(k+2, 1), (k+4, 2), \ldots, (2k, k/2)$, yielding the snapshot $(2, 4, 6, \ldots, 2k)$. Since this snapshot is the same as the original snapshot up to a scaling factor of 2, we can continue with update actions $(2k+4, 1), (2k+8, 2), \ldots, (4k, k/2)$ and so on. This is a *cyclic* algorithm, repeating the same sequence of freshness indices again and again but with times which form a geometric progression. As in each snapshot there are only two possible lengths for the subintervals of the form x and $2x$, all the snapshots in this algorithm are 2-compliant, and thus the algorithm is 2-efficient.

The best strategy for keeping the checkpoints as uniform as possible at all times is thus to keep in each snapshot a variety of subinterval lengths, so that the algorithm will always be able to join two relatively short adjacent subintervals into a single subinterval which is not too long. This can be viewed as a generalization of the algorithm that creates Fibonacci numbers: Whereas the standard algorithm is always adding the last two numbers and placing their sum on the right, in our case we can add any two consecutive numbers in the sequence, replacing them by their sum and adding any number we want on the right. Analyzing this problem is surprisingly difficult, and so far there had been no tight bounds on the best possible efficiencies q_k of online checkpointing algorithms in this model.

The main results in [4] are two online checkpointing algorithms whose asymptotic efficiencies are $\ln 4 + o(1) \approx 1.39$ for the sparse subset of k's which are powers of 2, and 1.59 for general k. In addition, they proved in their model the first nontrivial asymptotic lower bound of $2 - \ln 2 - o(1) \approx 1.30$. However, since the upper and lower bounds did not match, it was not clear whether the checkpointing algorithms they proposed were asymptotically optimal. For small values of $k < 60$ they presented concrete checkpointing algorithms whose efficiencies were all below 1.55, but again it was not clear whether they were optimal.

In this paper we solve the main open problems related to the mathematical formulation of the problem which was defined and studied in [4]. In particular, we develop a new checkpointing algorithm with an asymptotic efficiency of $\ln 4$ for all values of k, and prove its optimality by providing a matching asymptotic lower bound. For all the small values of $k < 10$ we develop optimal checkpointing algorithms by proving tight upper and lower bounds on the achievable efficiency for these k's. This analysis enables us to show that for some values of k (such as $k = 8$), the algorithms presented in [4] are in fact suboptimal.

The rest of this paper is organized as follows. In Section 2 we go over basic observations about checkpointing algorithms (some from [4], some new). In Section 3 we focus on moderately small values of k and provide optimal algorithms for $k \leq 10$. In Section 4 we construct a recursive algorithm of asymptotically optimal efficiency $\ln 4 + o(1)$. In Section 5 we prove a matching asymptotic lower bound of $\ln 4 - o(1)$. In Section 6 we provide concluding remarks.

Most proofs were omitted from this extended abstract due to space constraints.

[4] Actually $q_k \geq (k+1)/k$ since subinterval $k+1$ has zero length upon updating, as noted in [1, Theorem 3].

2 Basic Observations

By definition, a k-checkpoint $\textsc{Alg} = (t, p)$ is q-efficient if and only if its snapshots at all times $T \geq t_k$ are q-compliant. However, as noted in [1, Lemma 2] (and also [4, Lemma 1]), it suffices to verify compliance only at the discrete times $T \in \{t_i\}_{i=k}^{\infty}$. It makes sense thus to only consider "standard" snapshots S_i taken at time t_i for $i \geq k$. Moreover, as shown in [4, Lemma 2], besides compliance of the initial snapshot S_k, it suffices to verify compliance of just two subintervals of S_i for every $i > k$ — subinterval k, that ends in the new checkpoint t_i, and subinterval p_i, created by merging two consecutive subintervals.

The following two observations about the sequence $p = (p_i)_{i=k+1}^{\infty}$ were mentioned in [4, Section 6] without proof.

▶ **Fact 1.** *Without loss of generality we can assume a k-checkpoint algorithm updates the least recent checkpoint infinitely often (i.e., $\liminf_{i\to\infty} p_i = 1$).*

▶ Remark 2. An important consequence of Fact 1 is that we can essentially ignore the compliance of the initial snapshot S_k by *rebasing*, i.e., running the algorithm until all checkpoints present in S_k are overwritten and treating the then-current snapshot as the new initial $(t_1, \ldots, t_k)$.

▶ **Fact 3.** *Without loss of generality we can assume a k-checkpoint algorithm never updates the most recent checkpoint (i.e., $p_i < k$ for all $i \geq 1$).*

▶ Remark 4. Fact 3 means that the two last checkpoints in snapshot S_i are t_{i-1} and t_i, and thus subinterval k is q-compliant if and only if $t_{i-1} - t_i \leq qt_i/(k+1)$, that is, $t_i \leq G \cdot t_{i-1}$, where $G = G(q) := (k+1)/(k+1-q)$. We refer to this condition by saying that the update times sequence $t = (t_i)_{i=1}^{\infty}$ should be *G-subgeometric*.

Next we introduce the notion of *cyclic* algorithms. Upper bounds on q_k presented in this paper, as well as in [1, 4], are all achieved by cyclic algorithms. Given a positive integer n and a real number $\gamma > 1$, a k-checkpoint algorithm $\textsc{Alg} = (t, p)$ is *(n, γ)-cyclic* if $t_{n+i} = \gamma \cdot t_i$ for all $i \geq 1$ and $p_i = p_{n+i}$ for all $i \geq k+1$. It has been observed in [4, Lemma 5] that any q-efficient (n, γ)-cyclic algorithm must satisfy $\gamma \leq G^n$ (to see this, apply subgeometry n times). An (n, γ)-cyclic algorithm is called *(n, G)-geometric* when $\gamma = G^n$ (and thus $t_{i+1} = G \cdot t_i$ for $i \geq k$).

We finish this subsection with two observations about the exponential growth of update times in efficient algorithms, relevant for upper and lower bounds on q_k.

The first one is an improvement of [4, Lemma 8]:

▶ **Fact 5.** *Any q-efficient k-checkpoint algorithm $\textsc{Alg} = (t, p)$ satisfies, without loss of generality, $t_{i+2} > t_i \cdot G$ for all $i \geq k$.*

An immediate corollary of Fact 5 is that $t_{i+j} > t_i \cdot G^{\lfloor j/2 \rfloor}$ for $j \geq 0$; in particular, $t_{i+j} > t_i \cdot G^2$ for $j \geq 4$. Our next observation says when we can get $t_{i+3} > t_i \cdot G^2$.

▶ **Fact 6.** *Let $S = (T_1, \ldots, T_k)$ be a snapshot of some q-efficient k-checkpoint algorithm $\textsc{Alg} = (t, p)$ such that $T_j = t_i$ and $T_{j+1} = t_{i+3}$ for some $j = 1, \ldots, k-1$. Thus, without loss of generality, $t_{i+3} > t_i \cdot G^2$.*

3 Optimal Algorithms for Small Values of k

3.1 Round-robin and $k \leq 5$

We now analyze the efficiency of the ROUND-ROBIN algorithm, which is geometric and always updates the oldest checkpoint (i.e., $p_i = 1$ for all $i \geq k+1$).[5] Besides serving as a first example, ROUND-ROBIN is optimal for $k \leq 3$ and will make an appearance within the asymptotically optimal algorithm RECURSIVE of Section 4.

▶ **Proposition 7.** *The efficiency of k-checkpoint* ROUND-ROBIN *is $q = (k+1)\,r$, where r is the smallest real root of $x = (1-x)^{k-1}$.*

▶ Remark. ROUND-ROBIN is pretty bad for large k; indeed, $(k+1)\,r \approx \ln k - \ln\ln k$ is asymptotically inferior to the simple bound $q_k \leq 2$ from the introduction.

The case $k = 2$ is made obvious by Fact 3, since without loss of generality ROUND-ROBIN is the *only* 2-checkpoint algorithm to consider. Thus $q_2 = 1.5$.

▶ **Proposition 8.** *For $k = 3$ we have $q_3 = 4r_3 \approx 1.52786$, where $r_3 = \frac{3-\sqrt{5}}{2} \approx 0.38197$ is the smaller root of $x^2 - 3x + 1 = 0$.*

Proof. For the upper bound, ROUND-ROBIN is $4r_3$-efficient. For the lower bound, consider a $4r$-efficient 3-checkpoint algorithm and a snapshot $S_i = (x, y, t_i)$. By subgeometry we must have $t_i \leq y/(1-r) \leq x/(1-r)^2$ and for subinterval 1 to be compliant we need $x \leq rt_i$, which together imply $(1-r)^2 \leq r$, i.e., $r^2 - 3r + 1 \leq 0$. Thus $r \geq r_3$. ◀

ROUND-ROBIN is no longer optimal for $k > 3$. Indeed, cyclic algorithms with better efficiency were described in [4, Figure 3] for $k = 4, 5, 6, 7, 8$. These provide upper bounds on $q_4, \ldots, q_8$, respectively. Nevertheless, no formal proof of optimality was provided.

▶ Remark. These algorithms were found by the use of linear programming, which is thoroughly discussed in the next subsection.

For $k = 4, 5$ the optimal algorithms are 2-cyclic; $k = 5$ is geometric while $k = 4$ is not.

▶ **Proposition 9.** *For $k = 4$ we have $q_4 = 5r_4 \approx 1.53989$, where $r_4 = (2 + 2\cos(2\pi/7))^{-1} \approx 0.307979$ is the smallest root of $x^3 - 5x^2 + 6x - 1 = 0$. Moreover, the efficiency of any geometric 4-checkpoint algorithm is at least $5\tilde{r}_4 \approx 1.58836$, where $\tilde{r}_4 \approx 0.31767 > r_4$ is the real root of $x^3 - 3x^2 + 4x - 1 = 0$.*

▶ **Proposition 10.** *For $k = 5$ we have $q_5 = 6r_5 \approx 1.47073$, where $r_5 \approx 0.24512$ is the (only) real root of $x^3 - 4x^2 + 5x - 1 = 0$.*

3.2 Casting the problem as a linear program

Fix $\lambda \geq 1$ and an update pattern $p = (p_i)_{i=k+1}^{\infty}$. Can we choose a sequence $t = (t_i)_{i=1}^{\infty}$ of update times such that the resulting k-checkpoint algorithm $\text{ALG} = (t, p)$ is λ-efficient?

Each snapshot S_i consists of a particular subset of the variables t, and using p we can determine exactly which. Furthermore, all constraints (e.g., monotonicity, subgeometry, compliance) can be expressed as linear inequalities. This gives rise to an infinite linear program $L = L(\lambda; p)$, which is feasible whenever a λ-efficient algorithm with the prescribed

[5] The case $k = 3$ of ROUND-ROBIN was considered in [4, Theorem 1] under the name SIMPLE.

pattern p exists. Note that all constraints are homogeneous, so to avoid the zero solution we add the non-homogeneous condition $t_k = 1$.

In addition, we are not interested in solutions where t is bounded. This can happen, for instance, when $p_i = k - 1$ for all $i \geq k + 1$.[6] Luckily, by using Fact 5 we can restrict our attention to exponentially increasing sequences t, so we add to L the linear inequalities from Facts 5 and 6. Now L is feasible if and only if a λ-efficient algorithm with the prescribed pattern p exists; in other words, q_k is the infimum[7] over $\lambda \geq 1$ for which there exists a pattern p such that $L(\lambda; p)$ is feasible.

As an infinite program, L is not too convenient to work with. We can thus limit our attention to finite subprograms $L(\lambda; (p_{k+1}, \ldots, p_{k+n}))$ for some $n \in \mathbb{N}$, which only involve the $k + n$ variables $t_1, \ldots, t_{k+n}$ and the relevant $3k + 6n$ constraints. Finite subprograms can no longer ensure the existence of a λ-efficient algorithm, but can be used to prove lower bounds on q_k in the following way. Write $\Sigma = \{1, \ldots, k - 1\}$ and consider the set Σ^* of strings, i.e, finite sequences over Σ.

▶ **Definition.** A string $B \in \Sigma^*$ is called a *λ-witness* if $L(\lambda; B)$ is infeasible. A string set $\mathcal{B} \subset \Sigma^*$ is called *blocking* if any infinite sequence p over Σ contains some $B \in \mathcal{B}$ as a substring.

▶ **Fact 11.** *If there exists a blocking set of λ-witnesses for some $\lambda \geq 1$, then $q_k > \lambda$.*

▶ Remark. The lower bound of Fact 11 holds for all algorithms, cyclic or not.

We now describe a strategy to approximate q_k to arbitrary precision. For the lower bound we use Fact 11; for the upper bound, we limit our focus to cyclic algorithms. Given $\gamma > 1$ and a string $P \in \Sigma^*$ of length n, we can augment $L(\lambda; P)$ with k equality constraints $\{t_{i+n} = \gamma \cdot t_i\}_{i=1}^{k}$; call the resulting program $L^*(\lambda, \gamma; P)$. This is a finite linear program, which we can computationally solve given λ, γ, and P. Although $\gamma \leq G^n$ is not known to us, we can first compute an approximation $\tilde{\gamma}$ of γ by solving $L_{10n}\left(\gamma; P^{10}\right)$, and then solve $L^*(\lambda, \tilde{\gamma}; P)$. Using binary search, we can compute a numerical approximation $\tilde{\lambda}$ of the minimal λ for which $L^*(\lambda, \tilde{\gamma}; P)$ is feasible. Lastly, we can enumerate short strings $P \in \Sigma^*$ in a BFS/DFS-esque manner and take the best $\tilde{\lambda}$ obtained.

To demonstrate this strategy, we computed $q_2, \ldots, q_{10}$ up to 7 decimal digits, using a Python program employing GLPK [6] via CVXOPT [2] (see Table 1; starred values of k are geometric algorithms).

At first it seems that Fact 11 cannot be used to pinpoint q_k exactly, since any finite blocking set $\mathcal{B}$ of $(q_k - \epsilon)$-witnesses for some $\epsilon > 0$ leaves an interval of uncertainty of length ϵ. The following proposition eliminates this uncertainly.

▶ **Proposition 12.** *For every string $B \in \Sigma^*$ there is finite set $\Lambda_B \subset \mathbb{R}$ such that the feasibility of $L(\lambda; B)$ for some $\lambda \geq 1$ only depends on the relative order between λ and members of Λ_B. In particular, there exists some $\epsilon > 0$ such that if $L(\lambda; B)$ is feasible and B is a $(\lambda - \epsilon)$-witness, then B is also a λ'-witness for all $\lambda - \epsilon < \lambda' < \lambda$.*

Proof. Fix $B \in \Sigma^*$. Treating λ as a parameter, note that the subprogram $L(\lambda; B)$ is feasible if and only if its feasible region, the convex polytope $\mathcal{P}(\lambda; B)$, is nonempty. Decreasing λ shrinks $\mathcal{P}(\lambda; B)$ until some critical λ_B for which $\mathcal{P}(\lambda_B; B)$ is reduced to a single vertex, at which a subset of the linear constraints are satisfied with equality. Hence λ_B is a solution of

[6] This may not seem a valid pattern to consider, given Fact 1; however, when solving a finite subprogram we might have to consider an arbitrarily long prefix of the pattern with no occurrences of 1.

[7] This infimum is actually a minimum, by [4, Theorem 8] and also by Proposition 12.

Table 1 Computationally-verified bounds on q_k for $2 \leq k \leq 10$.

k	q_k	γ	P	n	$\lvert\mathcal{B}\rvert$	$\max_{B\in\mathcal{B}} \lvert B\rvert$
2*	1.5	2	(1)	1	0	0
3*	1.5278641	1.618037	(1)	1	2	1
4	1.5398927	1.8019377	(1,3)	2	7	3
5*	1.4707341	1.7548777	(1,3)	2	36	13
6	1.5127400	3.627365	(1,2,3,1,3,5)	6	117	9
7	1.4974818	3.11201	(1,3,4,1,5,3)	6	559	10
8	1.4851548	10.712656	(1,2,4,7,5,3,1,7,5,3,7,1,4,2,4,5)	16	1698	14
9	1.4730721	3.2748095	(1,5,3,5,1,5,6,3)	8	5892	135
10	1.4678452	5.67943	(1,5,3,5,1,5,6,3,1,5,9,3,5,9)	14	32843	20

some polynomial equation determined by the relevant constraints. The set of constraints is finite, thus there are finitely many polynomial equations that can define λ_B, and we can take Λ_B as the set of all their roots. Now take ϵ to be smaller than the distance between any two distinct elements of Λ_B. ◀

▶ Remark. Note that when ϵ is small enough, we can actually retrieve the polynomial equations defining λ and γ from the polytope $\mathcal{P}^*\left(\tilde{\lambda}, \tilde{\gamma}; B\right)$; using this method we get an algebraic representation of q_k rather than a rational approximation. To demonstrate, $q_9 = 10r_9$, where $r_9 \approx 0.1473072131$ is the smallest real root of $x^8-7x^7+22x^6-40x^5+39x^4-17x^3+10x^2-8x+1$.

4 Asymptotically Optimal Upper Bounds

In this section we describe a family of geometric k-checkpoint algorithms. Despite our experience from Table 1—that only for $k = 2, 3, 5$ optimal algorithms are geometric—this family is rich enough to be asymptotically optimal, i.e., $(1 + o(1))\, q_k$-efficient.

4.1 A recursive geometric algorithm

Fix a real number $G > 1$ and an integer $m \geq 0$. We describe a k-checkpoint algorithm RECURSIVE(G, K), where K is an $(m + 2)$-subset $\{0, \ldots, k\}$ whose elements are

$$k = k_0 > k_1 > k_2 > \cdots > k_m > k_{m+1} = 0.$$

RECURSIVE(G, K) is $(2^m, G)$-geometric, and its update pattern p is defined as $p_{k+i} = 1 + k_{\mu(i)+1}$, where $\mu(i)$ is the largest $\mu \leq m$ for which 2^μ divides i. It is easy to see that p is 2^m-periodic, and we can just refer to $P = (p_{k+i})_{i=1}^{2^m}$. As per Remark 2, via rebasing there is no need to define the initial snapshot S_k explicitly.

▶ **Example.** For $K = \{0, 2, 4, 9, 19\}$ we get $P = (10, 5, 10, 3, 10, 5, 10, 1)$.

True to its name, RECURSIVE(G, K) can be viewed also as a recursive algorithm: the base case $m = 0$ (i.e., $K = \{0, k\}$) is simply k-checkpoint ROUND-ROBIN; for $m \geq 1$, RECURSIVE(G, K) alternates between updating the $(k_1 + 1)$-st oldest checkpoint and between acting according to the inner k_1-checkpoint algorithm RECURSIVE$\left(G^2, K \setminus \{k\}\right)$.

Let us elaborate a bit more on the recursive step. In every snapshot $S_i = (T_1, \ldots, T_k)$ we have $T_j = G^{i+j}$ for $k_1 + 1 \leq j \leq k$ since we never update checkpoints younger than $k_1 + 1$. In every *odd* snapshot S_i we have just updated the $(k_1 + 1)$-st oldest checkpoint,

so $T_{k_1} = G^{i+k_1-1}$ while $T_{k_1+1} = G^{i+k_1+1}$. This means that $\log_G T_j$ for $j = 1, \ldots, k_1$ all have the same parity as $i + k_1 - 1$ in any snapshot S_i. We thus treat $S' = (T_1, \ldots, T_{k_1})$ as a snapshot of a k_1-checkpoint algorithm, which operates at half speed and never sees half of the checkpoints. The inner algorithm can rightfully be called $\textsc{Recursive}\left(G^2, K \setminus \{k\}\right)$, as the common ratio of the update times sequence for the checkpoints that do make it to the inner algorithm is G^2, and taking only the even locations of P yields a 2^{m-1} periodic sequence p' such that $p'_{k+i} = p_{k+2i} = 1 + k_{\mu(2i)+1} = 1 + k_{\mu(i)+2}$.

4.2 Analyzing the recursive algorithm

First we determine exactly how efficient $\textsc{Recursive}(G, K)$ can be for any G and K, and then we work with a particular choice.

Denote by $r(G, K)$ the maximum of

$$1 - G^{-1}; \tag{1a}$$

$$\max\left\{G^{-e(\ell)}\left(G^{2^\ell} - G^{-2^\ell}\right)\right\}_{\ell=0}^{m-1}; \text{ and} \tag{1b}$$

$$G^{2^m - e(m)}, \tag{1c}$$

where $e(\ell) = \sum_{j=0}^{\ell} 2^j (k_j - k_{j+1})$ for $\ell = 0, 1, \ldots, m$.

▶ **Theorem 13.** *Given G and K, the efficiency of $\textsc{Recursive}(G, K)$ is $(k+1)r(G, K)$.*

Given an integer $k \geq 2$, let $m = \lfloor \log_2 k \rfloor - 1$. Define $K^* = \{k_0, \ldots, k_{m+1}\}$ by $k_j = \lfloor 2^{-j} k \rfloor$ for $j = 0, \ldots, m$ and $k_{m+1} = 0$. Note that $k_0 = k$ and that $k_m \in \{2, 3\}$.

▶ **Theorem 14.** $\textsc{Recursive}(G, K^*)$ *is q-efficient for large enough k, where $G = e^{q/(k+1)}$ and $q = \left(1 + \frac{3}{\log_2 k}\right) \frac{k+1}{k} \ln 4 = (1 + o(1)) \ln 4$.*

Proof. By Theorem 13, it suffices to verify that $(k+1)r(G, K^*) < q$, that is, $r(G, K^*) \leq \ln G$ for sufficiently large k. Clearly (1a) holds since $1 - G^{-1} < \ln G$ for all $G > 1$. It remains to verify (1b) and (1c), handled by Propositions 15 and 16 respectively. ◀

▶ **Proposition 15.** *For $k \geq 2^{13}$ and G as above, $G^{-e(\ell)}\left(G^{2^\ell} - G^{-2^\ell}\right) < \ln G$ for all $\ell = 0, \ldots, m-1$.*

▶ **Proposition 16.** *For $k \geq 5$ and G as above, $G^{2^m - e(m)} < \ln G$.*

▶ Remark. Theorem 14 chooses G suboptimally. Empirical evidence shows that, for all $k \geq 2$, the optimal $G = G^*$ for $\textsc{Recursive}(G, K^*)$ satisfies (1a) and one of (1b) and (1c). In other words, it is the smallest root of either $1 - x^{-1} = x^{2^m - e(m)}$ or $1 - x^{-1} = x^{-e(\ell)}\left(x^{2^\ell} - x^{-2^\ell}\right)$ for some $\ell = 0, \ldots, m-1$.

▶ Remark. With additional effort the constant 3 in Theorem 14 can be improved by a factor of almost 6 to $\tau := -\log_2 \ln 2 \approx 0.53$. The major obstacle is that cases $\ell = m-2$ and $\ell = m-1$ of (1b) need to be done separately since the appropriate $f(x, z)$ in the proof of Proposition 15 is negative for $z < \log_2(\ln 4 / (1 - \ln 2)) \approx 2.1756$. No proof is possible for $\tau' < \tau$ since then (1c) would be violated for large enough $k = 2^{m+2} - 1$.

▶ Remark. We verified that the algorithm $\textsc{Recursive}(G^*, K^*)$ is $(1 + \tau / \log_2 k) \frac{k+1}{k} \ln 4$-efficient for $2 \leq k \leq 2^{13}$ as well.

5 Asymptotically Optimal Lower Bounds

In this section we prove lower bounds on q_k, focusing on asymptotic lower bounds in which k grows to infinity.

We start by reproving the simple asymptotic lower bound $q_k \geq 2 - \ln 2 - o(1)$ of [4, Theorem 6], and then improve it to $q_k \geq \ln 4$, which is asymptotically optimal via the matching upper bound of Section 4.

5.1 Stability and bounding expressions

Obtaining lower bounds requires viewing the problem from a different perspective. It will sometimes be more convenient to refer to a certain physical checkpoint, without considering its temporary freshness index p in the checkpoint sequence at some snapshot S (which is variable and depends on S).

Given a k-checkpoint algorithm, we define a function $BE(s)$ and use it to bound its efficiency from below. The parameter s is related to the notion of stability, which we now define.

▶ **Definition.** Fix a k-checkpoint algorithm. A checkpoint updated at time T is called s-stable, for some $s = 1, \ldots, k-1$, if at least s previous checkpoints are updated before the next time it is updated.

By Fact 1 we can assume all checkpoints get updated eventually; this means that in a snapshot $S = (T_1, \ldots, T_k)$, where T_k is a time by which all checkpoints have been updated from the initial snapshot, we have that the checkpoint updated at time T_{k-s} is s-stable for $s = 1, \ldots, k-1$.

For convenience, the proofs in this section assume the update times sequence is normalized by a constant. This is captured by the following definition.

▶ **Definition.** A k-checkpoint algorithm is called s-normalized if an s-stable checkpoint is updated at time $R_0 = 1$.

Given an s-normalized k-checkpoint algorithm, we define a sequence of times $1 = R_0 < R_1 < \cdots < R_s$ as follows: R_i for $i \geq 1$ is the time at which the i-th checkpoint is *removed* from $(0, 1]$. In other words, R_1 is the time $T > R_0$ at which some checkpoint is updated; R_2 is the time $T > R_1$ at which we update the next checkpoint that was previously updated in $(0, 1]$ (but not at $T > 1$), and so forth. Note that the checkpoint updated at time R_0 is not updated at any time R_i for $i = 1, \ldots, s$ by the definition of stability. Now we are ready to define $BE(s)$.

▶ **Definition.** The T-truncated bounding expression of an s-normalized k-checkpoint algorithm is $BE_T(s) = \sum_{i=1}^{s} U_i$, where $U_i = \min\{T, R_i\}$.

The bounding expression plays a crucial role in proving lower bounds, based on Proposition 17 below. We note that the truncated bounding expression only depends on the algorithm's behavior until time T, and hence the bounds that can be obtained from it are not tight for $k > 3$. Nevertheless, the lower bound we obtain using BE_2 in Corollary 22 is asymptotically optimal, since the gap between it and the upper bound of Theorem 14 tends to zero as k grows to infinity.

▶ Remark. It is possible to analyze BE_T beyond $T = 2$ and obtain tight lower bounds for larger values of k. However, there is no asymptotic improvement and the analysis becomes increasingly more technical as k grows.

5.2 Asymptotic lower bound of $2 - \ln 2 \approx 1.3068$

To simplify the analysis, we assume k is even. It can be extended to cover odd values of k as well, but this gives no asymptotic improvement since $q_{k+1} \leq q_k \cdot \frac{k+2}{k+1}$ for all k, so we only lose an error term of $O(1/k)$, which is of the same order as the error terms in Corollaries 20 and 22.

To simplify our notation we write $b = q/(k+1)$ throughout this section.

▶ **Proposition 17.** *Any $(k/2)$-normalized $b(k+1)$-efficient k-checkpoint algorithm satisfies $BE_2(k/2) \geq 1/b$.*

Proof. At time $R_0 = 1$, the time interval $(0, 1]$ contains k subintervals of length $\leq b$, giving rise to the inequality $b \cdot k \geq 1$. At time R_1, a checkpoint is removed from $(0, 1]$ and it now contains one subinterval of length $\leq b \cdot R_1$ (two previous subintervals, each of length $\leq b$, were merged), and $k-2$ subintervals of length $\leq b$, giving $b \cdot (k - 2 + R_1) \geq 1$.

At time R_2, an additional checkpoint is removed from the time interval $(0, 1]$, hence it must contain an subinterval of length $\leq b \cdot R_2$ formed by merging two previous subintervals. We obtain $b \cdot (k - 4 + R_1 + R_2) \geq 1$, since the remaining $k-3$ subintervals must include $k-4$ subintervals of length $\leq b$ and one (additional) subinterval of length at most $\leq b \cdot R_1$. Note that this claim holds regardless of which checkpoint is updated at R_2, and it holds in particular in case one of the subintervals merged at time R_2 contains the subintervals merged at R_1 (in fact, this case gives the stronger inequality $b \cdot (k - 3 + R_2) \geq 1$).

In general, for $j = 1, 2, \ldots, k/2$, at time R_j the time interval $(0, 1]$ must contain j distinct subintervals of lengths $\leq b \cdot R_i$ for $i = 1, 2, \ldots, j$, and $k - 2j$ subintervals of length $\leq b$. This gives the inequality $k - 2j + \sum_{i=1}^{j} R_i \geq 1/b$.

Let $j \leq k/2$ be the largest index such that $R_j \leq 2$, so $U_i = R_i$ for all $1 \leq i \leq j$ and $U_i = 2$ for all $j < i \leq k/2$. Now at time R_j we have

$$1/b \leq k - 2j + \sum_{i=1}^{j} R_i = (k/2 - j) \cdot 2 + \sum_{i=1}^{j} U_i = \sum_{i=j+1}^{k/2} U_i + \sum_{i=1}^{j} U_i = BE_2(k/2). \quad \blacktriangleleft$$

Now we need an upper bound on the bounding expression. For the simpler lower bound of $2 - \ln 2$ we use the following proposition.

▶ **Proposition 18.** *Any $(k/2)$-normalized $b(k+1)$-efficient k-checkpoint algorithm satisfies $R_i \leq 1/(1 - bi)$ for $i = 1, \ldots, k/2$.*

Proof. At time $T = 1/(1 - bi)$, all subintervals are of length at most $bT = b/(1 - bi)$. Since $T - R_0 = 1/(1 - bi) - 1 = bi/(1 - bi)$, for any $\epsilon > 0$ the time interval $(R_0, T + \epsilon]$ must consist of at least $i + 1$ subintervals, implying that the i-th checkpoint was removed from the time interval $(0, 1]$ by time T. ◀

▶ **Proposition 19.** *Let $b < \frac{1}{2}$. Any $(k/2)$-normalized $b(k+1)$-efficient k-checkpoint algorithm satisfies $b \cdot BE_2(k/2) \leq \ln 2 + b/(1 - 2b) + bk - 1$.*

▶ **Corollary 20.** *For all even $k \geq 4$ we have $q_k \geq 2 - \ln 2 - o(1)$.*

Proof. Fix a $(k/2)$-normalized q_k-efficient k-checkpoint algorithm, and let $b = q_k/(k+1) < \frac{1}{2}$. By Propositions 17 and 19 we have

$$q_k = bk + b \geq 2 - \ln 2 - \frac{b}{1 - 2b} + b = 2 - \ln 2 - \frac{2b^2}{1 - 2b} \geq 2 - \ln 2 - \frac{8}{(k+1)(k-3)}. \quad \blacktriangleleft$$

5.3 Improved asymptotic lower bound of $\ln 4 \approx 1.3863$

We now improve the asymptotic lower bound to $\ln 4$. This result is a simple corollary of the following lemma, which gives a tighter upper bound on the bounding expression. Recall that $q = b(k+1)$ and thus $G = (k+1)/(k+1-q) = 1/(1-b)$.

▶ **Lemma 21.** *For any s-normalized $b(k+1)$-efficient k-checkpoint algorithm such that $1 \le s \le k/2$ and $G^{k/2} \le 2$ we have $b \cdot BE_2(s) \le G^s - 1$.*

▶ **Corollary 22.** *For all even $k \ge 2$ we have $(1 - q_k/(k+1))^{-k/2} \ge 2$. In particular, $q_k > \ln 4$.*

Proof. Write $b = q_k/(k+1)$ and assume for the sake of contradiction that $G^{k/2} < 2$. By Lemma 21 and Proposition 17 we have $1 \le b \cdot BE_2(k/2) \le G^{k/2} - 1$ for a $k/2$-normalized q_k-efficient k-checkpoint algorithm, so $G^{k/2} \ge 2$, contradicting our assumption. Now

$$\frac{q_k}{k+1} = b \ge 1 - 2^{-2/k} = 1 - e^{-(\ln 4)/k} \ge \frac{\ln 4}{k} - \frac{1}{2}\left(\frac{\ln 4}{k}\right)^2 = \left(1 - \frac{\ln 2}{k}\right)\frac{\ln 4}{k},$$

hence $q_k \ge \left(1 + (1 - \ln 2)/k - (\ln 2)/k^2\right)\ln 4 > \ln 4$. The last inequality is true when $k > \ln 2/(1 - \ln 2) \approx 2.26$, but we already know that $q_2 = 1.5 > \ln 4$. ◀

6 Concluding Remarks and Open Problems

In this paper we solved the main open problem in online checkpointing algorithms, which is to find tight asymptotic upper and lower bounds on their achievable efficiency. In addition, we developed efficient techniques for determining tight upper and lower bounds on q_k for small values of k, which enabled us to develop provably optimal concrete algorithms for all $k \le 10$. However, determining the values of q_k for larger values of k remains a computationally challenging problem, and finding more efficient ways to compute these values remains an interesting open problem.

References

1 Lauri Ahlroth, Olli Pottonen, and André Schumacher. Approximately Uniform Online Checkpointing with Bounded Memory. *Algorithmica*, 67(2):234–246, 2013. doi:10.1007/s00453-013-9772-5.

2 M.S. Andersen, J. Dahl, and L. Vandenberghe. CVXOPT: A Python package for convex optimization, 2016. version 1.1.9. Available at http://cvxopt.org.

3 Marshall W. Bern, Daniel H. Greene, Arvind Raghunathan, and Madhu Sudan. On-Line Algorithms for Locating Checkpoints. *Algorithmica*, 11(1):33–52, 1994. doi:10.1007/BF01294262.

4 Karl Bringmann, Benjamin Doerr, Adrian Neumann, and Jakub Sliacan. Online Checkpointing with Improved Worst-Case Guarantees. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 255–266, 2013. doi:10.1007/978-3-642-39206-1_22.

5 K. Mani Chandy and Chittoor V. Ramamoorthy. Rollback and Recovery Strategies for Computer Programs. *IEEE Trans. Computers*, 21(6):546–556, 1972. doi:10.1109/TC.1972.5009007.

6 Free Software Foundation. Gnu linear programming kit, 2012. Version 4.61, http://www.gnu.org/software/glpk/.

7 Erol Gelenbe. On the Optimum Checkpoint Interval. *J. ACM*, 26(2):259–270, 1979. doi:10.1145/322123.322131.

8 Sam Toueg and Özalp Babaoglu. On the Optimum Checkpoint Selection Problem. *SIAM J. Comput.*, 13(3):630–649, 1984. doi:10.1137/0213039.

Table 2 Computationally-verified upper bounds on q_k for $11 \leq k \leq 20$.

k	λ	γ	P	n
11	1.4650841	8.190656	(1,3,5,6,1,6,2,10,6,3,6,1,6,2,6,3,9,6)	18
12	1.4668421	8.862576	(1,2,3,5,6,7,1,2,6,3,6,7,1,2,6,3,6,9,7)	19
13	1.4592320	2.94	(1,3,6,7,4,7,1,7,8,3)	10
14	1.4570046	58.6	(1,4,2,6,7,4,7,8,1,8,2,3,7,12,4,7,8,1,4,7,2,7,8,4,13,8, 1,8,4,2,7,4,7,8,1,8,4,2,7,12,4,7,13,8)	44
15	1.4487459	2.104027	(1,2,7,8,4,8,9,5)	8
16	1.4487597	8.46	(1,2,4,7,8,9,5,9,1,2,8,4,8,9,5,9,1,2,8,4,8,13,9,5,9)	25
17	1.4593611	1.694884	(1,9,5,3,14,8,9)	7
18	1.4575670	2.57	(1,8,9,5,9,10,2,5,9,10,3,5)	12
19	1.4592194	2.45	(1,9,5,9,10,2,5,9,10,11,3,5)	12
20	1.4696048	13.3	(1,5,9,10,2,5,9,10,11,3,5,10,1,5,9,10,11,2,5,10,11,3,5,10, 1,5,9,10,6,2,9,10,11,3,5,10)	36

A Tables

The best algorithms our LP approach of Section 3.2 found for $k = 11, 12, \ldots, 20$ are described in Table 2. These are (perhaps non-tight) upper bounds on $q_{11}, \ldots, q_{20}$. Observe how some of the patterns are reminiscent of the pattern used in the algorithm RECURSIVE of Section 4.

Fast Reed-Solomon Interactive Oracle Proofs of Proximity

Eli Ben-Sasson
Technion, Haifa, Israel
eli@cs.technion.ac.il
https://orcid.org/0000-0002-0708-0483

Iddo Bentov
Cornell University, Ithaca, NY, USA
iddo333@gmail.com

Yinon Horesh
Technion – Israel Institute of Technology, Haifa, Israel
ynon980@gmail.com

Michael Riabzev
Technion – Israel Institute of Technology, Haifa, Israel
riabzevmichael@gmail.com

Abstract

The family of Reed-Solomon (RS) codes plays a prominent role in the construction of quasilinear probabilistically checkable proofs (PCPs) and interactive oracle proofs (IOPs) with perfect zero knowledge and polylogarithmic verifiers. The large concrete computational complexity required to prove membership in RS codes is one of the biggest obstacles to deploying such PCP/IOP systems in practice.

To advance on this problem we present a new interactive oracle proof of proximity (IOPP) for RS codes; we call it the *Fast RS IOPP* (FRI) because (i) it resembles the ubiquitous Fast Fourier Transform (FFT) and (ii) the arithmetic complexity of its prover is strictly linear and that of the verifier is strictly logarithmic (in comparison, FFT arithmetic complexity is quasi-linear but not strictly linear). Prior RS IOPPs and PCPs of proximity (PCPPs) required super-linear proving time even for polynomially large query complexity.

For codes of block-length N, the arithmetic complexity of the (interactive) FRI prover is less than $6 \cdot N$, while the (interactive) FRI verifier has arithmetic complexity $\leq 21 \cdot \log N$, query complexity $2 \cdot \log N$ and constant soundness – words that are δ-far from the code are rejected with probability $\min\{\delta \cdot (1 - o(1)), \delta_0\}$ where δ_0 is a positive constant that depends mainly on the code rate. The particular combination of query complexity and soundness obtained by FRI is better than that of the quasilinear PCPP of [Ben-Sasson and Sudan, SICOMP 2008], even with the tighter soundness analysis of [Ben-Sasson et al., STOC 2013; ECCC 2016]; consequently, FRI is likely to facilitate better concretely efficient zero knowledge proof and argument systems.

Previous concretely efficient PCPPs and IOPPs suffered a constant *multiplicative* factor loss in soundness with each round of "proof composition" and thus used at most $O(\log \log N)$ rounds. We show that when δ is smaller than the unique decoding radius of the code, FRI suffers only a negligible *additive* loss in soundness. This observation allows us to increase the number of "proof composition" rounds to $\Theta(\log N)$ and thereby reduce prover and verifier running time for fixed soundness.

2012 ACM Subject Classification Theory of computation → Interactive proof systems

Keywords and phrases Interactive proofs, low degree testing, Reed Solomon codes, proximity testing

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 14; pp. 14:1–14:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.14

Related Version Electronic Colloquium on Computational Complexity, report TR17-134 [10].

Funding Supported by the European Research Council under POC grant OMIP – DLV-693423, an Israel Science Foundation grant 1501/14 and the USA–Israel binational science fund, grant # 2014359.

Acknowledgements We thank Justin Drake, Peter Manohar and Nicholas Spooner for helping clarify the presentation and for pointing out and correcting errors in earlier manuscripts.

1 Introduction

The family of Reed-Solomon (RS) codes is a fundamental object of study in algebraic coding theory and theoretical computer science [56]. For an evaluation set S of N elements in a finite field $\mathbb{F}$ and a rate parameter $\rho \in (0,1]$, the code $\mathsf{RS}[\mathbb{F}, S, \rho]$ is the space of functions $f : S \to \mathbb{F}$ that are evaluations of polynomials of degree $d < \rho N$ [56]. The *RS proximity problem* assumes a verifier has oracle access to $f : S \to \mathbb{F}$, and asks that verifier to distinguish, with "large" confidence and "small" query complexity, between the case that f is a codeword of $\mathsf{RS}[\mathbb{F}, S, \rho]$ and the case that f is δ-far in relative Hamming distance from all codewords. This problem has been addressed in several different computational models (surveyed next and summarized in Table 1), and is also the focus of this paper.

RS proximity testing: When no additional data is provided to the verifier, the RS proximity problem is commonly called a *testing* problem, and has been first defined and addressed by Rubinfeld and Sudan in [58] (cf. [32]). In this case, one can see that $d+1$ queries are necessary and sufficient to solve the problem: codewords are accepted by their tester with probability 1 whereas functions that are δ-far from the code are rejected with probability $\geq \delta$. Since no additional information is provided to the verifier in this model, we may say that a prover attempting to convince the verifier that $f \in \mathsf{RS}[\mathbb{F}, S, \rho]$ spends zero computational effort, zero rounds of interaction and produces a proof of length zero.

RS proximity verification – PCPP model: Probabilistically checkable proofs of proximity (PCPP) [21, 30] relax the testing problem to a setting in which the verifier is given oracle access also to an auxiliary proof, called a PCPP and denoted π. This PCPP is produced by the prover, which is given $f \in \mathsf{RS}[\mathbb{F}, S, \rho]$ as input. The time required to produce π is the *prover complexity* and $|\pi|$ is called the *proof length*[1]; similarly, *verifier complexity* is the total time required to generate queries and check query-answers. The techniques used to prove the celebrated PCP Theorem [2, 3] also show that the proximity problem can be solved with constant query complexity and proof length and prover complexity $N^{O(1)}$, or with proof length $N^{1+\epsilon}$ and query complexity $(\log N)^{O(1/\epsilon)}$ [5]. The current state of the art in the PCPP model gives proofs of length $\tilde{O}(N) \stackrel{\triangle}{=} N \cdot \log^{O(1)} N$ with constant query complexity [23, 28] and prover complexity $\tilde{O}(N)$ [16]; verifier complexity is $\mathsf{poly} \log N$ [20, 50].

RS proximity verification – IOPP model: Interactive oracle proofs of proximity (IOPP), formally introduced in [13] and, independently, in [57] (under the name "probabilistically checkable interactive proofs of proximity"), generalize IPs, PCPs and interactive PCPs (IPCP) [42]. As in an IP and IPCP, several rounds of interaction are used in which the prover sends messages $\pi_1, \pi_2, \ldots, \pi_r$ in response to successive verifier messages. As in a PCP and

[1] Typically π is a sequence of elements in $\mathbb{F}$. Therefore, proof length is measured over the alphabet $\mathbb{F}$.

Table 1 Comparison of RS proximity protocols. For concreteness, all results are stated for binary additive RS codes with rate $\rho = 1/8$ evaluated over a sufficiently large set $S, |S| = N$ satisfying $N/|\mathbb{F}| < 0.001$ with proximity parameter $\delta < \delta_0$ (cf. Theorem 2) and soundness at least 0.99δ; i.e., the rejection probability of δ-far words is at least 0.99δ for $\delta < \delta_0$ (in particular, smaller δ leads to smaller soundness). Exponents for the 4th row taken from [16]; the various exponents c in the 5th and 6th row have not been estimated in prior works but are greater than the respective exponents in the 4th row.

	prover comp.	proof length	verifier comp.	query comp.	round comp.
1. Testing [58]	0	0	$\tilde{O}(\rho N)$	ρN	0
2. PCP [2, 3]	$N^{O(1)}$	$N^{O(1)}$	$N^{O(1)}$	$O\left(\frac{1}{\delta}\right)$	1
3. PCP [6, 5]	$N^{1+\epsilon}$	$N^{1+\epsilon}$	$\frac{1}{\delta}\log^{O(1/\epsilon)} N$	$\frac{1}{\delta}\log^{O(1/\epsilon)} N$	1
4. PCPP [23, 21, 16]	$\geq N\log^{1.5} N$	$\geq N\log^{1.5} N$	$\geq \frac{1}{\delta}\log^{5.8} N$	$\frac{1}{\delta}\log^{5.8} N$	1
5. PCPP [28, 50]	$N\log^{c} N$	$N\log^{c} N$	$\frac{1}{\delta}\log^{c} N$	$O\left(\frac{1}{\delta}\right)$	1
6. IOPP [12, 9]	$N\log^{c} N$	$> 4 \cdot N$	$\frac{1}{\delta}\log^{c} N$	$O\left(\frac{1}{\delta}\right)$	$\log\log N$
7. This work	$< 6 \cdot N$	$< \frac{N}{3}$	$\leq 21 \cdot \log N$	$2\log N$	$\frac{\log N}{2}$

IPCP, the verifier is not required to read prover messages in entirety but rather may query them at random locations (in an IPCP, verifier must read the full messages $\pi_2, \ldots$ but may query π_1 randomly); the query complexity is the total number of entries read from f and $\pi_1, \pi_2, \ldots, \pi_{\mathsf{r}}$. The prover is provided with $f \in \mathsf{RS}[\mathbb{F}, S, \rho]$ as input and *prover complexity* is the total time required to produce all (prover) messages[2], while *proof length* is generalized from the PCPP setting to the IOPP setting and defined as $|\pi_1| + \ldots + |\pi_{\mathsf{r}}|$. IOPPs can be used to "replace" PCPP proof composition with more rounds of interaction, and thereby reduce proof length and prover complexity without compromising soundness (see Section 1.3). In particular, the IOPP version of the aforementioned PCPP constructions reduces proof length to $O(N)$ with no change to soundness and/or query complexity [8, 13]. In spite of the shorter proof length, prover complexity in prior works was $\Theta(N\,\mathsf{poly}\log N)$ due to a limitation on the number of proof-composition rounds, explained in Section 2.1.

1.1 Main results

We present a new IOPP for RS codes, called the Fast RS IOPP (FRI) because of its resemblance to the Fast Fourier Transform (FFT) [26]; its analysis relies on the quasi-linear RS-PCPP [23] (see Section 2.1). FRI is the first RS-IOPP to have (i) *strictly linear* arithmetic complexity for the prover with (ii) *strictly logarithmic* arithmetic complexity for the verifier and (iii) *constant* soundness. We start by recalling IOPP systems as described in [12, Section 3.2], after informally summarizing the main complexity parameters of IOPs (introduced and discussed thoroughly in [19]).

1.1.1 IOP

An *Interactive Oracle Proof (IOP)* system S is defined by a pair of interactive randomized algorithms $\mathsf{S} = (\mathsf{P}, \mathsf{V})$, where P denotes the prover and V the verifier. On input x of length N, the number of rounds of interaction is denoted by $\mathsf{r}(N)$ and called the *round complexity* of the system. During a single round the prover sends a message to which the verifier is

[2] Notice that prover complexity does not include the time needed to produce f.

given oracle access, and the verifier responds with a message to the prover. The *proof length*, denoted $\ell(N)$, is the sum of lengths of all messages sent by the prover. The *query complexity* of the protocol, denoted $\mathsf{q}(N)$, is the number of entries read by V from the various prover messages; since the verifier has oracle access to those messages, typically $\mathsf{q}(N) \ll \ell(N)$. (For the FRI system $q(N) = O(\log \ell(N))$). We denote by $\langle \mathsf{P} \leftrightarrow \mathsf{V} \rangle(x)$ the output of V after interacting with P on input x; this output is either accept or reject. An IOP is said to be *transparent* (or have *public randomness*) if all messages sent from the verifier are public random coins and all queries are determined by public coins, which are broadcast to the prover (such protocols are also known as Arthur-Merlin protocols [4]).

1.1.2 IOPP

As its name suggests, an *IOP of proximity (IOPP)* is the natural generalization of a PCP of Proximity (PCPP) to the IOP model. An IOPP for a family of codes[3] $\mathcal{C}$ is a pair (P, V) of randomized algorithms, called *prover* and *verifier*, respectively. Both parties receive as common input a specification of a code $C \in \mathcal{C}$ which we view as a set of functions $C = \{f : S \to \Sigma\}$ for a finite set S and alphabet Σ. We also assume that the verifier has oracle access to a function $f^{(0)} : S \to \Sigma$ and that the prover receives the same function as explicit input. The number of rounds of interaction, or *round complexity*, is denoted by r, *query complexity* is denoted by q.

▶ **Definition 1** (Interactive Oracle Proof of Proximity (IOPP) [12])**.** An r-round Interactive Oracle Proof of Proximity (IOPP) $\mathsf{S} = (\mathsf{P}, \mathsf{V})$ is a $(\mathsf{r}+1)$-round IOP. We say S is an (r-round) IOPP for the error correcting code $C = \{f : S \to \Sigma\}$ with soundness $\mathbf{s}^- : (0, 1] \to [0, 1]$ with respect to distance measure Δ, if the following conditions hold:

- **First message format:** the first prover message, denoted $f^{(0)}$, is a purported codeword of C, i.e., $f^{(0)} : S \to \Sigma$
- **Completeness:** $\Pr\left[\langle \mathsf{P} \leftrightarrow \mathsf{V} \rangle = \mathsf{accept} \mid \Delta\left(f^{(0)}, C\right) = 0\right] = 1$; this means that for every $f^{(0)} \in C$ the protocol terminates in acceptance.
- **Soundness:** For any P^*, $\Pr\left[\langle \mathsf{P}^* \leftrightarrow \mathsf{V} \rangle = \mathsf{reject} \mid \Delta\left(f^{(0)}, C\right) = \delta\right] \geq \mathbf{s}^-(\delta)$

The sum of lengths of all prover messages, except for $f^{(0)}$, is the IOPP *proof length*; the time required to generate all messages except for $f^{(0)}$ is the *prover complexity*. The IOPP *query complexity* is the total number of queries to all messages, including $f^{(0)}$ and the *decision complexity* is the computational complexity (see following remark) required by the verifier to reach its verdict, once the queries and query answers are provided as inputs.

▶ Remark (Computational model for decision complexity). The computational model in which decision complexity is computed is left undefined. A natural default is to use *boolean circuit complexity*. However, later we study families of linear codes in which each IOPP query is answered by a field element. The natural computational model in this case is that of *arithmetic complexity*, i.e., for a linear code C over a finite field $\mathbb{F}$, it is the number of arithmetic operations over $\mathbb{F}$ made by the verifier to reach its decision.

1.1.3 Main Theorem

The finite field of size q is denoted here by $\mathbb{F}_q$; when q is clear from context we omit it. A field is called *binary* if $q = 2^m, m \in \mathbb{N}$. A subset S of a binary field is an *additive coset* if it is a coset of a subgroup of the additive group $\mathbb{F}^+$, i.e., if S is an additive shift of an

[3] The definition of an IOPP can be generalized to arbitrary languages; we study an IOPP for a specific family of codes so prefer to limit the scope of our definition accordingly.

$\mathbb{F}_2$-linear space contained in $\mathbb{F}_q$. The *binary additive RS code family* is the collection of codes $\mathsf{RS}[\mathbb{F}, S, \rho]$ where $\mathbb{F}$ is a binary field and S an additive coset. This family of codes is one for which quasilinear PCPP were defined in [23], and our main theorem is stated for it (see Table 1).

▶ **Theorem 2** (Main – FRI properties)**.** *The binary additive RS code family of rate* $\rho = 2^{-\mathcal{R}}, \mathcal{R} \geq 2, \mathcal{R} \in \mathbb{N}$ *has an IOPP (*FRI*) with the following properties, where* $N = |S|$ *denotes block-length (which equals the prover-side input length for a fixed* $\mathsf{RS}[\mathbb{F}, S, \rho]$ *code) and* $\rho N > 16$*:*

- ***Prover*** *Complexity is less than* $6N$ *arithmetic operations in* $\mathbb{F}$*; proof length is less than* $N/3$ *field elements and round complexity is at most* $\frac{\log N}{2}$*;*
- ***Verifier complexity*** *Query complexity is* $2 \log N$*; the verifier decision is computed using at most* $21 \log N$ *arithmetic operations over* $\mathbb{F}$*;*
- ***Soundness:*** *There exists* $\delta_0 \geq \frac{1}{4}(1 - 3\rho) - \frac{1}{\sqrt{N}}$ *such that every* f *that is* δ*-far in relative Hamming distance from the code, is rejected with probability at least* $\min\{\delta, \delta_0\} - \frac{3N}{|\mathbb{F}|}$*;*
- ***Parallelization:*** *Each prover-message can be computed in* $O(1)$ *time on a Parallel Random Access Machine (PRAM) with common read and exclusive write (CREW), assuming a single* $\mathbb{F}$ *arithmetic operation takes unit time.*

▶ Remark (Space complexity). Given the ith prover message as input, each symbol of the $(i+1)$th prover message can be computed with space complexity $O(\log |\mathbb{F}|)$, i.e., the space required to hold a constant number of field elements.

This follows immediately from the fact that each prover message is computed in $O(1)$ arithmetic operations on a parallel machine.

Generalizing Theorem 2 to arbitrary rate $\rho \in (0, 1]$ can be done as described in [23, Proposition 6.13] (cf. remark 6.2 there); this leads to slightly larger constants in the prover and verifier complexity. For practical applications like ZK-IOPs [14, 12], rates of the form stated in the theorem above suffice.

▶ Remark (FRI for "smooth codes"). We call a multiplicative group $H \subset \mathbb{F}_q$ *smooth* if its order ($|H|$) is 2^k for $k \in \mathbb{N}$. The family of *smooth* RS codes of rate ρ is the set of $\mathsf{RS}[\mathbb{F}_q, H, \rho]$ codes in which H is a smooth multiplicative group. Theorem 2 holds *also* with respect to the family of smooth RS codes, with somewhat smaller constants than 6 and 21 for the prover and verifier arithmetic complexity (see full version of this paper [10]); see Section 2.1 for a high-level overview of the smooth case and full version of this paper [10] for more details on modifying the protocol to this case. The protocol can be further generalized to groups of order c^k for constant c (perhaps with different arithmetic complexity constants), details omitted.

The soundness bound of Theorem 2 is nearly tight for $\delta \leq \delta_0$. We conjecture that a similar bound holds for all δ. See full version of this paper [10] for a more detailed version of the conjecture that implies it, and a discussion of Equation (1)

▶ **Conjecture 3.** *The soundness limit* δ_0 *of Theorem 2 approaches* $1 - \rho$*. Specifically, for all* $\delta \leq 1 - \rho$*, the rejection probability of any* f *that is* δ*-far from the RS-code of rate* ρ *and block-length* N *over* $\mathbb{F}$*, is at least*

$$\delta - \frac{2 \log N}{\sqrt{|\mathbb{F}|}}. \tag{1}$$

1.2 Applications to transparent zero knowledge implementations

Prover-efficient IOPPs of the kind presented here are crucially needed to facilitate *practical* ZK argument systems that are (i) *transparent* (public randomness), (ii) *universal* – apply to any computation – and (iii) *(doubly) scalable* – have quasi-linear proving time *and* poly-logarithmic verification time, simultaneously. In a follow-up paper we use the FRI protocol (among other things) to realize in code the first ZK system (called a ZK-STARK there) that achieves the three properties listed above [11]. The concrete efficiency of that protocol, which relies to a large degree on the efficiency of the FRI protocol presented here, allows one to construct ZK arguments of knowledge (ZK-ARKs) for computational statements, where verifying the computational integrity of the statement using the ZK-STARK verifier is stricly faster than naïve verification via re-exectution, and the communication compelxity is strictly smaller than the size of the non-deterministic witness supporting the claim. The ZK-STARK prover is $\approx 50\times$ faster than the previous state-of-the-art transparent system, code-named SCI [7] (that system does not have ZK), and $\approx 10\times$ faster than state-of-the-art ZK-SNARKs [17] (which are not transparent); see [11, Figure 5] for details.

In the remainder of this section we explain, briefly, how our system could be incorporated in a larger practical ZK system (like the ZK-STARK mentioned earlier). In Section 1.3 we discuss the range of block-lengths that are relevant in applications, and the resulting communication complexity arising from their use.

The seminal works of Babai et al. [6, 5] showed that verifying the correctness of an arbitrary nondeterministic computation running for $T(N)$ steps can be achieved by a verifier running in time $\mathsf{poly}(N, \log T(N))$ in the PCP model. Kilian's construction transforms such PCPs into a 4-round ZK argument in which the total communication complexity and verifier running time are bounded by $\mathsf{poly}\log T(N)$ [43] (cf. [44, 40, 41]), assuming a family of collision-resistant hash functions. Micali further compressed this system into a non-interactive computationally sound (CS) proof system, assuming both prover and verifier share access to the same random function [48]; this is typically realized in practice using a hash function like SHA2 and relying on the Fiat-Shamir heuristic [31]. No implementation of these marvelous techniques has appeared during the quarter century that has passed since they were first published. This is explained, in part, by concerns about the efficiency of these constructions for concrete programs and run-times. Among the numerous components involved in building these systems, a significant computational bottleneck is that of computing solutions to the Reed-Muller (RM) proximity problem, also known as "low degree testing" of multivariate polynomials.

Quasilinear PCPs based on RS codes have prover complexity that is asymptotically more efficient than RM codes which lead to PCPs with super-quasi-linear length, and a number of works have explored the concrete efficiency of these RS-based protocols [16, 8]. Recently, Ben-Sasson et al. suggested an IOP with perfect zero knowledge (PZK) for NP [14], later extended to NEXP [12], in which prover complexity is quasilinear and verifier complexity is $\mathsf{poly}(N, \log T(N))$; this PZK-IOP can be compiled, using Kilian's technique, into an interactive ZK argument with succinct[4] communication complexity, or, using Micali's technique (cf. [61]), into a *non-interactive random oracle proof (NIROP)* as defined in [19]. In light of this, the practicality of Kilian- and Micali-type ZK argument systems with polylogarithmic verifiers should be reconsidered.

To add motivation, a number of interesting practical succinct argument systems (with and without zero-knowledge) have been reported recently (see [62] for an excellent updated survey

[4] Here, as in past works, "succinct" is synonymous to "polylogarithmic".

of the subject and [7] for a comparison of PCP/IOP-based solutions to other approaches). A particular system based on the *quadratic span programs* (QSP) of Gennaro et al. [33] (cf. [17]) has been used by Ben-Sasson et al. to build a decentralized anonymous payment (DAP) system called "Zerocash" [15], later deployed as a practical commercial crypto-currency called "Zcash" [52, 38]. However, the QSP based ZK system used in Zerocash/Zcash, called a "preprocessing SNARK" [24], requires a setup phase that involves *private* randomness; additionally, it relies on rather strong cryptographic "knowledge of exponent" assumptions, and quantum computers can create pseudo-proofs of falsities in polynomial time for such systems [60] (cf. [54]). In contrast, the aforementioned succinct interactive and non-interactive (NIROP) systems based on quasilinear PZK-IOPs require only public randomness for their setup, and the only cryptographic assumption required to realize them[5] is the existence of a family of collision resistant hash functions [43], in particular, they are not known to be breakable by quantum computers in polynomial time. Therefore, there is great interest in understanding whether succinct (interactive and non-interactive) ZK argument systems which require only public randomness (and resistant to known polynomial time quantum algorithms) can be *practically* built and used, say, by Zcash. Ben-Sasson et al. [7] describe such an implemented system, called "succinct computational integrity (SCI)" which is not ZK and has comparatively large communication complexity[6]. As mentioned above, the RS proximity solution described in Theorem 2 is already used within an implemented ZK system [11].

1.3 Concrete degree, communication, and round complexity

In this section we *briefly* discuss the "size" of RS codes that would be needed for various practical applications and the effect of logarithmic round complexity on security. Due to space limitations, and because the focus of this paper is *theoretical* (within the information theoretic IOP model), we omit implementation details and point the interested reader to full version of this paper [10]; cf. [7, 14].

The message length of RS codes of degree $d = \rho \cdot N - 1$ is precisely d, so we start by recounting the range of degrees (message sizes) that seem practically relevant. Later we calculate the communication complexity arising from using the FRI protocol to argue proximity to codes of practically relevant block-lengths, and end by discussing the practical implications of an IOPP with $\log d$ rounds. Throughout this section $\rho = 1/8$ ($N = 8 \cdot d$) because this setting is used in prior [7] and future [11] works.

1.3.1 RS block-length of systems realized in code

The recently realized IOP-based argument system called SCI ("Scalable Computational Integrity") reduces computational statements, like *"the output of program P on input x equals y after T steps"* to a *pair* of RS-proximity testing problems. SCI uses an IOP version of the quasilinear PCP of [23], which could be replaced with FRI. Programs bench-marked by SCI were executed on a simple MIPS-like virtual machine called *TinyRAM* [18]. Generally speaking, RS degree increases in size with the number of TinyRAM machine cycles T.

[5] To reach a (non-interactive) computationally sound (CS) proof [49], the "random oracle" is assumed, and realized in practice by relying on the Fiat-Shamir heuristic. In particular, this approach as well is not known to be breakable by quantum computers in polynomial time.

[6] Communication complexity in SCI is on the order of tens of megabytes long, compared with QSP based zk-SNARKs that are shorter than 300 Bytes.

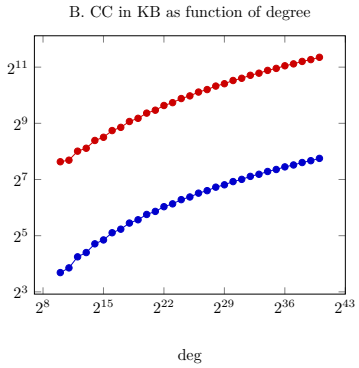


Figure 1 A. Degree of RS code arising from the *exhaustive subset sum* program [7, Appendix C], as a function of the number of TinyRAM machine cycles. B. Communication complexity (CC) as a function of degree, using $\lambda = 160$ bits, field size 2^{64}, soundness error $\epsilon = 2^{-80}$, and maximal proximity parameter $\delta = 1 - \rho$. The higher (red) graph corresponds to proven soundness (see full version of this paper [10]) and the lower (blue) corresponds to conjectured soundness (Conjecture 3). Both plots use code rate $\rho = 1/8$.

Figure 1.A plots the degree d as a function of T for a specific simple program, showing that $d \approx T \cdot 2^{21}$.

For crypto-currency applications requiring zero knowledge, block-length will be dominated by the type of cryptographic primitives required, and the number of times they are invoked within a computational statement. For instance, *ZK contingent payments* [47] require a single hash, and Zerocash's *Pour circuit* [15] uses 64 hash invocations, leading in that work to RS codewords (over a prime field) with degree (=number of gates) approximately 2^{22}. Our new work in progress shows that a single hash invocation requires RS block-length between $2^{12} = 4096$ (for a Davies–Meyer hash based on AES128) to 2^{19} (for SHA2), meaning that degrees in the range $d \in [2^{12}, 2^{26}]$ are relevant for existing crypto-currency (ZK) applications [11].

1.3.2 Estimated communication complexity and argument length

The practical realization of interactive proof systems (see Section 1.2) into interactive argument systems [43] and CS proofs [49] can be extended to the IOPP model, in which multiple rounds of interaction are used [19]. Using Kilian's scheme [43], during the ith round the prover sends the root $\mathsf{root}^{(i)}$ of a Merkle hash tree $\mathsf{Tree}^{(i)}$ whose leaves are labeled by entries of $f^{(i)}$, and the verifier replies with randomness. Using Micali's scheme [49], the (non-interactive) prover queries the random oracle with $\mathsf{root}^{(i)}$ to "simulate" the verifier's ith message. When verifier queries to $f^{(i)}$ are answered by the prover, each answer is accompanied by an *authentication path* (AP) that shows the query answer is consistent with $\mathsf{root}^{(i)}$. Let $\mathsf{CC}_{\delta,\epsilon}(N)$ denote the prover-side communication complexity (in bits) of an argument/CS proof realized by applying the Kilian/Micali scheme to FRI, where δ is the proximity parameter and ϵ is the error bound, i.e., words that are δ-far from the RS code are rejected with probability $< \epsilon$. Then

$$\mathsf{CC}_{\delta,\epsilon}(N) = \mathsf{q}_{\delta,\epsilon} \cdot \log |\mathbb{F}| + \mathsf{AP}_{\delta,\epsilon} \cdot \lambda \tag{2}$$

where $\mathsf{q}_{\delta,\epsilon}$ denotes total query complexity in the IOP model to reach soundness $\geq 1 - \epsilon$ for proximity parameter δ, $\mathsf{AP}_{\delta,\epsilon}$ is the number of nodes in the sub-forest of the Merkle trees $\mathsf{Tree}^{(0)}, \ldots, \mathsf{Tree}^{(\mathsf{r})}$ induced by all authentication paths, and λ is the number of output bits of

the hash function used to construct the Merkle trees. In our preliminary results [11] we use $\lambda = 160$, $\epsilon = 2^{-80}$, $|\mathbb{F}| = 2^{64}$ and $\rho = 1/8$. Figure 1.B plots the communication complexity for this setting under the proven soundness of Theorem 2 and the (better) soundness of Conjecture 3. In both cases we use maximally large distance $\delta = 1 - \rho = 7/8$ to show the concrete difference in communication complexity between the proven and conjectured soundness. This plot also motivates the quest for improving the soundness analysis of Theorem 2.

1.3.3 Round complexity considerations

Assuming that a crypto-currency block-chain serves as a time-stamping service for public messages and a public beacon of randomness, one may use block-chains to simulate verifier messages. Several block-chains (including Zcash) generate blocks every 2.5 minutes, which means that a FRI proof for $d = 2^k$ will take roughly $k \cdot \frac{5}{4}$ minutes to complete, or less than 1 hour[7] for $d < 2^{40}$.

For fixed d, the round complexity stated in Theorem 2 is $\frac{1}{2}\log d$, but the more refined version (see [10]) gives a trade-off between query (q) and round (r) complexity, of the form $\mathsf{r} = \log d / \log \mathsf{q}$, allowing further reduction in round complexity in exchange for larger communication complexity.

Finally, the Random Oracle model used by Micali to "compress" interactive argument systems (like Kilian's) into CS proofs applies equally to multi-round IOPs like FRI, with negligible impact on argument length; see [19, Remark 1.6] for a detailed discussion. Practically speaking, those who treat hash functions like SHA2 as realizations of the RO model (a position taken by Bitcoin and other crypto-currency miners), might feel comfortable compiling IOP protocols like FRI into succinct non-interactive arguments, as described in [19].

1.4 Related works

High-rate LTCs Locally testable codes (LTCs) are error correcting codes for which – by definition – prover complexity and proof length equal 0 (as stated for the case of RS codes by Rubinfeld and Sudan [58]); in other words, when focusing solely on prover complexity, LTCs offer an optimal solution (zero complexity). Nevertheless, as discussed in Section 1.2, the specific question of small prover complexity for RS codes is highly relevant because of the its applications to practical ZK-IOPs.

Classical "direct" constructions of LTCs, such as the Hadamard code studied by Blum, Luby and Rubinfeld [25] and the $\log N$-variate RM codes used in early PCP constructions [1, 5] have sub-constant rate, thus lead to long proofs and large PCP prover complexity.

More recently, there has been remarkable progress on constructing locally testable codes (LTCs) with small query complexity and large soundness. Kopparty et al. obtained such codes with rate approaching 1 [45] and Gopi et al. presented LTCs that reach the Gilbert Varshamov bound [36]. These LTCs have super-polylogarithmic query complexity. Additionally, in contrast to RS codes, we are not aware of PCP constructions with similar parameters nor do we know how to convert these LTCs into PCPs.

PCPs and IOPs: A number of recent works have considered PCP constructions with small proof length and query complexity. In addition to the aforementioned works on quasi-linear PCPs, Moshkovitz and Raz constructed PCPs with optimally small query complexity

[7] Compare this with Bitcoin's "best practice" of waiting 1 hour for confirmations, or 3 days required to clear standard checks.

(measured in bits) and proofs of length $N^{1+o(1)}$ [51], where N denotes the length of the NP statement (like a 3CNF) for which the PCP is constructed, achieving better soundness than Hr astad's result [37]. A different line of works attempts to optimize the *bit-length* of PCP proofs; the state of the art, due to Ben-Sasson et al., achieves PCPs of bit-length $O(N)$ and query complexity N^{ϵ} [22]. In the IOP model, which generalizes PCPs by allowing more rounds of interaction, Ben-Sasson et al. presented a 2-round IOP with bit-length $O(N)$, constant query complexity (measured in bits) and constant soundness [13]. (Prover arithmetic complexity in all of these systems is super-linear.)

Soundness amplification: A number of results in the PCP literature have suggested techniques for improving soundness of general PCP constructions, including the parallel repetition theorem of Raz [55], the gap amplification technique of Dinur [28] and direct-product testing, introduced by Goldreich and Safra [34] (cf. [29, 39]). These techniques lead to excellent soundness bounds with small query complexity. The concrete prover complexity of PCPs and PCPPs associated with these methods has not been studied in prior works but prover complexity is at least super-linear, and often polynomially large.

Doubly-efficient "proofs for muggles": A recent line of works, initiated by Goldwasser, Kalai and Rothblum [35], revisits the IP model which is equivalent to PSPACE [46, 59], focusing on *doubly efficient* systems in which the prover runs in polynomial time (as opposed to polynomial space, as in the aforementioned results) and verifier runs in nearly linear time. The state of the art along this line is due to Reingold et al. [57], they construct doubly-efficient IP protocols with a constant number of rounds for a family of languages in P. Prover complexity in this line of works is at least super-linear, and typically polynomially large and verifier complexity is super-polylogarithmic, and often super-linear as well (cf. [27, 57]).

2 Overview of the FRI IOPP and its soundness

In this section we consider the task of building an IOPP for a "smooth" RS code (defined below). We start in Section 2.1 by considering the *completeness* case, where we describe the interaction between the verifier and an *honest prover* attempting to prove membership in the RS code of a valid codeword $f^{(0)}$. The IOPP protocol is explained in similarity to the Inverse Fast Fourier Transform (IFFT) [26]. Then, in Section 2.2, we consider the *soundness* case, where we assume $f^{(0)}$ is far in relative Hamming distance from the code and need to prove lower bounds on the verifier's rejection probability. Soundness analysis is the most challenging aspect of our work (as it is for all prior PCPP/IOPP works). Our analysis uses the soundness analysis of the quasilinear RS-PCPP [23] for the case of "large" Hamming distance (beyond the unique decoding radius of the code), and presents a novel, tighter, analysis for "small" Hamming distance (below that radius).

2.1 FRI overview and similarity to the Fast Fourier Transform (FFT)

We start by describing the protocol in similarity to the IFFT algorithm; that algorithm is also related to the quasi-linear PCPP for RS codes of [23], and towards the end of this section we explain the connection between FRI and that quasi-linear PCPP.

Let $\omega^{(0)}$ generate a *smooth* multiplicative group of order $N = 2^n$ (see Remark 1.1.3), denoted $L^{(0)}$, that is contained in a field $\mathbb{F}$; in signal processing applications $\omega^{(0)}$ is a complex root of unity of order 2^n and $\mathbb{F}$ is the field of complex numbers (we shall use a different setting). Assume the prover claims that $f^{(0)} : L^{(0)} \to \mathbb{F}$ is a member of $\mathsf{RS}[\mathbb{F}, L^{(0)}, \rho]$, i.e., $f^{(0)}$ is the evaluation of an unknown polynomial $P^{(0)}(X) \in \mathbb{F}[X], \deg(P) < \rho 2^n$; for simplicity we assume $\rho = 2^{-\mathcal{R}}$ and $\mathcal{R}$ is a positive integer. The task of the verifier is to distinguish between

low-degreeness ($f^{(0)} \equiv P^{(0)}$ for some low degree $P^{(0)}$) and cases where $f^{(0)}$ is far from all polynomials of degree $< \rho 2^n$. Recalling the IFFT, if $f^{(0)} \equiv P^{(0)}$ there exist polynomials $P_0^{(1)}, P_1^{(1)} \in \mathbb{F}[Y]$ such that $\max\left\{\deg\left(P_0^{(1)}\right), \deg\left(P_1^{(1)}\right)\right\} < \frac{1}{2}\rho 2^n$ and

$$\forall x \in L^{(0)} \;\; f^{(0)}(x) = P^{(0)}(x) = P_0^{(1)}(x^2) + x \cdot P_1^{(1)}(x^2),$$

or, letting $Q^{(1)}(X,Y) \stackrel{\triangle}{=} P_0^{(1)}(Y) + X \cdot P_1^{(1)}(Y)$ and defining $q^{(0)}(X) \stackrel{\triangle}{=} X^2$, we have

$$P^{(0)}(X) \equiv Q^{(1)}(X, Y) \mod Y - q^{(0)}(X) \tag{3}$$

where $\deg_X\left(Q^{(1)}\right) < 2$ and $\deg_Y\left(Q^{(1)}\right) < \frac{1}{2}\rho 2^n$. The map $x \mapsto q^{(0)}(x)$ is 2-to-1 on $L^{(0)}$ because $q^{(0)}(x) = q^{(0)}(-x)$, and the output of this map is the multiplicative group generated by $\omega^{(1)} = \left(\omega^{(0)}\right)^2$, this group has order 2^{n-1}, denote it by $L^{(1)}$. Moreover, for every $x^{(0)} \in \mathbb{F}$ and $y \in L^{(1)}$, the value of $Q^{(1)}\left(x^{(0)}, y\right)$ can be computed by querying two entries of $f^{(0)}$ because $\deg_X\left(Q^{(1)}\right) < 2$ (the two entries are the two roots of the polynomial $y - q^{(0)}(X)$).

Our verifier thus samples $x^{(0)} \in \mathbb{F}$ uniformly at random and requests the prover to send as its first oracle a function $f^{(1)} : L^{(1)} \to \mathbb{F}$ that is supposedly the evaluation of $Q^{(1)}\left(x^{(0)}, Y\right)$ on $L^{(1)}$. Assuming $f^{(0)} \in \mathsf{RS}[\mathbb{F}, L^{(0)}, \rho]$, the discussion above shows that $f^{(1)} \in \mathsf{RS}[\mathbb{F}, L^{(1)}, \rho]$. Notice that there exists a 3-query test for the consistency of $f^{(0)}$ and $f^{(1)}$, we call it the *round consistency test*:

1. sample a pair of distinct elements $s_0, s_1 \in L^{(0)}$ such that $s_0^2 = s_1^2 = y$; in other words, sample a uniform $y \in L^{(1)}$ and let s_0, s_1 be the two roots of the polynomial $y - X^2$;
2. query $f^{(0)}(s_0), f^{(0)}(s_1)$ and $f^{(1)}(y)$, denote the query answers by α_0, α_1 and β, respectively;
3. interpolate the "line" through (s_0, α_0) and (s_1, α_1), i.e., find the polynomial $p(X)$ of degree at most 1 that satisfies $p(s_0) = \alpha_0$ and $p(s_1) = \alpha_1$; notice p is unique and well-defined because $s_0 \neq s_1$;
4. accept if and only if $p\left(x^{(0)}\right) = \beta$ and otherwise reject;

Tallying the costs of the first round, the verifier sends a single field element ($x^{(0)}$) and the prover responds with a message (oracle) $f^{(1)} : L^{(1)} \to \mathbb{F}$ evaluated on a domain that is half the size of $L^{(0)}$; testing the consistency of $f^{(0)}$ and $f^{(1)}$ requires three field elements per test (repeating the test boosts soundness). We thus reduced a single proximity problem of size 2^n and rate ρ to a single analogous problem of size 2^{n-1} and same rate. Repeating the process for $\mathsf{r} = n - \mathcal{R}$ rounds leads to a function $f^{(\mathsf{r})}$ that is supposedly of constant degree and evaluated over a domain of constant size $2^{\mathcal{R}}$, so at this point the prover sends the single constant that describes the function, and the verifier uses it as $f^{(\mathsf{r})}$ in the last round consistency test, the one that tests consistency of $f^{(\mathsf{r}-1)}$ and $f^{(\mathsf{r})}$.

Applying inductive analysis to all r rounds, if $f^{(0)} \in \mathsf{RS}[\mathbb{F}, L^{(0)}, \rho]$ (and the prover is honest) then all r round consistency tests pass with probability 1 and $f^{(\mathsf{r})}$ is indeed a constant function. In other words, the protocol we described has perfect completeness.

▶ Remark (FRI as a "biased" version of quasi-linear RS-PCPP). The quasi-linear PCPP of [23] is quite similar to FRI, including the degree-reduction (from $P^{(0)}$ to $P^{(1)}$) obtained by requesting the prover to evaluate a bivariate polynomial $Q(X,Y)$ on a collection of axis-parallel lines. There are two main differences between FRI and that PCPP:

1. the quasilinear PCPP is non-interactive, and thus the prover evaluates $Q^{(1)}(X,Y)$ on a large subset of $\mathbb{F} \times \mathbb{F}$, whereas the FRI protocol uses interaction to reduce proving time, by requesting the prover to apply recursion only to the axis-parallel lines selected by the verifier.

2. the polynomial $q^{(0)}(X)$ used in [23] has degree $\approx \sqrt{\deg\left(P^{(0)}\right)}$ and thus $Q^{(1)}(X,Y)$ has degree $\approx \sqrt{\deg\left(P^{(0)}\right)}$ in each of its variables. In contrast, the polynomial $q^{(0)}$ used by FRI has *constant* degree, and so the degrees of $Q^{(1)}$ are very biased (constant degree in X vs. $\deg\left(P^{(0)}\right)/2$ in Y). This leads to larger recursion depth for FRI but also avoids the necessity to apply recursive low-degree testing to each of the axes (the X-axis) because of its constant degree.

2.1.1 Differences between informal and actual protocol

The differences between the informal and formal protocols are mostly technical; we list them now. The field $\mathbb{F}$ is finite and *binary*, i.e., of characteristic 2; nevertheless the construction and analysis can be immediately applied to RS codes evaluated over smooth multiplicative groups (of order 2^n), as explained informally above (cf. Remark 1.1.3). In binary fields, the natural evaluation domains (like $L^{(0)}, L^{(1)}$ above) are cosets of *additive* groups (not multiplicative ones), i.e, $L^{(i)}$ is an affine shift of a linear space over $\mathbb{F}_2$. The map $q^{(0)}(X) = X^2$ is *not* 2-to-1 on $L^{(0)}$ (in binary fields it is a 1-to-1 map, a Frobenius automorphism of $\mathbb{F}$ over $\mathbb{F}_2$) so we use a different polynomial $q^{(0)}(X)$ that is many-to-one on $L^{(0)}$ and such that the set $L^{(1)} = \left\{y = q^{(0)}(x) \mid x \in L^{(0)}\right\}$ is a coset of an *additive* group, like $L^{(0)}$, but of smaller size ($|L^{(1)}| \ll |L^{(0)}|$); the polynomial $q^{(0)}$ is known as an *affine subspace* polynomial, belonging to the class of *linearized* polynomials. We use $q^{(0)}$ of degree 4 instead of 2 because this reduces the number of rounds from n to $n/2$ with no increase in total query complexity; notice that a similar reduction could be applied in the multiplicative setting by using $q^{(0)} = X^4$ (but we preferred simplicity to efficiency in the informal exposition above). Finally, the actual protocol performs all queries only after the prover has sent all of $f^{(1)}, \ldots, f^{(\mathsf{r})}$. Thus, we construct a protocol with two phases. The first phase, called the COMMIT phase, involves r *rounds*. At the beginning of the ith round the prover has sent oracles $f^{(0)}, \ldots, f^{(i-1)}$, and during this ($i$th) round the verifier samples and sends $x^{(i)}$ and the prover responds by sending the next oracle $f^{(i)}$. During the second phase, called the QUERY phase, the verifier applies the *round consistency test* to all r rounds. To save query complexity *and boost soundness*, the query made to $L^{(i)}$ is used to test *both* consistency of $f^{(i-1)}$ vs. $f^{(i)}$ *and* consistency of $f^{(i)}$ vs. $f^{(i+1)}$.

2.2 Soundness analysis – overview

Proof composition is a technique introduced by Arora and Safra [3] in the context of PCPs, adapted to PCPPs in [21, 30] and optimized for the special case of the RS code in [23]. Informally, it reduces proximity testing problems over a large domain to similar proximity testing problems over significantly smaller domains. The process reducing $f^{(0)}$ to $f^{(1)}$ above is a special case of proof composition, and each invocation of it incurs two costs on behalf of the verifier. The first is the *query complexity* needed to check consistency of $f^{(0)}$ and $f^{(1)}$ (the "round consistency test") and the second is the reduction in *distance*, which affects the *soundness* of the protocol. Assuming $f^{(0)}$ is $\delta^{(0)}$-far from all codewords in relative Hamming distance, for proof composition to work one should prove that with high probability $f^{(1)}$ is $\delta^{(1)}$-far from all codewords where $\delta^{(1)}$ depends on $\delta^{(0)}$; larger values of $\delta^{(1)}$ imply higher (better) soundness and smaller communication complexity. A benefit of the FRI protocol is that with high probability $\delta^{(1)} \geq (1 - o(1))\delta^{(0)}$, i.e., the reduction in distance in our protocol is *negligible*. In contrast, prior RS proximity PCPP and IOPP solutions follow the construction and analysis of [23] which in turn is based on the bivariate testing Theorem of

Polischuk and Spielman [53] and incur a *constant multiplicative loss* in distance per round of proof composition ($\delta^{(1)} \leq \delta^{(0)}/2$). This loss limited the number of proof composition rounds to $\leq \log N$ and thus required replacing $q^{(0)}(X) = X^2$ with a higher degree polynomial, like $q^{(0)}(X) = X^{2^{n/2}}$. The higher degree of $q^{(0)}$ results in $Q^{(1)}(X, Y)$ having *balanced* X- and Y-degrees, namely

$$\deg_X \left(Q^{(1)}\right) \approx \deg_Y \left(Q^{(1)}\right) \approx 2^{n/2}.$$

Moving to $q^{(0)}(X)$ of *constant* degree as in FRI gives a *biased RS-IOPP* (because $\deg_X(Q^{(1)}) \ll \deg_Y(Q^{(1)})$). The main benefit of this bias is that one side of the recursive process (that of X) terminates immediately and consequently *removes* the constant multiplicative soundness loss incurred in prior works, replacing it with a negligible additive loss. More to the point, we show that for $\delta^{(0)}$ less than the unique decoding radius of the code ($\delta^{(0)} < (1-\rho)/2$), with high probability (namely, $1 - \frac{O(1)}{|\mathbb{F}|}$) the sum of (i) the round consistency error and (ii) the "new" distance $\delta^{(1)}$ is at least as large as the "old" distance $\delta^{(0)}$. This statement is relatively straightforward to prove in case the prover is *honest*, i.e., when $f^{(1)}(y) = Q^{(1)}\left(x^{(0)}, y\right)$ for all $y \in L^{(1)}$ (in this case there is no round consistency error). The challenging part of the proof is to show this also holds for non-honest provers and arbitrary $f^{(1)}$; see full version of this paper [10] for more details.

References

1. Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 14–23, 1992.
2. Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998. Preliminary version in FOCS '92.
3. Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998. Preliminary version in FOCS '92.
4. László Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, STOC '85, pages 421–429, 1985.
5. László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, STOC '91, pages 21–32, 1991.
6. László Babai, Lance Fortnow, and Carsten Lund. Nondeterministic exponential time has two-prover interactive protocols. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, FOCS '90, pages 16–25, 1990.
7. Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. Computational integrity with a public random string from quasi-linear PCPs. *IACR Cryptology ePrint Archive*, 2016:646, 2016. URL: `http://eprint.iacr.org/2016/646`.
8. Eli Ben-Sasson, Iddo Bentov, Ariel Gabizon, and Michael Riabzev. A security analysis of probabilistically checkable proofs. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:149, 2016. URL: `http://eccc.hpi-web.de/report/2016/149`.
9. Eli Ben-Sasson, Iddo Bentov, Ariel Gabizon, and Michael Riabzev. A security analysis of probabilistically checkable proofs. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:149, 2016. URL: `http://eccc.hpi-web.de/report/2016/149`.

10 Eli Ben-Sasson, Iddo Bentov, Ynon Horesh, and Michael Riabzev. Fast Reed-Solomon interactive oracle proofs of proximity (2nd revision). *Electronic Colloquium on Computational Complexity (ECCC)*, 24:134, 2017. URL: `https://eccc.weizmann.ac.il/report/2017/134`.

11 Eli Ben-Sasson, Iddo Bentov, Ynon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity, 2017. Unpublished manuscript.

12 Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. On probabilistic checking in perfect zero knowledge. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:156, 2016. URL: `http://eccc.hpi-web.de/report/2016/156`.

13 Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. Short interactive oracle proofs with constant query complexity, via composition and sumcheck. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:46, 2016.

14 Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, and Madars Virza. Quasilinear-size zero knowledge from linear-algebraic PCPs. In *Proceedings of the 13th Theory of Cryptography Conference*, TCC '16, pages 33–64, 2016.

15 Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, 2014.

16 Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *Proceedings of the 45th ACM Symposium on the Theory of Computing*, STOC '13, pages 585–594, 2013.

17 Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Proceedings of the 33rd Annual International Cryptology Conference*, CRYPTO '13, pages 90–108, 2013.

18 Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Tinyram architecture specification v2. 00, 2013. Technical report, SCIPR Lab, 2013. URL: `http://www.scipr-lab.org/doc/TinyRAM-spec-0.991.pdf`.

19 Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. *Interactive Oracle Proofs*, pages 31–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. `doi:10.1007/978-3-662-53644-5_2`.

20 Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Short PCPs verifiable in polylogarithmic time. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, CCC '05, pages 120–134, 2005.

21 Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM Journal on Computing*, 36(4):889–974, 2006.

22 Eli Ben-Sasson, Yohay Kaplan, Swastik Kopparty, Or Meir, and Henning Stichtenoth. Constant rate pcps for circuit-sat with sublinear query complexity. *J. ACM*, 63(4):32:1–32:57, 2016. `doi:10.1145/2901294`.

23 Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM Journal on Computing*, 38(2):551–607, 2008. Preliminary version appeared in STOC '05.

24 Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 326–349, 2012.

25 Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993. `doi:10.1016/0022-0000(93)90044-W`.

26 James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

27 Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 4th Symposium on Innovations in Theoretical Computer Science*, ITCS '12, pages 90–112, 2012.

28 Irit Dinur. The PCP theorem by gap amplification. *Journal of the ACM*, 54(3):12, 2007.

29 Irit Dinur and Elazar Goldenberg. Locally testing direct product in the low error range. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 613–622, 2008. doi:10.1109/FOCS.2008.26.

30 Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '04, pages 155–164, 2004.

31 Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings of the 6th Annual International Cryptology Conference*, CRYPTO '86, pages 186–194, 1986.

32 K. Friedl and M. Sudan. Some improvements to total degree tests. In *Proceedings Third Israel Symposium on the Theory of Computing and Systems*, pages 190–198, Jan 1995. doi:10.1109/ISTCS.1995.377032.

33 Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Proceedings of the 32nd Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT '13, pages 626–645, 2013.

34 Oded Goldreich and Shmuel Safra. A combinatorial consistency lemma with application to proving the pcp theorem. *SIAM Journal on Computing*, 29(4):1132–1154, 2000. doi:10.1137/S0097539797315744.

35 Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for Muggles. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, STOC '08, pages 113–122, 2008.

36 Sivakanth Gopi, Swastik Kopparty, Rafael Mendes de Oliveira, Noga Ron-Zewi, and Shubhangi Saraf. Locally testable and locally correctable codes approaching the gilbert-varshamov bound. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2073–2091, 2017. doi:10.1137/1.9781611974782.135.

37 Johan Hr astad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.

38 Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox, March 2017. URL: https://github.com/zcash/zips/blob/master/protocol/protocol.pdf.

39 Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. New direct-product testers and 2-query pcps. *SIAM Journal on Computing*, 41(6):1722–1768, 2012. doi:10.1137/09077299X.

40 Yuval Ishai, Mohammad Mahmoody, Amit Sahai, and David Xiao. On zero-knowledge PCPs: Limitations, simplifications, and applications, 2015. Available at http://www.cs.virginia.edu/~mohammad/files/papers/ZKPCPs-Full.pdf.

41 Yuval Ishai and Mor Weiss. Probabilistically checkable proofs of proximity with zero-knowledge. In *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, pages 121–145, 2014. doi:10.1007/978-3-642-54242-8_6.

42 Yael Kalai and Ran Raz. Interactive PCP. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, ICALP '08, pages 536–547, 2008.

43 Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, STOC '92, pages 723–732, 1992.

44 Joe Kilian, Erez Petrank, and Gábor Tardos. Probabilistically checkable proofs with zero knowledge. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, STOC '97, pages 496–505, 1997.

45 Swastik Kopparty, Or Meir, Noga Ron-Zewi, and Shubhangi Saraf. High-rate locally-correctable and locally-testable codes with sub-polynomial query complexity. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 202–215, 2016. doi:10.1145/2897518.2897523.

46 Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992.

47 Gregory Maxwell. Zero knowledge contingent payment, 2011. [Online; accessed 13-October-2017].

48 Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000. Preliminary version appeared in FOCS '94.

49 Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000. doi:10.1137/S0097539795284959.

50 Thilo Mie. Short PCPPs verifiable in polylogarithmic time with o(1) queries. *Annals of Mathematics and Artificial Intelligence*, 56:313–338, 2009.

51 Dana Moshkovitz and Ran Raz. Two-query PCP with subconstant error. *J. ACM*, 57(5):29:1–29:29, 2010. doi:10.1145/1754399.1754402.

52 M. Peck. A blockchain currency that beat s bitcoin on privacy [news]. *IEEE Spectrum*, 53(12):11–13, December 2016. doi:10.1109/MSPEC.2016.7761864.

53 Alexander Polishchuk and Daniel A. Spielman. Nearly-linear size holographic proofs. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, STOC '94, pages 194–203, 1994.

54 John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *Quantum Info. Comput.*, 3(4):317–344, 2003. URL: http://dl.acm.org/citation.cfm?id=2011528.2011531.

55 Ran Raz. A parallel repetition theorem. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, STOC '95, pages 447–456, 1995.

56 I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960. doi:10.1137/0108018.

57 Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 49–62, 2016. doi:10.1145/2897518.2897652.

58 Ronitt Rubinfeld and Madhu Sudan. Self-testing polynomial functions efficiently and over rational domains. In *Proceedings of the Third Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 27-29 January 1992, Orlando, Florida.*, pages 23–32, 1992. URL: http://dl.acm.org/citation.cfm?id=139404.139410.

59 Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.

60 P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Nov 1994. doi:10.1109/SFCS.1994.365700.

61 Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Proceedings of the 5th Conference on Theory of Cryptography*, TCC'08, pages 1–18, Berlin, Heidelberg, 2008. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=1802614.1802616.

62 Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, 58(2):74–84, 2015. doi:10.1145/2641562.

NP-Hardness of Coloring 2-Colorable Hypergraph with Poly-Logarithmically Many Colors

Amey Bhangale[1]
Weizmann Institute of Science, Rehovot, Israel
amey.bhangale@weizmann.ac.il

Abstract

We give very short and simple proofs of the following statements: Given a 2-colorable 4-uniform hypergraph on n vertices,

1. It is NP-hard to color it with $\log^{\delta} n$ colors for some $\delta > 0$.
2. It is *quasi*-NP-hard to color it with $O\left(\log^{1-o(1)} n\right)$ colors.

In terms of NP-hardness, it improves the result of Guruswam, Håstad and Sudani [SIAM Journal on Computing, 2002], combined with Moshkovitz-Raz [Journal of the ACM, 2010], by an 'exponential' factor. The second result improves the result of Saket [Conference on Computational Complexity (CCC), 2014] which shows *quasi*-NP-hardness of coloring a 2-colorable 4-uniform hypergraph with $O\left(\log^{\gamma} n\right)$ colors for a sufficiently small constant $1 \gg \gamma > 0$.

Our result is the first to show the NP-hardness of coloring a c-colorable k-uniform hypergraph with poly-logarithmically many colors, for *any* constants $c \geq 2$ and $k \geq 3$.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases Hypergraph coloring, Inapproximability, Schrijver graph

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.15

Acknowledgements I would like to thank Irit Dinur for helpful discussions. I would also like to thank Subhash Khot and Rishi Saket for their invaluable comments on the draft. Finally, I would like to thank all the anonymous reviewers for their comments which helped in improving the presentation significantly.

1 Introduction

A k-uniform hypergraph $\mathcal{H}(\mathcal{V}, \mathcal{E})$ is a collection of vertices $\mathcal{V}$ and a family $\mathcal{E}$ of subsets of size k of the vertices. Let n denote the number of vertices $|\mathcal{V}|$. A coloring of $\mathcal{H}$ with c colors is a mapping $\chi : \mathcal{V} \to [c]$. A coloring χ is said to be a valid c-coloring if for every $e \in \mathcal{E}$, χ assigns at least two different colors to the vertices in e. The *chromatic number* of a hypergraph is the minimum c for which a valid c-coloring exists.

It is easy to find a 2-coloring of a 2-colorable graph. Deciding whether a graph is 3-colorable or not is a well known NP-complete problem [7]. In an approximate graph (or hypergraph) coloring problem, given a graph which is c colorable, the algorithm is allowed to use more than c colors to properly color the given graph. The best known polynomial-time algorithms to color a 3-colorable graph require $n^{\Omega(1)}$ colors. However, the known NP-hardness results can only rule out coloring a 3-colorable graph with 4 colors [11, 10]. This also implies that coloring a t-colorable graph with $t + 2\lfloor t/3 \rfloor - 1$ colors is NP-hard. Garey and Johnson [7] proved NP-hardness of $(2t-5)$-coloring a t-colorable graph, for all $t \geq 6$. Recently,

[1] Research supported by Irit Dinur's ERC-CoG grant 772839.

Brakensiek-Guruswami [4] showed that for all $t \geq 3$, it is NP-hard to find a c-coloring when $c \leq 2t - 2$.

For hypergraphs with uniformity 3 and more, the situation is completely different when the hypergraph is 2-colorable. The best known approximation algorithm for coloring a 2-colorable hypergraph is with n^{ϵ} color for some explicit $\epsilon \in (0, 1)$. It is easy to see that if t-coloring of c-colorable k-uniform hypergraphs is hard, then it also implies a similar hardness result for k'-uniform hypergraphs for any $k' > k$ up to polynomial-time reductions i.e. coloring a c-colorable k'-uniform hypergraph with t colors is also hard. Therefore, the wishful inapproximability result is to get n^{ϵ} coloring hardness for 2-colorable 3-uniform hypergraphs for some $\epsilon > 0$.

Two directions have been pursued in literature. One is getting better and better hardness results in terms of colorability, where the uniformity of a hypergraph is any constant k and the guarantee is that the hypergraph is colorable with constantly many colors. The other is getting hardness results for smaller values of k . Both the directions are equally interesting and have been looked into for several years. Table 1 summarizes the known inapproximability results: For *e.g.* Guruswami *et al.* [9] (the first row) showed that given a 2-colorable 4-uniform hypergraph, there is no polynomial-time algorithm to color it with $O\left(\frac{\log\log n}{\log\log\log n}\right)$ many colors unless $\mathsf{NP} \subseteq \mathsf{DTIME}\left(n^{O(\log\log n)}\right)$. We would like to point out that the condition $\mathsf{NP} \subsetneq \mathsf{DTIME}\left(n^{O(\log\log n)}\right)$ from [9] can be replaced with $\mathsf{P} \neq \mathsf{NP}$, if one carries out the reduction of [9] starting with the PCP of [17].

We prove the following results:

▶ **Theorem 1.** *Assuming* $\mathsf{NP} \nsubseteq \mathsf{DTIME}(n^{O(\log\log n)})$*, there is no polynomial-time algorithm that colors a* 2*-colorable* 4*-uniform hypergraph on* n *vertices with* $O\left(\frac{\log n}{\log\log n}\right)$ *colors.*

Although an explicit constant in the exponent of $\log n$ was not mentioned in [19], it is less than 0.01. Therefore, Theorem 1 improves the best known inapproximability result by Saket [19] by a large polynomial factor, in terms of coloring with few colors. However, both the results are incomparable (See Remark 1).

Our second result shows NP-hardness of coloring a 2-colorable 4-uniform hypergraph.

▶ **Theorem 2.** *Assuming* $\mathsf{P} \neq \mathsf{NP}$*, there is no polynomial-time algorithm that colors a* 2*-colorable* 4*-uniform hypergraph on* n *vertices with* $\log^{\delta} n$ *colors for some* $\delta > 0$*.*

Theorem 2 improves the best known inapproximability result by Guruswami *et al.*[9] by an'exponential' factor, in terms of coloring with few colors. In fact, our result is the first to show the NP-hardness of coloring a c-colorable k-uniform hypergraph with $\texttt{poly}(\log n)$ colors for any constants $c \geq 2$ and $k \geq 3$.

An independent set in a hypergraph is defined as a set of vertices such that no hyperedge lies entirely within the set. In a valid coloring of a hypergraph, the vertices colored with the same color form an independent set. Thus, saying that a hypergraph does not contain an independent set of size n/c is stronger than saying that it cannot be colored with c colors.

In terms of approximating the size of the maximum independent set, Khot-Saket [15] showed that given an *almost* 2-colorable 4-uniform hypergraph, it is quasi-NP-hard to find an independent set of fractional size $2^{(\log n)^{1-\gamma}}$, for any $\gamma > 0$. More concretely, they show that it is quasi-NP-hard to distinguish between cases when a 4-uniform hypergraph has an independent set of size at least $\left(\frac{n}{2} - o(n)\right)$ vs. when it has no independent set of size $n/2^{(\log n)^{1-\gamma}}$, for any $\gamma > 0$.

▶ Remark. With the exception of results from this paper and Dinur *et al.*(denoted by ** in Table 1), all the remaining results give stronger inapproximability results than showing non

Table 1 Known inapproximability results for hypergraph coloring (for ** see Remark 1).

	Completeness	Soundness	Assumption
Guruswami *et al.* [9, 17]	**2-colorable 4-uniform hypergraph**	$\Omega\left(\frac{\log\log n}{\log\log\log n}\right)$	$\mathsf{P} \neq \mathsf{NP}$
Khot [12]	q-colorable 4-uniform hypergraph, $q \geq 5$	$(\log n)^{cq}$ for some $c > 0$	$\mathsf{NP} \not\subseteq \mathsf{DTIME}\left(n^{(\log n)^{O(1)}}\right)$
Khot [13]	3-colorable 3-uniform hypergraph	$(\log\log n)^{1/9}$	$\mathsf{NP} \not\subseteq \mathsf{DTIME}\left(n^{(\log n)^{O(1)}}\right)$
**Dinur *et al.* [5]	2-colorable 3-uniform hypergraph	$\Omega(\sqrt[3]{\log\log n})$	$\mathsf{NP} \not\subseteq \mathsf{DTIME}\left(n^{(\log n)^{O(1)}}\right)$
Saket [19]	**2-colorable 4-uniform hypergraph**	$\Omega(\log^{\delta} n)$ for $0 < \delta \ll 1$	$\mathsf{NP} \not\subseteq \mathsf{DTIME}\left(n^{O(\log\log n)}\right)$
Guruswami *et al.* [8]	2-colorable 8-uniform hypergraph 4-colorable 4-uniform hypergraph	$2^{2^{\Omega(\sqrt{\log\log n})}}$	$\mathsf{NP} \not\subseteq \mathsf{DTIME}\left(n^{2^{O(\sqrt{\log\log n})}}\right)$
Guruswami *et al.* [8]	3-colorable 3-uniform hypergraph	$2^{\Omega(\frac{\log\log n}{\log\log\log n})}$	$\mathsf{NP} \not\subseteq \mathsf{DTIME}\left(n^{2^{O(\frac{\log\log n}{\log\log\log n})}}\right)$
Khot-Saket [14]	2-colorable 12-uniform hypergraph	$2^{(\log n)^{\Omega(1)}}$	$\mathsf{NP} \not\subseteq \mathsf{DTIME}\left(n^{(\log n)^{O(1)}}\right)$
Varma [21]	2-colorable 8-uniform hypergraph 4-colorable 4-uniform hypergraph	$2^{(\log n)^{\Omega(1)}}$	$\mathsf{NP} \not\subseteq \mathsf{DTIME}\left(n^{(\log n)^{O(1)}}\right)$
**This paper	**2-colorable 4-uniform hypergraph**	$\Omega\left(\frac{\log n}{\log\log n}\right)$	$\mathsf{NP} \not\subseteq \mathsf{DTIME}\left(n^{O(\log\log n)}\right)$
**This paper	**2-colorable 4-uniform hypergraph**	$\Omega(\log^{\delta} n)$ for $0 < \delta \ll 1$	$\mathsf{P} \neq \mathsf{NP}$

c-colorability in *soundness*. Namely, they show it is hard to distinguish between cases when a hypergraph satisfies the property in the *completeness* column vs. when there does not exist an independent set of size at least $1/s$, where s is the quantity in the *soundness* column.

1.1 Proof Overview

Our inapproximability result is obtained by constructing a *probabilistically checkable proof* where the locations are the vertices of a hypergraph and every check denotes a hyperedge of the hypergraph. Typically, the PCP is constructed by composing the so-called *outer verifier* with the *inner verifier*. Our starting point is a basic outer verifier that one gets from the PCP Theorem [6, 1, 2], along with Raz's parallel repetition theorem [18]. We can view it as an instance of a *Label Cover*. A Label Cover instance consists of a bipartite graph $G(U, V, E)$ where every edge is associated with a projection constraint $\pi_e : [R] \to [L]$ (See section 2.1). The inner verifier consists of encoding of labels of the vertices in a specific format and performing a test on the encoding.

A typical hardness of approximation result uses the *Long Code* encoding where the encoding of a k bit binary (or q-ary) string has an entry for every $f : \{0,1\}^k \to \{0,1\}$ (or $f : [q]^k \to [q]$ for $q \geq 2$). This blows up a string of length k into a string of length 2^{2^k}. Such an encoding is used in the construction of PCP by Guruswami *et al.* [9] and they managed to show a $\approx \log\log n$ lower bound on the inapproximability of coloring a 2-colorable 4-uniform

hypergraph. Khot [12] used a different encoding based on *Split Code* to get the hardness of poly-logarithmic number of colors, but for q-colorable 4-uniform hypergraph where $q \geq 5$. Saket [19] improved the result of Guruswami *et al.* [9] and showed the inapproximability of coloring with $O((\log n)^{\delta})$ colors for a very small constant $0 < \delta \ll 1$. Saket also used the Long Code encoding, but the hypergraph is formed using a different inner verifier compared to [9]. In a more recent series of works [8, 14, 21], efficient encodings were designed based on so-called *Short Code* which was used to break the poly-logarithmic barrier. It is not clear how to use this Short Code encoding to prove better inapproximability for 2-colorable 4-uniform hypergraphs. In all the previous constructions of PCPs based on Short Code, either the alphabet size is at least 3 (the guarantee on the hypergraph colorability in the completeness case) or the number of queries (the uniformity of the constructed hypergraph) is at least 6 if the alphabet size is restricted to 2 (See Table 1).

Our reduction is along the lines of the reduction in [5] which showed that coloring a 2-colorable 3-uniform hypergraph with $(\log \log n)^{1/3}$ colors is quasi-NP-hard. We give a very brief outline of their reduction first: This reduction also encodes a label in the outer verifier with a (rather large) subset of the Long Code encoding of size $2^{2^k - \mathtt{poly}(k)}$. They were able to construct a PCP over a binary alphabet which only queries 3 bits, giving the hardness for the above mentioned coloring problem. The encoding consists of all the locations in the Long Code encoding which correspond to a *slice* of $\{0,1\}^{2^k}$ of hamming weight m, where m is roughly $2^k/2$. The reduction crucially used two important properties of a specific graph, known as *Kneser graph*, on these locations. The first is that the chromatic number of the graph is *large* and the second is that in all the colorings of this graph with strictly fewer colors than the chromatic number, there is a *large* color class containing a monochromatic edge. The reduction also needed a multi-layered version of Label Cover, which gives the hardness of $(\log \log n)^{1/3}$ colors based on $\mathsf{NP} \not\subseteq \mathsf{DTIME}(n^{(\log n)^{O(1)}})$.

In our construction of the PCP, we start with a basic outer verifier and use a very short encoding of the labels of the outer verifier. More specifically, to get an inapproximability of c-colorability, we use an encoding of size roughly 2^{kc}, which is much shorter than the Long Code encoding 2^{2^k} or the encoding used in [5] for $c \ll 2^k$. This short encoding is the reason for improvement in the inapproximability. Of course, the whole analysis needs to work for such a short encoding. For the analysis to work, we need to query one more bit from the composed PCP, giving a hardness result for 4-uniform hypergraphs.

The subset of locations in the Long Code that we use corresponds to the vertex set of the *Schrijver graph* (See section 2.2 for the definition). It has much fewer number of vertices than the Kneser graph on the same slice. Moreover, the Schrijver graph also has a property that its chromatic number is large. Unfortunately, it is known to be a *vertex-critical graph* [2]. Thus, the property analogous to the second property of the Kneser graph mentioned previously no longer holds for the Schrijver graph. However, if we are allowed to query one more bit, then given an efficient coloring of the constructed 4-uniform hypergraph, it is possible to list-decode, from the coloring, a small list of possible labelings to the vertices of the outer verifier. The structure of the hyperedges in our 4-uniform hypergraph ensures that the lists are *consistent* with each other i.e. there is a way to assign labels to the vertices of the Label Cover instance from the list, which satisfies many constraints of the Label Cover instance.

[2] A vertex-critical graph is a graph G in which every vertex is a critical element, that is, if its deletion decreases the chromatic number of G.

$\log^{\delta} n$ NP-hardness

As highlighted earlier, the size of the encoding we use is 2^{kc} and if we start with the NP-hardness of the Label Cover given by [17], which has better parameters in the subconstant soundness regime, we can run the whole reduction in polynomial-time if we set $c = \log^{\delta} n$ for small enough $\delta > 0$. Therefore, we get the required inapproximability result proving Theorem 2.

$\Omega\left(\frac{\log n}{\log\log n}\right)$ *quasi*-NP-hardness

The soundness parameter of the Label Cover instance given by [17] has an *inverse exponential* dependence on the alphabet size of the instance (See Theorem 5). To get the improved $\log^{1-o(1)} n$ coloring hardness, we start with the usual Label Cover instance derived from the combination of the PCP Theorem and the parallel repetition theorem. In this case, there is an *inverse polynomial* dependence on the soundness parameter and the alphabet size which enables us to prove a better hardness factor. One drawback of using this Label Cover instance is that the instance size grows polynomially with the number of parallel repetitions. More concretely, an instance of size n becomes an instance of size $n^{O(r)}$ with r repetitions. Thus, it restricts us to assume $\mathsf{NP} \not\subseteq \mathsf{DTIME}\left(n^{O(\log\log n)}\right)$ instead of $\mathsf{P} \neq \mathsf{NP}$, as we need $r = \Omega(\log\log n)$ repetitions.

1.2 $\log^{\delta} n$ NP-hardness via Independent set analysis?

We discuss here informally why the previous approaches could not reach $\mathtt{poly}(\log n)$ factor NP-hardness for approximating hypergraph coloring. Since all the previous approaches (except [5]) get the inapproximability result by showing the hardness of approximating an independent set, we focus on those works here. To get NP-hardness of $\mathtt{poly}(\log n)$-coloring by showing the NP-hardness of approximating an independent set by $1/\mathtt{poly}(\log n)$ factor, usually the soundness of the outer verifier (or the Label Cover instance) that one needs is at most $1/\mathtt{poly}(\log n)$. The outer verifier of [17] has an exponential dependence on the label set size and the inverse of the soundness parameter. Thus, the label set size which is required for such reductions is at least $2^{\mathtt{poly}(\log n)}$. The analysis of [8] uses the Short Code encoding where the degree is $\omega(1)$. Therefore, the blowup in the reduction size is $2^{\mathtt{poly}(\log n)^{\omega(1)}}$, which is super polynomial.

One can use a Short Code encoding where the degree is constant so that the blow-up is limited to $2^{\mathtt{poly}(\log n)^{O(1)}}$. [14] used a Quadratic Code where the degree is 2. One can possibly use the Quadratic Code encoding or Short Code encoding of constant degree and get a hypergraph of polynomial size starting with a Label Cover instance with $1/\mathtt{poly}(\log n)$ soundness. Unfortunately, the analysis of [14] crucially needed an outer verifier with certain properties. It is not yet known how to get a Label Cover instance by a *polynomial-time reduction* starting from a 3SAT instance, with similar properties that were needed in [14] and with $1/\mathtt{poly}(\log n)$ soundness (the latter was done in [17]).

2 Preliminaries

We denote a set $\{1, 2, \ldots, n\}$ by $[n]$. Bold face letters $\mathbf{a}, \mathbf{b}, \mathbf{c} \ldots$ are used to denote strings and subscripts are used to denote the elements at the respective locations in a string. By an abuse of notation, we will use $\mathbf{a} \in \{0,1\}^n$ as a binary string of length n as well as a subset of $[n]$ given by $\{i \in [n] \mid \mathbf{a}_i = 1\}$. We will denote by $2^{[n]}$ the set of all subsets of $[n]$ and by $\binom{[n]}{k}$ the set of all subsets of $[n]$ of size k. We denote the quantity $O(n^k)$, where $k \in \mathbb{R}^+$ by $\mathtt{poly}(n)$.

2.1 Label Cover

Now, we define a Label Cover instance which is the starting point of our reduction.

▶ **Definition 3** (Label Cover)**.** A Label Cover instance $\mathcal{L} = (U, V, E, [R], [L], \phi)$ consists of a bi-regular bipartite graph on *two* sets of variables $U \cup V$. The range of variables in U is denoted by $[R]$ and that in V by $[L]$. Every $(x, y) \in E$, has a constraint $\phi_{x\to y} : [R] \to [L]$. Moreover, every constraint between a pair of variables is a projection constraint i.e. a labeling to x uniquely defines a labeling to y that satisfies the constraint $\phi_{x\to y}$.

For brevity, we say $x \sim y$, or 'x is a neighbor of y' if $\phi_{x\to y} \in \phi$. We say that a Label Cover instance is ϵ-satisfiable, if there exists a labeling to the variables which satisfies at least ϵ fraction of the constraints between U and V.

We have the following NP-hardness result which follows from the PCP Theorem [6, 1, 2] along with Raz's parallel repetition theorem [18].

▶ **Theorem 4.** *For any parameter $l \in \mathbb{N}$, there exists a reduction from a 3-SAT instance of size n to a Label Cover instance with $n^{O(\ell)}$ variables over a range of size $2^{O(\ell)}$. The Label Cover instance has the following completeness and soundness conditions:*

- *If the 3-SAT instance is satisfiable, then there exists an assignment to the Label Cover instance that satisfies all the constraints.*
- *If the 3-SAT instance is not satisfiable, then every assignment to the Label Cover instance satisfies at most $2^{-\Omega(\ell)}$ fraction of the constraints.*

Moreover, the reduction runs in time $n^{O(\ell)}$.

For our NP-hardness result, we need the following reduction from a 3-SAT instance to a Label Cover instance by [17], which gives better parameters for *subconstant soundness*.

▶ **Theorem 5.** *There exist absolute constants $c', c'' > 1$ such that for every n and $\epsilon > 0$ (ϵ can be any function of n), there exists a reduction from a 3-SAT instance of size n to a Label Cover instance with $n^{1+o(1)} \cdot \left(\frac{1}{\epsilon}\right)^{c'}$ variables over a range of size $2^{\left(\frac{1}{\epsilon}\right)^{c''}}$. The Label Cover instance has the following completeness and soundness conditions:*

- *If the 3-SAT instance is satisfiable, then there exists an assignment to the Label Cover instance that satisfies all the constraints.*
- *If the 3-SAT instance is not satisfiable, then every assignment to the Label Cover instance satisfies at most ϵ fraction of the constraints.*

Moreover, the reduction runs in time $\texttt{poly}(n, \frac{1}{\epsilon})$.

2.2 Schrijver Graphs

In this section, we define a Schrijver graph of order n, k, denoted by $SG(n, k)$, where $n \geq 3, k < n$ and $n - k$ is even, which we use as a gadget in our reduction. The vertex set of this Schrijver graph, denoted by $V_{SG}(n, k)$, is a subset of $\{0, 1\}^n$. An element $\mathbf{a} \in \{0, 1\}^n$ belongs to $V_{SG}(n, k)$ iff

1. The hamming weight of $\mathbf{a}$ is $\frac{n-k}{2}$.
2. There exists no $i \in [n]$ such that both $\mathbf{a}_i = 1$ and $\mathbf{a}_{(i+1) \mod (n+1)} = 1$.

In other words, if we denote a cycle graph on $\{1, 2, \ldots, n\}$ by C_n, then the vertex set $V(n, k)$ corresponds to all independent sets in C_n of size $\frac{n-k}{2}$. The edge set of $SG(n, k)$ is as follows:

$$E_{SG}(n, k) = \{(\mathbf{a}, \mathbf{b}) \mid \mathbf{a} \cap \mathbf{b} = \emptyset\}.$$

Vertices $\mathcal{V}$. Each vertex $x \in U$ in the Label Cover instance $\mathcal{L}$ is replaced by a cloud of size $|V_{SG}(R,k)|$ denoted by $C[x] := x \times V_{SG}(R,k)$. We refer to a vertex from the cloud $C[x]$ by a pair $(x, \mathbf{a})$, where $\mathbf{a} \in V_{SG}(R,k)$. The vertex set of the hypergraph $\mathcal{H}$ is given by

$$\mathcal{V} = \cup_{x \in U} C[x].$$

Hyperedges $\mathcal{E}$. The hyperedges of $\mathcal{H}$ are given by sets $\{(x,\mathbf{a}),(x,\mathbf{b}),(y,\mathbf{c}),(y,\mathbf{d})\}$ such that the following conditions hold:

1. There exists a common neighbor z of x and y in V.
2. For every $\alpha, \beta \in [R]$ such that $\phi_{x\to z}(\alpha) = \phi_{y\to z}(\beta)$, $\{\mathbf{a}_\alpha, \mathbf{b}_\alpha, \mathbf{c}_\beta, \mathbf{d}_\beta\}$ are not all the same i.e. $|\{\mathbf{a}_\alpha, \mathbf{b}_\alpha, \mathbf{c}_\beta, \mathbf{d}_\beta\}| = 2$.

Figure 1 Reduction to a 4-uniform hypergraph $\mathcal{H}(\mathcal{V},\mathcal{E})$

In other words, $\mathbf{a} \sim \mathbf{b}$ in $SG(n,k)$ iff their supports are disjoint.

We need two properties of the Schrijver graph $SG(n,k)$, first of which is easy to prove.

▶ **Claim 6** ([5]). $|V_{SG}(n,k)| \leq \binom{n}{k}$.

Proof. Map every vertex $\mathbf{x} \in V_{SG}(n,k)$ to $\tilde{\mathbf{x}} \in \{0,1\}^n$ by setting $\tilde{\mathbf{x}}_i = \tilde{\mathbf{x}}_{i+1 \mod (n+1)} = 1$ iff $\mathbf{x}_i = 1$ and rest of the coordinates to 0. Clearly, this is an injective map from $V_{SG}(n,k)$ to the set $\binom{[n]}{n-k}$. ◀

The second property of $SG(n,k)$ is its chromatic number. The following theorem from [20] gives the exact chromatic number of $SG(n,k)$ which builds on the beautiful proofs by Lovász [16] and Bárány [3].

▶ **Theorem 7** ([20]). *The chromatic number of $SG(n,k)$ is $k+2$.*

3 Main Reduction

We give a reduction from a Label Cover instance $\mathcal{L} = (U, V, E, [R], [L], \phi)$ with a parameter $R > L$ and a parameter k, both of which we will set later, to a 4-uniform hypergraph $\mathcal{H}(\mathcal{V},\mathcal{E})$. We also assume that $R - k$ is even. The reduction is given in Figure 1.

The completeness is easy to prove:

▶ **Lemma 8** (Completeness). *If the Label Cover instance is satisfiable then the hypergraph $\mathcal{H}$ is 2-colorable.*

Proof. Let $A : U \cup V \to [R] \cup [L]$ be the perfectly satisfiable labeling to the Label Cover instance $\mathcal{L}$. The 2-coloring of the hypergraph $\mathcal{H}$ is given by assigning a vertex $(x,\mathbf{a}) \in C[x]$ with a color $\mathbf{a}_{A(x)}$. We show that this is a valid 2-coloring of $\mathcal{H}$. Suppose not, in which case, there exists a monochromatic hyperedge. Let that hyperedge be $\{(x,\mathbf{a}),(x,\mathbf{b}),(y,\mathbf{c}),(y,\mathbf{d})\}$. Let z be the common neighbor responsible for adding this edge. It must be that $\phi_{x\to z}(A(x)) \neq \phi_{y\to z}(A(y))$ and hence A is not a perfectly satisfiable assignment, which is a contradiction. ◀

We now show the soundness of the reduction.

▶ **Lemma 9** (Soundness). *If the Label Cover instance is not $\frac{1}{(k+1)k^2}$ satisfiable then the hypergraph is not even $k+1$-colorable.*

Proof. In this case, we show that if the hypergraph is $(k+1)$-colorable, then there exists a labeling to the Label Cover instance that satisfies at least $\frac{1}{(k+1)k^2}$ fraction of the constraints between the two layers U and V.

Suppose the hypergraph is $(k+1)$-colorable. Fix a $k+1$ coloring χ of the hypergraph. Consider the Schrijver Graph $SG(R,k)$ defined on the cloud $C[x]$ where $x \in U$. By Theorem 7, there exists a monochromatic edge i.e $\chi((x,\mathbf{a})) = \chi((x,\mathbf{b}))$ where $(\mathbf{a},\mathbf{b}) \in E_{SG}(R,k)$. Label a cloud $C[x]$ (and hence the vertex x) $c \in [k+1]$ if that edge is c-colored (breaking ties arbitrarily). Out of $|U|$ vertices, there exists at least $1/(k+1)$ fraction of the vertices with the same color. Let that color be c'.

From now on, we denote the c' colored vertices of U by U'. We know from the bi-regularity of the Label Cover instance that the total number of constraints between U' and V is at least $\frac{1}{k+1}$ fraction of the constraints between layers U and V. Thus, if we show that we can satisfy at least $\frac{1}{k^2}$ fraction of the constraints between U' and V, then we are done. This will satisfy $\frac{1}{(k+1)k^2}$ fraction of the constraints between layers U and V.

List-Labeling

We define the list-labeling A to the vertices in U' as follows: For $x \in U'$, in the induced Schrijver Graph on $C[x]$, there exists a monochromatic edge with color c'. Let that edge be $\{(x,\mathbf{a}),(x,\mathbf{b})\}$. Let $A(x) = [R] \setminus (\mathbf{a} \cup \mathbf{b})$.

▶ **Observation 10.** *For every $x \in U'$, $|A(x)| = k$.*

We need a simple fact as follows:

▶ **Fact 11.** *For every pairwise intersecting family $\mathcal{F} \subseteq \binom{[n]}{k}$, there exists $i \in [n]$ which is present in at least $1/k$ fraction of the sets in $\mathcal{F}$.*

Proof. Suppose not. Consider $A = \{a_1, a_2, \ldots, a_k\} \in \mathcal{F}$ and the sub-families $\mathcal{A}_i = \{A' \in \mathcal{F} \mid a_i \in A'\}$ for $i \in [k]$. By assumption, we have $|\mathcal{A}_i| < |\mathcal{F}|/k$. By the union bound, $|\cup_i \mathcal{A}| \leq \sum_i |\mathcal{A}_i| < |\mathcal{F}|$. Thus, there exists $B \in \mathcal{F} \setminus (\cup_i \mathcal{A}_i)$ and $A \cap B = \emptyset$, a contradiction. ◀

The following two claims finish the proof.

▶ **Claim 12.** *For every $x, y \in U'$ that have a common neighbor z in V,*

$$\phi_{x\to z}(A(x)) \cap \phi_{y\to z}(A(y)) \neq \emptyset.$$

Proof. For contradiction, suppose there exist $x, y \in U'$ and a common neighbor z such that $\phi_{x\to z}(A(x)) \cap \phi_{y\to z}(A(y)) = \emptyset$. Let $\{(x,\mathbf{a}),(x,\mathbf{b})\}$ and $\{(y,\mathbf{c}),(y,\mathbf{d})\}$ be the monochromatic c'-colored edges that were used to define A on x and y respectively. Since, $\phi_{x\to z}(A(x)) \cap \phi_{y\to z}(A(y)) = \emptyset$, we know that $\phi_{x\to z}([R] \setminus (\mathbf{a} \cup \mathbf{b}))$ and $\phi_{y\to z}([R] \setminus (\mathbf{c} \cup \mathbf{d}))$ are disjoint. This means that for every $\alpha, \beta \in [R]$ such that $\phi_{x\to z}(\alpha) = \phi_{y\to z}(\beta)$, we have $\mathbf{a}_\alpha, \mathbf{b}_\alpha, \mathbf{c}_\beta, \mathbf{d}_\beta$, not all of which are 1 (this follows from $\mathbf{a}_\alpha, \mathbf{b}_\alpha$ not both equal to 1 as they form an edge in the Schrijver graph $SG(R,k)$). Also, not all of $\mathbf{a}_\alpha, \mathbf{b}_\alpha, \mathbf{c}_\beta, \mathbf{d}_\beta$ are 0, since otherwise $\alpha \in A(x)$ and $\beta \in A(y)$ and hence $\phi_{x\to y}(A(x)) \cap \phi_{y\to z}(A(y)) \neq \emptyset$. Thus, $\{(x,\mathbf{a}),(x,\mathbf{b}),(y,\mathbf{c}),(y,\mathbf{d})\}$ is a valid hyperedge in $\mathcal{H}$, which is monochromatic with color c'. This contradicts the fact that χ is a valid $k+1$ coloring of $\mathcal{H}$. ◀

Now, consider the following randomized labeling B:

$$\forall z \in V, \qquad B(z) \leftarrow \arg\max_{i \in [L]} |\{x \mid (x \in U') \wedge (z \sim x) \wedge i \in \phi_{x \to z}(A(x))\}|$$

$$\forall x \in U', \qquad B(x) \leftarrow \text{ A uniformly random label from } A(x).$$

In other words, $B(z)$ for $z \in V$ is a label $i \in [L]$ which is the most common projection (w.r.t. $\phi_{x \to z}$) of the labels $A(x)$, where $x \sim z$ and $x \in U'$.

▶ **Claim 13.** *The randomized labeling B satisfies at least $\frac{1}{k^2}$ fraction of the constraints between U' and V in expectation.*

Proof. Fix $z \in V$ and let $X = \{x_1, x_2, \ldots, x_t\}$ be the neighbors of z in U' which satisfy $B(z) \in \phi_{x_i \to z}(A(x_i))$. Thus, X constitutes at least $1/k$ fraction of z's neighbors in U' using Claim 12 and Fact 11. The edge (x_i, z) is satisfied by the labeling $(B(x_i), B(z))$ with probability $1/|A(x_i)| = 1/k$ using Observation 10. Thus, in expectation, the randomized labeling satisfies at least $1/k^2$ fraction of the constraints between z and U'. Finally, by linearity of expectation, B satisfies at least $1/k^2$ fraction of the constraints between U' and V. ◀

Thus, the partial labeling B satisfies at least $\frac{1}{(k+1)k^2}$ fraction of the constraints between U and V in expectation. ◀

3.1 Setting of parameters

Proof of Theorem 1: Starting with a 3-SAT instance of size n, we first reduce it to a Label Cover instance given by Theorem 4 with parameter $\ell = c_0 \log\log n$ for a large constant $c_0 > 1$, so that the soundness of the Label Cover instance is $2^{-\Omega(\ell)} \ll \Omega(\log^{-3} n)$. We will set $k = \log n$. Thus, the number of vertices in the hypergraph $\mathcal{H}$ is $N = n^{O(\ell)} \cdot 2^{O(\ell) \cdot k} = n^{O(\log\log n)}$ and the reduction runs in time $n^{O(\log\log n)}$. Therefore, assuming $\mathsf{NP} \not\subseteq \mathsf{DTIME}(n^{O(\log\log n)})$, there is no polynomial-time algorithm to color a 2-colorable 4-uniform hypergraph on N vertices with $O\left(\frac{\log N}{\log\log N}\right)$ colors. ◀

Proof of Theorem 2: Starting with a 3-SAT instance of size n, we first reduce it to a Label Cover instance given by Theorem 5 with parameter $\epsilon = (\log n)^{-4\delta}$ for a sufficiently small constant $\delta > 0$. Thus, the soundness of the Label Cover instance is $(\log n)^{-4\delta}$, the alphabet size is $|\Sigma| = 2^{(\log n)^{c'\delta}}$ and the size of the Label Cover instance is $n^{1+o(1)} \cdot (\log n)^{c''\delta}$ for some absolute constants $c', c'' > 1$. We will set $k = (\log n)^{\delta}$. Thus, the number of vertices in the hypergraph $\mathcal{H}$ is $N \leq n^{1+o(1)} \cdot (\log n)^{c''\delta} \cdot |\Sigma|^k = n^{1+o(1)} \cdot (\log n)^{c''\delta} \cdot 2^{(\log n)^{4c'\delta+\delta}} = n^{1+o(1)}$, for small enough δ. Also, the overall reduction starting from an instance of 3-SAT of size n runs in time $\mathtt{poly}(n)$. Therefore, it is NP-hard to color a 2-colorable 4-uniform hypergraph on N vertices with $\log^{\delta} N$ colors for some $\delta > 0$. ◀

4 Conclusion

Long Code has been used extensively to prove inapproximability results. Many inapproximability results benefit from *puncturing/derandomization* of the Long Code constructions. Most of them are algebraic in nature *e.g.* Hadamard code, Split code, Short code etc. We made an attempt to find a puncturing of the Long Code which is more combinatorial in nature. We believe that such combinatorial puncturings might find new applications in the

hardness of approximation as well as in other areas which use the Long Code or similar objects.

We conclude with a couple of obvious interesting open problems:

1. Can we show that coloring a 2-colorable 3-uniform hypergraph with $\omega(\log\log n)$ colors is NP-hard, or even *quasi*-NP-hard?
2. Can we go beyond $\texttt{poly}(\log n)$ coloring NP-hardness factor for coloring a c-colorable k-uniform hypergraph for some constants $c \geq 2$ and $k \geq 3$?

References

1 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof Verification and the Hardness of Approximation Problems. *J. ACM*, 45(3):501–555, 1998. (Preliminary version in *33rd FOCS*, 1992). doi:10.1145/278298.278306.

2 Sanjeev Arora and Shmuel Safra. Probabilistic Checking of Proofs: A new characterization of NP. *JACM*, 45(1):70–122, 1998. (Preliminary version in *33rd FOCS*, 1992).

3 J Bárány. A short proof of Kneser's conjecture. *Journal of Combinatorial Theory, Series A*, 25(3):325–326, 1978.

4 Joshua Brakensiek and Venkatesan Guruswami. New hardness results for graph and hypergraph colorings. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 50. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

5 I Dinur, O Regev, and C Smyth. The hardness of 3-uniform hypergraph coloring. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 33–40. IEEE, 2002.

6 Uriel Feige, Shafi Goldwasser, Laszlo Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM (JACM)*, 43(2):268–292, 1996.

7 Michael R Garey and David S Johnson. The complexity of near-optimal graph coloring. *Journal of the ACM (JACM)*, 23(1):43–49, 1976.

8 Venkatesan Guruswami, Prahladh Harsha, Johan Håstad, Srikanth Srinivasan, and Girish Varma. Super-polylogarithmic Hypergraph Coloring Hardness via Low-degree Long Codes. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 614–623, New York, NY, USA, 2014. ACM.

9 Venkatesan Guruswami, Johan Håstad, and Madhu Sudan. Hardness of approximate hypergraph coloring. *SIAM Journal on Computing*, 31(6):1663–1686, 2002.

10 Venkatesan Guruswami and Sanjeev Khanna. On the hardness of 4-coloring a 3-colorable graph. *SIAM Journal on Discrete Mathematics*, 18(1):30–40, 2004.

11 Sanjeev Khanna, Nathan Linial, and Shmuel Safra. On the hardness of approximating the chromatic number. In *Theory and Computing Systems, 1993., Proceedings of the 2nd Israel Symposium on the*, pages 250–260. IEEE, 1993.

12 Subhash Khot. Hardness results for approximate hypergraph coloring. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 351–359. ACM, 2002.

13 Subhash Khot. Hardness results for coloring 3-colorable 3-uniform hypergraphs. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 23–32. IEEE, 2002.

14 Subhash Khot and Rishi Saket. Hardness of Coloring 2-Colorable 12-Uniform Hypergraphs with exp (logˆ{Omega (1)} n) Colors. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 206–215. IEEE, 2014.

15 Subhash Khot and Rishi Saket. Hardness of finding independent sets in 2-colorable and almost 2-colorable hypergraphs. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1607–1625. SIAM, 2014.

16 László Lovász. Kneser's conjecture, chromatic number, and homotopy. *Journal of Combinatorial Theory, Series A*, 25(3):319–324, 1978.

17 Dana Moshkovitz and Ran Raz. Two-query PCP with subconstant error. *Journal of the ACM (JACM)*, 57(5):29, 2010.

18 Ran Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998.

19 Rishi Saket. Hardness of finding independent sets in 2-colorable hypergraphs and of satisfiable CSPs. In *IEEE 29th Conference on Computational Complexity (CCC)*, pages 78–89, 2014.

20 A Schrijver. Vertex-critical subgraphs of Kneser-graphs. *Nieuw Archief voor Wiskunde*, 26:454–461, 1978.

21 Girish Varma. Reducing uniformity in Khot-Saket hypergraph coloring hardness reductions. *Chicago Journal OF Theoretical Computer Science*, 3:1–8, 2015.

Sublinear Algorithms for MAXCUT and Correlation Clustering

Aditya Bhaskara
School of Computing, University of Utah, USA
http://www.cs.utah.edu/~bhaskara
bhaskara@cs.utah.edu

Samira Daruki
Expedia Research, USA
https://sites.google.com/site/samiradaruki
sdaruki@expedia.com

Suresh Venkatasubramanian
School of Computing, University of Utah, USA
http://www.cs.utah.edu/~suresh
suresh@cs.utah.edu

Abstract

We study sublinear algorithms for two fundamental graph problems, MAXCUT and correlation clustering. Our focus is on constructing core-sets as well as developing streaming algorithms for these problems. Constant space algorithms are known for *dense* graphs for these problems, while $\Omega(n)$ lower bounds exist (in the streaming setting) for sparse graphs.

Our goal in this paper is to bridge the gap between these extremes. Our first result is to construct core-sets of size $\widetilde{O}(n^{1-\delta})$ for both the problems, on graphs with average degree n^{δ} (for any $\delta > 0$). This turns out to be optimal, under the exponential time hypothesis (ETH). Our core-set analysis is based on studying random-induced sub-problems of optimization problems. To the best of our knowledge, all the known results in our parameter range rely crucially on near-regularity assumptions. We avoid these by using a biased sampling approach, which we analyze using recent results on concentration of quadratic functions. We then show that our construction yields a 2-pass streaming $(1+\varepsilon)$-approximation for both problems; the algorithm uses $\widetilde{O}(n^{1-\delta})$ space, for graphs of average degree n^{δ}.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases Sublinear algorithms, Streaming algorithms, Core-sets, Maximum cut, Correlation clustering

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.16

Related Version A full version of the paper is available at https://arxiv.org/abs/1802.06992.

Funding This research was supported in part by National Science Foundation under grants IIS-1251049, IIS-1633724

1 Introduction

Sublinear algorithms are a powerful tool for dealing with large data problems. The range of questions that can be answered accurately using sublinear (or even polylogarithmic) space or time is enormous, and the underlying techniques of sketching, streaming, sampling and core-sets have been proven to be a rich toolkit.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 16; pp. 16:1–16:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

When dealing with large graphs, the sublinear paradigm has yielded many powerful results. For many NP-hard problems on graphs, classic results from property testing [20, 7] imply extremely efficient sublinear approximations. In the case of dense graphs, these results (and indeed older ones of [10, 16]) provide constant time/space algorithms. More recently, graph sketching techniques have been used to obtain efficient approximation algorithms for cut problems on graphs [2, 3] in a streaming setting. These algorithms use space that is nearly linear in n (the number of vertices) and are sublinear in the number of edges as long as $|E| = \omega(n)$ (this is called the "semi-streaming" setting).

By way of lower bounds, recent results have improved our understanding of the limits of sketching and streaming. In a sequence of results [22, 23, 25], it was shown that for problems like matching and MaxCut in a streaming setting, $\Omega(n)$ space is necessary in order to obtain any approximation better than a factor 2 in one round. (Note that a factor 2 is trivial by simply counting edges.) Furthermore, Andoni et al. [9] showed that any sketch for all the cuts in a graph must have size $\Omega(n)$.

While these lower bounds show that $O(n)$ space is the best possible for approximating problems like MaxCut in general, the constructions used in these bounds are quite specialized. In particular, the graphs involved are sparse, i.e., have $\Theta(n)$ edges. Meanwhile, as we mentioned above, if a graph is *dense* ($\Omega(n^2)$ edges), random sampling is known to give $O(1)$ space and time algorithms. The question we study in this paper is if there is a middle ground: can we get truly sublinear (i.e., $o(n)$) algorithms for natural graph problems in between (easy) dense graphs and (hard) sparse graphs?

Our main contribution is to answer this in the affirmative. As long as a graph has average degree n^δ for some $\delta > 0$, truly sub-linear space $(1+\epsilon)$ approximation algorithms are possible for problems such as MaxCut and correlation clustering.[1] Indeed, we show that a biased sample of vertices forms a "core-set" for these problems. A core-set for an optimization problem (see [1]), is a subset of the input with the property that a solution to the subset provides an approximation to the solution on the entire input.

Our arguments rely on understanding the following fundamental question: given a graph G, is the induced subgraph on a random subset of vertices a core-set for problems such as MaxCut? This question of sub-sampling and its effect on the value of an optimization problem is well studied. Results from property testing imply that a uniformly random sample of constant size suffices for many problems on *dense* graphs. [16, 6] generalized these results to the case of arbitrary k-CSPs. More recently, [12], extending a result in [14], studied the setting closest to ours. For graphs, their results imply that when the maximum and minimum degrees are both $\Theta(n^\delta)$, then a random induced subgraph with $\widetilde{O}(n^{1-\delta})$ acts as a core-set for problems such as MaxCut. Moreover, they showed that for certain lifted relaxations, subsampling does *not* preserve the value of the objective. Finally, using more modern techniques, [31] showed that the *cut norm* of a matrix (a quantity related to the MaxCut) is preserved up to a constant under random sampling, improving on [16, 6]. While powerful, we will see that these results are not general enough for our setting. Thus we propose a new, conceptually simple technique to analyze sub-sampling, and present it in the context of MaxCut and correlation clustering.

1.1 Our Results

As outlined above, our main result is to show that there exist core-sets of size $\widetilde{O}(n^{1-\delta})$ for MaxCut and correlation clustering for graphs with $\Omega(n^{1+\delta})$ edges (where $0 < \delta \leq 1$). This

[1] We consider the max-agreement version of correlation clustering (see Section 2).

then leads to a two-pass streaming algorithm for MaxCut and correlation clustering on such graphs, that uses $\widetilde{O}(n^{1-\delta})$ space and produces a $1+\varepsilon$ approximation.

This dependence of the core-set size on δ is optimal up to logarithmic factors, by a result of [15]. Specifically, [15] showed that any $(1+\varepsilon)$ approximation algorithm for MaxCut on graphs of average degree n^{δ} must have running time $2^{\Omega(n^{1-\delta})}$, assuming the exponential time hypothesis (ETH). Since a core-set of size $o(n^{1-\delta})$ would trivially allow such an algorithm (we can perform exhaustive search over the core-set), our construction is optimal up to a logarithmic factor, assuming ETH.

Our streaming algorithm for correlation clustering can be viewed as improving the semi-streaming (space $\tilde{O}(n)$) result of Ahn et al. [4], while using an additional pass over the data. Also, in the context of the lower bound of Andoni et al. [9], our result for MaxCut can be interpreted as saying that while a sketch that approximately maintains *all* cuts in a graph requires an $\Omega(n)$ size, one that preserves the MaxCut can be significantly smaller, when the graph has a polynomial average degree.

At a technical level, we analyze the effect of sampling on the value of the MaxCut and correlation clustering objectives. As outlined above, several techniques are known for such an analysis, but we give a new and conceptually simple framework that (a) allows one to analyze *non-uniform* sampling for the first time, and (b) gets over the assumptions of near-regularity (crucial for [14, 12]) and density (as in [16, 6]). We expect the ideas from our analysis to be applicable to other settings as well, especially ones for which the 'linearization' framework of [10] is applicable.

The formal statement of results, an outline of our techniques and a comparison with earlier works are presented in Section 3.

1.2 Related Work

MaxCut and correlation clustering are both extremely well-studied problems, and thus we will only mention the results most relevant to our work.

Dense graphs. A graph is said to be dense if its average degree is $\Omega(n)$. Starting with the work of Arora et al. [10], many NP hard optimization problems have been shown to admit a PTAS when the instances are dense. Indeed, a small random induced subgraph is known to be a core-set for problems such as MaxCut, and indeed all k-CSPs [20, 6, 16, 29]. The work of [10] relies on an elegant *linearization* procedure, while [16, 6] give a different (and more unified) approach based on "cut approximations" of a natural tensor associated with a CSP.

Polynomial density. The focus of our work is on graphs that are *in between* sparse (constant average degree) and dense graphs. These are graphs whose density (i.e., average degree) is n^{δ}, for some $0 < \delta < 1$. Fotakis et al. [15] extended the approach of [10] to this setting, and obtained $(1+\varepsilon)$ approximation algorithms with run-time $\exp(\widetilde{O}(n^{1-\delta}))$. They also showed that it was the best possible, under the exponential time hypothesis (ETH). By way of core-sets, in their celebrated work on the optimality of the Goemans-Williamson rounding, Feige and Schechtman [14] showed that a random sample of $\widetilde{O}(n^{1-\delta})$ is a core-set for MaxCut, if the graphs are *almost regular* and have an average degree n^{δ}. This was extended to other CSPs by [12]. These arguments seem to use near-regularity in a crucial way, and are based on restricting the number of possible 'candidates' for the maximum cut.

Streaming algorithms and lower bounds. In the streaming setting, there are several algorithms [2, 27, 18, 3, 19, 26] that produce cut or spectral sparsifiers with $O(\frac{n}{\epsilon^2})$ edges using $\tilde{O}(\frac{n}{\epsilon^2})$ space. Such algorithms preserve every cut within $(1+\epsilon)$-factor (and therefore

also preserve the max cut). Andoni et al. [9] showed that such a space complexity is essential; in fact, [9] show that any sketch for all the cuts in a graph must have bit complexity $\Omega(\frac{n}{\epsilon^2})$ (not necessarily streaming ones). However, this does not rule out the possibility of being able to find a *maximum* cut in much smaller space.

For MaxCut, Kapralov et al. [24] and independently Kogan et al. [28] proved that any streaming algorithm that can approximate the MaxCut value to a factor better than 2 requires $\tilde{\Omega}(\sqrt{n})$ space, even if the edges are presented in random order. For adversarial orders, they showed that for any $\epsilon > 0$, a one-pass $(1+\epsilon)$-approximation to the max cut value must use $n^{1-O(\epsilon)}$ space. Very recently, Kapralov et al. [25] went further, showing that there exists an $\epsilon^* > 0$ such that every randomized single-pass streaming algorithm that yields a $(1+\epsilon^*)$-approximation to the MAXCUT size must use $\Omega(n)$ space.

Correlation clustering. Correlation clustering was formulated by Bansal et al. [11] and has been studied extensively. There are two common variants of the problem – maximizing agreement and minimizing disagreement. While these are equivalent for exact optimization (their sum is a constant), they look very different under an approximation lens. Maximizing agreement typically admits constant factor approximations, but minimizing disagreement is much harder. In this paper, we focus on the maximum-agreement variant of correlation clustering and in particular we focus on $(1+\epsilon)$-approximations. Here, Ailon and Karnin [5] presented an approximation scheme with sublinear query complexity (which also yields a semi-streaming algorithm) for dense instances of correlation clustering. Giotis and Guruswami [17] described a sampling based algorithm combined with a greedy strategy which guarantees a solution within (ϵn^2) additive error. (Their work is similar to the technique of Mathieu and Schudy [29].) Most recently, Ahn et al. [4] gave a single-pass semi-streaming algorithm for max-agreement. For bounded weights, they provide an $(1+\epsilon)$-approximation streaming algorithm and for graphs with arbitrary weights, they present a $0.766(1-\epsilon)$-approximation algorithm. Both algorithms require $(n\epsilon^{-2})$ space. The key idea in their approach was to adapt multiplicative-weight-update methods for solving the natural SDPs for correlation clustering in a streaming setting using linear sketching techniques.

2 Definitions

▶ **Definition 1** (MaxCut). Let $G = (V, E, w)$ be a graph with weights $w\colon E \to \mathbb{R}^+$. Let (A, B) be a partition of V and let $w(A, B)$ denote the sum of weights of edges between A and B. Then $\text{MaxCut}(G) = \max_{(A,B) \text{ partition of } V} w(A, B)$.

For ease of exposition, we will assume that the input graph for MaxCut is unweighted. Our techniques apply as long as all the weights are $O(1)$. Also, we denote by Δ the average degree, i.e., $\sum_{i,j} w_{ij}/|V|$.

Moving now to correlation clustering, let $G = (V, E, c^+, c^-)$ be a graph with edge weights c^+_{ij} and c^-_{ij} where for every edge ij we have $c^+_{ij}, c^-_{ij} \geq 0$ and only one of them is nonzero. For every edge $ij \in E$, we define $\eta_{ij} = c^+_{ij} - c^-_{ij}$ and for each vertex, $d_i = \sum_{i \in \Gamma(j)} |\eta_{ij}|$. We will also assume that all the weights are bounded by an absolute constant in magnitude (for simplicity, we assume it is 1). We define the "average degree" Δ (used in the statements that follow) of a correlation clustering instance to be $(\sum_i d_i)/n$.

▶ **Definition 2** (MAX-AGREE correlation clustering). Given $G = (V, E, c^+, c^-)$ as above, consider a partition of V into *clusters* $C_1, C_2, \ldots, C_k$, and let χ_{ij} be an indicator that is 1 if i an j are in the same cluster and 0 otherwise. The MAX-AGREE score of this clustering is given by $\sum_{ij} c^+_{ij}\chi_{ij} + \sum_{ij} c^-_{ij}(1-\chi_{ij})$. The goal is to find a partition *maximizing* this score. The maximum value of the score over all partitions of V will be denoted by $CC(G)$.

Note that the objective value can be simplified to $\sum_{ij} c_{ij}^- + \eta_{ij}\chi_{ij} = C^- + \sum_{ij} \eta_{ij}\chi_{ij}$, where C^- denotes the sum $\sum_{ij} c_{ij}^-$.

We will also frequently use concentration bounds; these are stated in full version of the paper.

3 Technical overview

We now present an outline of our main ideas. Suppose we have a graph $G = (V, E)$. First, we define a procedure VERTEX SAMPLE. This takes as input probabilities p_i for every vertex, and produces a random weighted induced subgraph.

Procedure vertex sample ($\{p_i\}_{i\in V}$). Sample a set S' of vertices by selecting each vertex v_i with probability p_i independently. Define H to be the induced subgraph of G on the vertex set S'. For $i, j \in S'$, define $w_{ij} = \frac{1}{p_i p_j \Delta^2}$.[2]

Intuitively, the edge weights are chosen so that the total number of edges remains the same, in expectation. Next, we define the notion of an importance score for vertices. Let d_i denote the degree of vertex i.

▶ **Definition 3.** The *importance score* h_i of a vertex i is defined as $h_i = \min\{1, \frac{\max\{d_i, \epsilon\Delta\}}{\Delta^2 \alpha_\epsilon}\}$, where α_ϵ is an appropriately chosen parameter (for MAXCUT, we set it to $\frac{\epsilon^4}{C \log n}$, and for correlation clustering, we set it to $\frac{\epsilon^8}{C \log n}$, where C is an absolute constant).

The main result is now the following:

▶ **Theorem 4** (Core-set). *Let $G = (V, E)$ have an average degree Δ. Suppose we apply* VERTEX SAMPLE *with probabilities $p_i \in [h_i, 2h_i]$ to obtain a weighted graph H. Then H has $\widetilde{O}(\frac{n}{\Delta})$ vertices and the quantities* MAXCUT(H) *and $CC(H)$ are within a $(1+\epsilon)$ factor of the corresponding quantities* MAXCUT(G) *and $CC(G)$, w.p. at least $1 - \frac{1}{n^2}$.*

While the number of vertices output by the vertex sample procedure is small, we would like a core-set of small "total size". This is ensured by the following.

Procedure edge sample (H). Given a weighted graph H with total edge weight W, sample each edge $e \in E(H)$ independently with probability $p_e := \min(1, \frac{8|S'|w_e}{\varepsilon^2 W})$, to obtain a graph H'. Now, assign a weight w_e/p_e to the edge e in H'.

The procedure samples roughly $|S'|/\varepsilon^2$ edges, with probability proportional to the edge weights. The graph is then re-weighted in order to preserve the total edge weight in expectation, yielding:

▶ **Theorem 5** (Sparse core-set). *Let G be a graph with n vertices and average degree $\Delta = n^\delta$. Let H' be the graph obtained by first applying* VERTEX SAMPLE *and then applying* EDGE SAMPLE. *Then H' is a ϵ-core-set for* MAXCUT *and CC, having size $\widetilde{O}(\frac{n}{\Delta}) = \widetilde{O}(n^{1-\delta})$.*

We then show how to implement the above procedures in a streaming setting. This gives:

▶ **Theorem 6** (Streaming algorithm). *Let G be a graph on n vertices and average degree $\Delta = n^\delta$, whose edges arrive in a streaming fashion in adversarial order. There is a two-pass streaming algorithm with space complexity $\widetilde{O}(\frac{n}{\Delta}) = \widetilde{O}(n^{1-\delta})$ for computing a $(1+\epsilon)$-approximation to* MAXCUT(G) *and $CC(G)$.*

[2] In correlation clustering, we have edge weights to start with, so the weight in H will be $w_{ij} \cdot c_{ij}^+$ (or c_{ij}^-).

Of these, Theorem 4 is technically the most challenging. Theorem 5 follows via standard edge sampling methods akin to those in [2] (which show that w.h.p., every cut size is preserved). It is presented in full version of the paper, for completeness. The streaming algorithm, and a proof of Theorem 6, are presented in Section 6. In the following section, we give an outline of the proof of Theorem 4.

3.1 Proof of the sampling result (theorem 4): an outline

In this outline we will restrict ourselves to the case of MaxCut as it illustrates our main ideas. Let G be a graph as in the statement of the theorem, and let H be the output of the procedure VERTEX SAMPLE.

Showing that $\textsc{MaxCut}(H)$ is at least $\textsc{MaxCut}(G)$ up to an $\varepsilon n\Delta$ additive term is easy. We simply look at the projection of the maximum cut in G to H (see, for instance, [14]). Thus, the challenge is to show that a sub-sample cannot have a significantly larger cut, w.h.p. The natural approach of showing that every cut in G is preserved does not work as 2^n cuts is too many for the purposes of a union bound.

There are two known ways to overcome this. The first approach is the one used in [20, 14] and [12]. These works essentially show that in a graph of average degree Δ, we need to consider only roughly $2^{n/\Delta}$ cuts for the union bound. If all the degrees are roughly Δ, then one can show that all these cuts are indeed preserved, w.h.p. There are two limitations of this argument. First, for non-regular graphs, the variance (roughly $\sum_i pd_i^2$, where p is the sampling probability) can be large, and we cannot take a union bound over $\exp(n/\Delta)$ cuts. Second, the argument is combinatorial, and it seems difficult to generalize this to analyze non-uniform sampling.

The second approach is via *cut decompositions*, developed in [16, 6]. Here, the adjacency matrix A is decomposed into $\text{poly}(1/\varepsilon)$ rank-1 matrices, plus a matrix that has a small *cut norm*. It turns out that solving many quadratic optimization problems (including MaxCut) on A is equivalent (up to an additive $\varepsilon n\Delta$) to solving them over the sum of rank-1 terms (call this A'). Now, the adjacency matrix of H is an induced square sub-matrix of A, and since we care only about A' (which has a simple structure), [6] could show that $\textsc{MaxCut}(H) \leq \textsc{MaxCut}(G) + \varepsilon n^2$, w.h.p. To the best of our knowledge, such a result is not known in the "polynomial density" regime (though the cut decomposition still exists).

Our technique. We consider a new approach. While inspired by ideas from the works above, it also allows us to reason about non-uniform sampling in the polynomial density regime. Our starting point is the result of Arora et al. [10], which gives a method to estimate the MaxCut using a collection of linear programs (which are, in turn, derived using a sample of size n/Δ). Now, by a double sampling trick (which is also used in the approaches above), it turns out that showing a sharp concentration bound for the value of an *induced sub-program* of an LP as above, implies Theorem 4. As it goes via a linear programming and not a combinatorial argument, analyzing non-uniform sampling turns out to be quite direct. Let us now elaborate on this high level plan.

Induced sub-programs. First, we point out that an analysis of induced sub-programs is also an essential idea in the work of [6]. The main difference is that in their setting, only the variables are sub-sampled (and the number of constraints remains the same). In our LPs, the constraints correspond to the vertices, and thus there are fewer constraints in the sampled LP. This makes it harder to control the value of the objective. At a technical level,

while a duality-based argument using Chernoff bounds for linear forms suffices in the setting of [6], we need the more recent machinery on concentration of quadratic functions.

We start by discussing the estimation technique of [10].

Estimation with Linear Programs. The rough idea is to start with the natural quadratic program for MaxCut: $\max \sum_{(i,j)\in E} x_i(1-x_j)$, subject to $x_i \in \{0,1\}$.[3] This is then "linearized" using a seed set of vertices sampled from G. We refer to Section 4 for details. For now, $\textsc{Est}(G)$ is a procedure that takes a graph G and a set of probabilities $\{\gamma_i\}_{i\in V(G)}$, samples a seed set using γ, and produces an estimate of MaxCut (G).

Now, suppose we have a graph G and a sample H. We can imagine running $\textsc{Est}(G)$ and $\textsc{Est}(H)$ to obtain good estimates of the respective MaxCut values. But now suppose that in both cases, we could use precisely the same seed set. Then, it turns out that the LPs used in $\textsc{Est}(H)$ would be 'induced' sub-programs (in a sense we will detail in Section 5) of those used in $\textsc{Est}(G)$, and thus proving Theorem 4 reduces to showing a sufficiently strong concentration inequality for sub-programs.

The key step above was the ability to use same seed set in the Est procedures. This can be formalized as follows.

Double sampling. Consider the following two strategies for sampling a *pair* of subsets (S, S') of a universe $[n]$ (here, $q_v \le p_v$ for all v):

- Strategy A: choose $S' \subseteq [n]$, by including every v w.p. p_v, independently; then for $v \in S'$, include them in S w.p. q_v/p_v, independently.
- Strategy B: pick $S \subseteq [n]$, by including every v w.p. q_v; then iterate over $[n]$ once again, placing $v \in S'$ with a probability equal to 1 if $v \in S$, and p_v^* if $v \notin S$.

▶ **Lemma 7.** *Suppose* $p_v^* = p_v(1 - \frac{q_v}{p_v})/(1-p_v)$. *Then the distribution on pairs* (S, S') *obtained by strategies A and B are identical.*

The proof is by a direct calculation and the details can be found in full version.

Proof of Theorem 4. To show the theorem, we use p_v as in the statement of the theorem, and set q to be the uniform distribution $q_v = \frac{16 \log n}{\varepsilon^2 \Delta}$. The proof now proceeds as follows. Let S' be a set sampled using the probabilities p_v. These form the vertex set of H. Now, the procedure Est on H (with sampling probabilities q_v/p_v) samples the set S (as in strategy A). By the guarantee of the estimation procedure (Corollary 5.1.2 in full version), we have $\textsc{MaxCut}(H) \approx \textsc{Est}(H)$, w.h.p. Next, consider the procedure Est on G with sampling probabilities q_v. Again, by the guarantee of the estimation procedure (Corollary 5.1.1), we have $\textsc{MaxCut}(G) \approx \textsc{Est}(G)$, w.h.p.

Now, we wish to show that $\textsc{Est}(G) \approx \textsc{Est}(H)$. By the equivalence of the sampling strategies, we can now take the strategy B view above. This allows us to assume that the Est procedures use the same S, and that we pick S' *after* picking S. This reduces our goal to one of analyzing the value of a random induced sub-program of an LP, as mentioned earlier. The details of this step are technically the most involved, and are presented in Section 5. This completes the proof of the theorem. (Note that the statement also includes a bound on the number of vertices of H. This follows immediately from the choice of p_v.) ◀

[3] This is a valid formulation, because for every $x_i \neq x_j$ that is an edge contributes 1 to the objective, and $x_i = x_j$ contribute 0.

4 Estimation via linear programming

We now present the estimation procedure Est used in our proof. It is an extension of [10] to the case of weighted graphs and non-uniform sampling probabilities.

Let $H = (V, E, w)$ be a weighted, undirected graph with edge weights w_{ij}, and let $\gamma : V \to [0, 1]$ denote sampling probabilities. The starting point is the quadratic program for MaxCut: max $\sum_{ij \in E} w_{ij} x_i (1 - x_j)$, subject to $x_i \in \{0, 1\}$. The objective can be re-written as $\sum_{i \in V} x_i (d_i - \sum_{j \in \Gamma(i)} w_{ij} x_j)$, where d_i is the weighted degree, $\sum_{j \in \Gamma(i)} w_{ij}$. The key idea now is to "guess" the value of $\rho_i := \sum_{j \in \Gamma(i)} w_{ij} x_j$, by using a seed set of vertices. Given a guess, the idea is to solve the following linear program, which we denote by $LP_\rho(V)$.

$$\begin{aligned} \text{maximize} \quad & \sum_i x_i (d_i - \rho_i) - s_i - t_i \\ \text{subject to} \quad & \rho_i - t_i \le \sum_{j \in \Gamma(i)} w_{ij} x_j \le \rho_i + s_i \\ 0 \le x_i \le 1, \qquad & s_i, t_i \ge 0. \end{aligned}$$

The variables are x_i, s_i, t_i. Note that if we fix the x_i, the optimal s_i, t_i will satisfy $s_i + t_i = |\rho_i - \sum_{j \in \Gamma(i)} w_{ij} x_j|$. Also, note that if we have a perfect guess for ρ_i's (coming from the MaxCut), the objective can be made $\ge$ MaxCut(H).

Estimation procedure. The procedure Est is the following: first sample a set $S \subseteq V$ where each $i \in V$ is included w.p. γ_i independently. For every partition $(A, S \setminus A)$ of S, set $\rho_i = \sum_{j \in \Gamma(i) \cap A} \frac{w_{ij}}{\gamma_j}$, and solve $LP_\rho(V)$ (in what follows, we denote this LP by $LP^\gamma_{A, S \setminus A}(V)$, as this makes the partition and the sampling probabilities clear). Return the maximum of the objective values.

Our result here is a sufficient condition for having Est$(H) \approx$ MaxCut(H).

▶ **Theorem 8.** *Let H be a weighted graph on n vertices, with edge weights w_{ij} that add up to W. Suppose the sampling probabilities γ_i satisfy the condition*

$$w_{ij} \le \frac{W \varepsilon^2}{8 \log n} \frac{\gamma_i \gamma_j}{\sum_u \gamma_u} \quad \text{for all } i, j. \tag{1}$$

Then, we have Est$(H, \gamma) \in$ MaxCut$(H) \pm \varepsilon W$*, with probability at least $1 - 1/n^2$ (where the probability is over the random choice of S).*

The proof consists of claims showing the upper and lower bound separately. The technical details are presented in full version. Finally, to show Theorem 4 (as outlined in Section 3.1), we need to apply Theorem 8 with specific values for γ and w_{ij}. The related corollaries are stated in full version of the paper.

5 Random induced linear programs

We will now show that the Est on H has approximately the same value as the estimate on G (with appropriate γ values). First, note that Est(G) is $\max_{A \subseteq S} LP^\gamma_{A, S \setminus A}(G)$, where $\gamma_i = q_i$. To write the LP, we need the constants ρ_i, defined by the partition $(A, S \setminus A)$ as $\rho_i := \sum_{j \in \Gamma(i) \cap A} \frac{1}{q_j}$. For the graph H, the estimation procedure uses an identical program, but the sampling probabilities are now $\alpha_i := q_i / p_i$, and the estimates ρ, which we now denote by $\widetilde{\rho}_i$, are defined by $\widetilde{\rho}_i := \sum_{j \in \Gamma(i) \cap A} \frac{p_j w_{ij}}{q_j}$. Also, by the way we defined w_{ij}, $\widetilde{\rho}_i = \frac{\rho_i}{p_i \Delta^2}$.

$$\begin{aligned}
\max \quad & \sum_{i\in G}[x_i(d_i-\rho_i)-(s_i+t_i)] \\
\text{s.t.} \quad & \sum_{j\in\Gamma(i)} x_j \le \rho_i + s_i, \quad \forall i\in[n] \\
& -\sum_{j\in\Gamma(i)} x_j \le -\rho_i + t_i, \quad \forall i\in[n] \\
& 0\le x_i\le 1 \quad \forall i\in[n]
\end{aligned}$$

(a) The LP on the full graph

$$\begin{aligned}
\max \quad & \sum_{i\in S'}[x_i(\widetilde{d}_i-\widetilde{\rho}_i)-(\tilde{s}_i+\tilde{t}_i)] \\
\text{s.t.} \quad & \sum_{j\in\Gamma(i)\cap S'} w_{ij}x_j \le \widetilde{\rho}_i + \tilde{s}_i, \quad \forall i\in S' \\
& -\sum_{j\in\Gamma(i)\cap S'} w_{ij}x_j \le -\widetilde{\rho}_i + \tilde{t}_i, \quad \forall i\in S' \\
& 0\le x_i\le 1, \quad \tilde{s}_i,\tilde{t}_i\ge 0 \quad \forall i\in S'
\end{aligned}$$

(b) The sampled LP

Figure 1 The two LPs.

$$\begin{aligned}
\text{minimize} \quad & \sum_{i\in G} u_i+\rho_i z_i \quad \text{s.t.} \\
& u_i + \sum_{j\in\Gamma(i)} z_j \ge d_i-\rho_i \quad \forall i\in V \\
& u_i\ge 0, \quad -1\le z_i\le 1 \quad \forall i\in V
\end{aligned}$$

(a) The dual of $LP^{\gamma}_{A,S\setminus A}(G)$

$$\begin{aligned}
\text{minimize} \quad & \sum_{i\in S'}[\tilde{u}_i+\widetilde{\rho}_i\tilde{z}_i] \quad \text{s.t.} \\
& \tilde{u}_i + \sum_{j\in\Gamma(i)\cap S'} w_{ij}\tilde{z}_j \ge \widetilde{d}_i-\widetilde{\rho}_i \quad \forall i\in S' \\
& \tilde{u}_i\ge 0, \quad -1\le \tilde{z}_i\le 1 \quad \forall i\in S'.
\end{aligned}$$

(b) The dual of the induced program $LP^{\alpha}_{A,S\setminus A}(H)$.

Figure 2 The dual LPs.

The degrees are now $\widetilde{d}_i := \sum_{j\in\Gamma(i)\cap S'} w_{ij} = \sum_{j\in\Gamma(i)\cap S'} \frac{1}{p_i p_j \Delta^2}$. The two LPs are shown in Figure 1. Our aim in this section is to show the following:

▶ **Theorem 9.** *Let G be an input graph, and let (S,S') be sampled as described in Section 3.1. Then, with probability $\ge 1-\frac{1}{n^2}$, we have*

$$\max_{A\subseteq S} LP^{\gamma}_{A,S\setminus A}(G) \ge \Delta^2 \cdot \max_{A\subseteq S} LP^{\alpha}_{A,S\setminus A}(H) - \varepsilon n\Delta.$$

Proof outline. To prove the theorem, the idea is to take the "strategy B" viewpoint of sampling (S,S'), i.e., fix S, and sample S' using the probabilities p^*. Then, we only need to understand the behavior of an "induced sub-program" sampled with the probabilities p^*. This is done by considering the duals of the LPs, and constructing a feasible solution to the induced dual whose cost is not much larger than the dual of the full program, w.h.p. This implies the result, by linear programming duality.

Let us thus start by understanding the dual of $LP^{\gamma}_{A,S\setminus A}(G)$ given A, shown in Figure 2a. We note that for any given z, the optimal choice of u_i is $\max\{0, d_i-\rho_i-\sum_{j\in\Gamma(i)} z_j\}$; thus we can think of the dual solution as being the vector z. The optimal u_i may thus be bounded by $2d_i$, a fact that we will use later. Next, we write down the dual of the induced program, $LP^{\alpha}_{A,S\setminus A}(H)$, as shown in Figure 2b. Following the outline above, we will construct a feasible solution to LP (2b), whose cost is close to the optimal dual solution to LP (2a). The construction we consider is very simple: if z is the optimal dual solution to (2a), we set $\tilde{z}_i = z_i$ for $i\in S'$ as the candidate solution to (2b). This is clearly feasible, and thus we only

need to compare the solution costs. The dual objective values are as follows

$$\mathsf{Dual}_G = \sum_{i \in V} \rho_i z_i + \max\{0,\ d_i - \rho_i - \sum_{j \in \Gamma(i)} z_j\} \tag{2}$$

$$\mathsf{Dual}_H \leq \sum_{i \in S'} \widetilde{\rho}_i z_i + \max\{0,\ \widetilde{d}_i - \widetilde{\rho}_i - \sum_{j \in \Gamma(i) \cap S'} w_{ij} z_j\} \tag{3}$$

Note that there is a $\leq$ in (3), as $\tilde{z}_i = z_i$ is simply one feasible solution to the dual (which is a minimization program). Next, our goal is to prove that w.p. at least $1 - \frac{1}{n^2}$,

$$\max_{A \subseteq S} \mathsf{Dual}_H \leq \frac{1}{\Delta^2} \cdot \max_{A \subseteq S} \mathsf{Dual}_G + \frac{\varepsilon n}{\Delta}.$$

Note that here, the probability is over the choice of S' *given* S (as we are taking view-B of the sampling). The first step in proving the above is to move to a slight variant of the quantity Dual_H, which is motivated by the fact that $\mathbf{Pr}[Y_i = 1]$ is not quite p_i, but p_i^* (as we have conditioned on S). Let us define $\widetilde{\rho}_i^* := \frac{\rho_i}{p_i^* \Delta^2}$ (recall that $\widetilde{\rho}_i$ is $\frac{\rho_i}{p_i \Delta^2}$), and $w_{ij}^* := \frac{1}{p_i^* p_j^* \Delta^2}$. So also, let $d_i^* := \sum_{j \in \Gamma(i)} Y_j w_{ij}^*$. Then, define

$$\mathsf{Dual}_H^* := \sum_{i \in S'} \widetilde{\rho}_i^* z_i + \max\{0,\ \widetilde{d}_i^* - \widetilde{\rho}_i^* - \sum_{j \in \Gamma(i) \cap S'} w_{ij}^* z_j\}. \tag{4}$$

A straightforward lemma is as follows. The proof is presented in full version of the paper.

▶ **Lemma 10.** *Let (S, S') be sampled as in Section 3.1. Then w.p. at least $1 - \frac{1}{n^4}$, we have that for all $z \in [-1, 1]^n$ and for all partitions $(A, S \setminus A)$ of S,*[4] *$|\mathsf{Dual}_H - \mathsf{Dual}_H^*| \leq \frac{\varepsilon n}{2\Delta}$.*

Thus our goal is to show the following (where condition (b) on S is a technical one needed to bound ρ_i).

▶ **Lemma 11.** *Let S satisfy the conditions (a) $|S| \leq \frac{20 n \log n}{\varepsilon^2 \Delta}$, and (b) for all $i \in V$, $\sum_{j \in \Gamma(i) \cap S} \frac{1}{q_j} \leq 2(d_i + \varepsilon\Delta)$. Then, w.p. $\geq 1 - \frac{1}{n^4}$ over the choice of S' given S, we have*

$$\max_{A \subseteq S} \mathsf{Dual}_H^* \leq \frac{1}{\Delta^2} \cdot \max_{A \subseteq S} \mathsf{Dual}_G + \frac{\varepsilon n}{2\Delta}.$$

Proof of Theorem 9. The conditions we assumed on S in Lemma 11 hold w.p. at least $1 - \frac{1}{n^4}$ (via a simple application of Bernstein's inequality). Thus the conclusion of the lemma holds w.p. at least $1 - \frac{2}{n^4}$. Combining this with Lemma 10, we have that $\max_A \mathsf{Dual}_H \leq \frac{1}{\Delta^2} \max_A \mathsf{Dual}_G + \frac{\varepsilon n}{\Delta}$ w.p. at least $1 - \frac{3}{n^4}$. The theorem then follows via LP duality. ◀

It thus suffices to prove Lemma 11 via a concentration bound on a quadratic function that is not quite a quadratic *form*. Details can be found in full version of the paper.

6 A 2-pass streaming algorithm

We now show how our main core set result can be used to design a streaming algorithm for MaxCut. The algorithm works in two passes: the first pass builds a core-set S of size $\widetilde{O}(n/\Delta)$ as prescribed by Theorem 4 and the second pass builds the induced weighted graph $G[S]$ and computes its max cut. This algorithm works under edge insertion/deletion.

[4] Note that the partition defines the ρ_i.

6.1 Pass 1: Building a core set

To construct S, Theorem 4 states that each vertex must be sampled with probability p_i, where $p_i \geq h_i$ and $h_i = \min(1, \frac{\max(d_i, \epsilon\Delta)}{\Delta^2 \alpha_\epsilon})$ is the importance score of a vertex. As the goal is to only choose a small number of vertices, we will also make sure that $p_i \leq 2h_i$. The challenge here is two-fold: we need to sample (roughly) proportional to the degree d_i, which can only be computed *after* the stream has passed, and we also need the actual value of p_i (or a close enough estimate of it) in order to correctly reweight edges in the second pass.

The degree d_i of a vertex i is the "count" of the number of times i appears in the edge stream. To sample with probability proportional to d_i we will therefore make use of streaming algorithms for ℓ_1-sampling [30, 8, 21]. We borrow some notation from [8].

▶ **Definition 12.** Let $\rho > 0, f \in [1, 2]$. A (ρ, f)-approximator to $\tau > 0$ is a quantity $\hat{\tau}$ such that $\tau/f - \rho \leq \hat{\tau} \leq f\tau + \rho$

▶ **Lemma 13** ([21] (rephrased from [8])). *Given a vector $x \in \mathbb{R}^n$ and parameters $\epsilon, \delta > 0, c > 0$ there exists an algorithm A that uses space $O(\log(1/\epsilon)\epsilon^{-1} \log^2 n \log(1/\delta))$ and generates a pair (i, v) from a distribution D_x on $[1 \ldots n]$ such that with probability $1 - \delta$*

- *$D_x(i)$ is a $(\frac{1}{n^c}, 1 + \epsilon)$-approximator to $|x_i|/\|x\|_1$*
- *v is a $(0, 1 + \epsilon)$-approximator to x_i*

where c is a fixed constant.

We will also need to maintain heavy hitters: all vertices of degree at least Δ^2 (up to constants). To do this, we will make use of the standard CountMin sketch [13]. For completeness, we state its properties here.

▶ **Lemma 14** ([13]). *Fix parameters $k, \delta > 0$. Then given a stream of m updates to a vector $x \in \mathbb{R}^n$ there is a sketch CM of size $O(k \log \delta^{-1}(\log m + \log n))$ and a reconstruction procedure $f : [n] \to \mathbb{R}$ such that with probability $1 - \delta$, for any x_i, $|x_i - f(i)| \leq \|x\|_1/k$*

Outline. We will have a collection of r, roughly n/Δ ℓ_1-samplers. These samplers will together give a good estimate ($(1+\varepsilon)$-approximation) of the importance h_i for all the vertices that have a *small* degree (which we define to be $< \alpha_\varepsilon \Delta^2$). Then, we use the CM sketch to maintain the degrees of all the 'high degree' vertices, i.e., those with degrees $\geq \alpha_\varepsilon \Delta^2$. Taken together, we obtain the desired sampling in the first pass.

▶ **Definition 15.** Given two sets of pairs of numbers S, S', let

$$S \cup_{\max} S' = \{(x, \max_{(x',y) \in S \cup S', x' = x} y)\}.$$

▶ **Lemma 16.** *Let $S = \{(i, v_i)\}$ be the set returned by Algorithm 1. Then*

- *S has size $\widetilde{O}\left(\frac{n}{\Delta}\right)$.*
- *Each $i \in [n]$ is sampled with probability p_i that is $(0, 1 + \epsilon)$ approximated by v_i and that $(n^{-c}, 1 + \epsilon)$-approximates h_i.*

Proof sketch. Consider any vertex i with $d_i \geq \Delta^2 \alpha_\epsilon$. By Lemma 14, such a vertex will report a count of at least $f(i) = (1 - \zeta)\Delta^2 \alpha_\epsilon$ and thus is guaranteed to be included in S_h. Its reported score $v_i = 1$ satisfies the requirement of the Lemma. Secondly, consider any vertex with degree $d_i < \epsilon\Delta$. For such a vertex, $h_i = \epsilon/\Delta\alpha_\epsilon$ and thus it is included in S_l with the desired probability and v_i.

Finally, consider a vertex i with $\epsilon\Delta \leq d_i < \Delta^2 \alpha_\epsilon$. The probability that none of the ℓ_1-samplers yield i is $(1 - d_i/n\Delta)^r$, and since $d_i/n\Delta \ll 1$, this can be approximated as $(1 - rd_i/n\Delta)$. Thus, the probability of seeing i is $rd_i/n\Delta = d_i/\Delta^2 \alpha_\epsilon$ as desired. ◀

Algorithm 1 Given n and average degree Δ

Initialize $S_l, S_m, S_h \leftarrow \emptyset$, and $\zeta = \alpha_\varepsilon$.
Sample elements from $[1 \dots n]$ each with probability $\epsilon/\Delta\alpha_\epsilon$. For each sampled i add $(i, \epsilon/\Delta\alpha_\epsilon)$ to S_l.
Fix $\zeta > 0$. Initialize a CountMin sketch CM with size parameter $k = n/\Delta\zeta^2$. Let f be the associated reconstruction procedure.
Initialize $r = O(n/\Delta\alpha_\epsilon)$ copies $A_1 \dots A_r$ of the algorithm A from Lemma 13.
for each stream update (i, w) (a vertex to which current edge is incident, and weight) **do**
  Update each $A_j, 1 \le j \le r$. Update CM.
for $j = 1$ to r **do**
  Sample (i, v) from A_j. $S_m = S_m \cup_{\max} \{(i, v)\}$
$S_h = \{(i, 1) \mid f(i) \ge (1 - \zeta)\Delta^2\alpha_\epsilon\}$
return $S_l \cup_{\max} S_m \cup_{\max} S_h$

▶ **Corollary 17.** *For each* $(i, v_i) \in S, h_i \le v_i \le 2h_i$.

6.2 Pass 2: Building the induced weighted graph

The first pass produces a set S of $\widetilde{O}(n/\Delta)$ vertices together with estimates v_i for their importance score h_i. If we weight each edge ij in $G[S]$ by $w_{ij} = 1/v_i v_j \Delta^2$, Theorem 4, along with Corollary 17 guarantee that a MaxCut in the resulting weighted graph is a good approximation of the true max cut. Thus, knowing S, constructing the re-weighted $G[S]$ in the second pass is trivial if we had space proportional to the number of edges in $G[S]$. Unfortunately this can be quadratic in $|S|$, so our goal is to implement the edge sampling of Theorem 5 in the second pass. This is done as follows: we maintain a set of edges E'. Every time we encounter an edge ij with $i, j \in S$, we check to see if it is already in E'. If not, we toss a coin and with probability $p_{ij} = \min(1, w_{ij} \log n/\epsilon^2)$ we insert $(i, j, w_{ij}/p_{ij})$ into E'. By Lemma 7.1 in full version of the paper, the size of E' is $\widetilde{O}(n/\Delta)$, and the resulting graph yields a $(1 + \epsilon)$ approximation to the MaxCut.

7 Correlation Clustering

Our argument for correlation clustering parallels the one for MaxCut. The MAX-AGREE variant of correlation clustering, while not a CSP (as the number of clusters is arbitrary), almost behaves as one. We start with two simple observations. The first is that we can restrict the number of clusters to $1/\varepsilon$, for the purposes of a $(1 + \varepsilon)$ approximation (See full version of the paper). Next, observe that the optimum objective value is at least $\max\{C^+, C^-\} \ge n\Delta/2$. This is simply because placing all the vertices in a single cluster gives a value C^+, while placing them all in different clusters gives C^-. Thus, it suffices to focus on additive approximation of $\varepsilon n\Delta$. Once we fix the number of clusters k, we can write correlation clustering as a quadratic program in a natural way: for each vertex i, have k variables $x_{i\ell}$, which is supposed to indicate if i is given the label ℓ. We then have the constraint that $\sum_\ell x_{i\ell} = 1$ for all i. The objective function then has a clean form:

$$\sum_{ij}[\sum_{\ell=1}^{k} x_{i\ell}(1 - x_{j\ell})c^-_{ij} + x_{i\ell}x_{j\ell}c^+_{ij}] = \sum_{ij}\sum_{\ell} x_{i\ell}c^-_{ij} + x_{i\ell}x_{j\ell}\eta_{ij} = \sum_{i,\ell} x_{i\ell}(\rho_{i\ell} + d^-_i),$$

where $x_{i\ell} = 1$ iff vertex $i \in C_\ell$, and $\rho_{i\ell} = \sum_{j\in\Gamma(i)} x_{j\ell}\eta_{ij}$ and $d^-_i = \sum_j c^-_{ij}$.

Note the similarity with the program for MaxCut. We will show that the framework from Section 3.1 carries over with minor changes. The details of the new Est procedure can be found in full version of the paper (it requires one key change: we now need to consider k-partitions of the seed set in order to find ρ). The duality based proof is slightly more involved; however we can use the same rough outline. The proof is presented in full version of the paper.

References

1. Pankaj K Agarwal, Sariel Har-Peled, and Kasturi R Varadarajan. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52:1–30, 2005.
2. Kook Jin Ahn and Sudipto Guha. Graph sparsification in the semi-streaming model. In *International Colloquium on Automata, Languages, and Programming*, pages 328–338. Springer, 2009.
3. Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st symposium on Principles of Database Systems*, pages 5–14. ACM, 2012.
4. KookJin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. In *International Conference on Machine Learning*, pages 2237–2246, 2015.
5. Nir Ailon and Zohar Karnin. A note on: No need to choose: How to get both a ptas and sublinear query complexity. *arXiv preprint arXiv:1204.6588*, 2012.
6. Noga Alon, W Fernandez De La Vega, Ravi Kannan, and Marek Karpinski. Random sampling and approximation of max-csps. *Journal of computer and system sciences*, 67(2):212–243, 2003.
7. Noga Alon, Eldar Fischer, Ilan Newman, and Asaf Shapira. A combinatorial characterization of the testable graph properties: it's all about regularity. *SIAM Journal on Computing*, 39(1):143–167, 2009.
8. Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 363–372. IEEE, 2011.
9. Alexandr Andoni, Robert Krauthgamer, and David P Woodruff. The sketching complexity of graph cuts. *arXiv preprint arXiv:1403.7058*, 2014.
10. Sanjeev Arora, David Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of np-hard problems. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 284–293. ACM, 1995.
11. Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.
12. Boaz Barak, Moritz Hardt, Thomas Holenstein, and David Steurer. Subsampling mathematical relaxations and average-case complexity. In *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 512–531, Philadelphia, PA, USA, 2011. Society for Industrial and Applied Mathematics. URL: `http://dl.acm.org/citation.cfm?id=2133036.2133077`.
13. Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
14. Uriel Feige and Gideon Schechtman. On the optimality of the random hyperplane rounding technique for max cut. *Random Structures & Algorithms*, 20(3):403–440, 2002.
15. Dimitris Fotakis, Michael Lampis, and Vangelis Th Paschos. Sub-exponential approximation schemes for csps: from dense to almost sparse. *arXiv preprint arXiv:1507.04391*, 2015.

16 Alan Frieze and Ravi Kannan. The regularity lemma and approximation schemes for dense problems. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 12–20. IEEE, 1996.

17 Ioannis Giotis and Venkatesan Guruswami. Correlation clustering with a fixed number of clusters. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1167–1176. Society for Industrial and Applied Mathematics, 2006.

18 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. Graph sparsification via refinement sampling. *arXiv preprint arXiv:1004.4915*, 2010.

19 Ashish Goel, Michael Kapralov, and Ian Post. Single pass sparsification in the streaming model with edge deletions. *arXiv preprint arXiv:1203.4900*, 2012.

20 Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.

21 Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 49–58. ACM, 2011.

22 Michael Kapralov. Better bounds for matchings in the streaming model. In *SODA*, pages 1679–1697. SIAM, 2013.

23 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *SODA*, pages 734–751. SIAM, 2014.

24 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Streaming lower bounds for approximating max-cut. In *SODA*, pages 1263–1282. SIAM, 2015.

25 Michael Kapralov, Sanjeev Khanna, Madhu Sudan, and Ameya Velingker. $(1+ \omega (1))$-approximation to max-cut requires linear space. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1703–1722. SIAM, 2017.

26 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 561–570. IEEE, 2014.

27 Jonathan A Kelner and Alex Levin. Spectral sparsification in the semi-streaming setting. *Theory of Computing Systems*, 53(2):243–262, 2013.

28 Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proc. ITCS*, pages 367–376. ACM, 2015.

29 Claire Mathieu and Warren Schudy. Yet another algorithm for dense max cut: go greedy. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 176–182. Society for Industrial and Applied Mathematics, 2008.

30 Morteza Monemizadeh and David P Woodruff. 1-pass relative-error lp-sampling with applications. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1143–1160. SIAM, 2010.

31 Mark Rudelson and Roman Vershynin. Sampling from large matrices: An approach through geometric functional analysis. *J. ACM*, 54(4), 2007. doi:10.1145/1255443.1255449.

Parameterized Intractability of Even Set and Shortest Vector Problem from Gap-ETH

Arnab Bhattacharyya[1]
Indian Institute of Science, Bangalore, India
arnabb@iisc.ac.in

Suprovat Ghoshal
Indian Institute of Science, Bangalore, India
suprovat@iisc.ac.in

Karthik C. S.[2]
Weizmann Institute of Science, Rehovot, Israel
karthik.srikanta@weizmann.ac.il

Pasin Manurangsi[3]
University of California, Berkeley, USA
pasin@berkeley.edu

Abstract

The *k-Even Set* problem is a parameterized variant of the *Minimum Distance Problem* of linear codes over $\mathbb{F}_2$, which can be stated as follows: given a generator matrix $\mathbf{A}$ and an integer k, determine whether the code generated by $\mathbf{A}$ has distance at most k. Here, k is the parameter of the problem. The question of whether k-Even Set is fixed parameter tractable (FPT) has been repeatedly raised in literature and has earned its place in Downey and Fellows' book (2013) as one of the "most infamous" open problems in the field of Parameterized Complexity.

In this work, we show that k-Even Set does not admit FPT algorithms under the (randomized) Gap Exponential Time Hypothesis (Gap-ETH) [Dinur'16, Manurangsi-Raghavendra'16]. In fact, our result rules out not only exact FPT algorithms, but also any constant factor FPT approximation algorithms for the problem. Furthermore, our result holds even under the following weaker assumption, which is also known as the *Parameterized Inapproximability Hypothesis* (PIH) [Lokshtanov et al.'17]: no (randomized) FPT algorithm can distinguish a satisfiable 2CSP instance from one which is only 0.99-satisfiable (where the parameter is the number of variables).

We also consider the parameterized *k-Shortest Vector Problem (SVP)*, in which we are given a lattice whose basis vectors are integral and an integer k, and the goal is to determine whether the norm of the shortest vector (in the ℓ_p norm for some fixed p) is at most k. Similar to k-Even Set, this problem is also a long-standing open problem in the field of Parameterized Complexity. We show that, for any $p > 1$, k-SVP is hard to approximate (in FPT time) to some constant factor, assuming PIH. Furthermore, for the case of $p = 2$, the inapproximability factor can be amplified to any constant.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized Complexity, Inapproximability, Even Set, Minimum Distance Problem, Shortest Vector Problem, Gap-ETH

[1] This work was supported by Ramanujan Fellowship DSTO 1358.
[2] This work was supported by Irit Dinur's ERC-CoG grant 772839.
[3] Some part of this work was done while the author was visiting Indian Institute of Science and supported by the Indo-US Joint Center for Pseudorandomness in Computer Science.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 17; pp. 17:1–17:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.17

Related Version A full version of the paper is available at http://arxiv.org/abs/1803.09717.

Acknowledgements We are grateful to Ishay Haviv for providing insights on how the gap amplification from [26] works. PM thanks Danupon Nanongkai for introducing him to k-Even Set.

1 Introduction

The study of error-correcting codes gives rise to many computational problems. One of the most fundamental among these is the problem of computing the distance of a linear code. In this problem, which is commonly referred to as the *Minimum Distance Problem* (MDP), we are given as input a generator matrix $\mathbf{A} \in \mathbb{F}_2^{n \times m}$ of a binary[4] linear code and an integer k. The goal is to determine whether the code has distance at most k. Recall that the distance of a linear code is $\min_{\mathbf{0} \neq \mathbf{x} \in \mathbb{F}_2^m} \|\mathbf{Ax}\|_0$ where $\|\cdot\|_0$ denotes the 0-norm (aka the Hamming norm).

The study of MDP dates back to at least 1978 when Berlekamp et al. [8] conjectured that it is NP-hard. This conjecture remained open for almost two decades until it was positively resolved by Vardy [46, 47]. Later, Dumer et al. [22] strengthened this result by showing that even *approximately* computing the minimum distance of the code is hard. Specifically, they showed that, unless NP = RP, no polynomial time algorithm can distinguish between a code with distance at most k and one whose distance is greater than $\gamma \cdot k$ for any constant $\gamma \geqslant 1$. Furthermore, under stronger assumptions, the ratio can be improved to superconstants and even almost polynomial. Dumer et al.'s result has been subsequently derandomized by Cheng and Wan [11] and further simplified by Austrin and Khot [6] and Micciancio [36].

While the aforementioned results rule out not only efficient algorithms but also efficient approximation algorithms for MDP, there is another popular technique in coping with NP-hardness of problems which is not yet ruled out by the known results: *parameterization*.

In parameterized problems, part of the input is an integer designated as the parameter of the problem, and the goal is now not to find a polynomial time algorithm but a *fixed parameter tractable (FPT)* algorithm. This is an algorithm whose running time can be upper bounded by some (computable) function of the parameter in addition to some polynomial in the input length. Specifically, for MDP, its parameterized variant[5] k-MDP has k as the parameter and the question is to decide if the code generated by $\mathbf{A}$ has distance at most k in time $T(k) \cdot \text{poly}(mn)$ where T can be any computable function that depends only on k.

The parameterized complexity of k-MDP was first posed as an open problem by Downey et al. [21][6,7] who showed that parameterized variants of several other coding-theoretic problems, including the Nearest Codeword Problem and the Nearest Vector Problem[8] which we will discuss in more details in Section 1.1.1, are W[1]-hard. Thereby, assuming the widely believed W[1] $\neq$ FPT hypothesis, these problems are rendered intractable from the parameterized perspective. Unfortunately, Downey et al. fell short of proving such hardness for k-MDP and left it as an open problem:

[4] Note that MDP can be defined over larger fields as well; we discuss more about this in Section 3.
[5] Throughout Sections 1 and 2, for a computational problem Π, we denote its parameterized variant by k-Π, where k is the parameter of the problem.
[6] k-MDP is formulated differently in [21] where the input is the parity-check matrix instead of the generator matrix. Since we can efficiently compute one given the other, the two formulations are equivalent.
[7] k-MDP is commonly referred to as *k-Even Set* due to its graph theoretic interpretation (see [21]).
[8] The Nearest Vector Problem is also referred to in the literature as the Closest Vector Problem.

▶ **Open Question 1.** *Is k-MDP fixed parameter tractable?*

Although almost two decades have passed, the above question remains unresolved to this day, despite receiving significant attention from the community. In particular, the problem was listed as an open question in the seminal book of Downey and Fellows [19] and has been reiterated numerous times over the years [15, 23, 25, 20, 12, 14, 9, 13, 31]. In fact, in their second book [20], Downey and Fellows even include this problem as one of the six "most infamous" open questions in the area of Parameterized Complexity.

Another question posted in [21] that remains open is the parameterized *Shortest Vector Problem (k-SVP)* in lattices. The input of k-SVP (in the ℓ_p norm) is an integer $k \in \mathbb{N}$ and a matrix $\mathbf{A} \in \mathbb{Z}^{n \times m}$ representing the basis of a lattice, and we want to determine whether the shortest (non-zero) vector in the lattice has length at most k, i.e., whether $\min_{\mathbf{0} \neq \mathbf{x} \in \mathbb{Z}^m} \|\mathbf{Ax}\|_p \leqslant k$. Again, k is the parameter of the problem. Note that, similar to [21], we require the basis of the lattice to be integer-valued, which is sometimes not enforced in the literature (e.g. [45, 3]). This is because, if $\mathbf{A}$ is allowed to have rational entries, then parameterization is meaningless because we can simply scale $\mathbf{A}$ down by a large multiplicative factor.

The (non-parameterized) Shortest Vector Problem (SVP) has been intensively studied, motivated partly due to the fact that both algorithms and hardness results for the problem have numerous applications. Specifically, the celebrated LLL algorithm for SVP [28] can be used to factor rational polynomials, and to solve integer programming (parameterized by the number of unknowns) [29] and many other computational number-theoretic problems (see e.g. [38]). Furthermore, the hardness of (approximating) SVP has been used as the basis of several cryptographic constructions [3, 4, 39, 40]. Since these topics are out of scope of our paper, we refer the interested readers to the following surveys for more details: [41, 37, 38, 42].

On the computational hardness side of the problem, van Emde-Boas [45] was the first to show that SVP is NP-hard for the ℓ_∞ norm, but left open the question of whether SVP on the ℓ_p norm for $1 \leqslant p < \infty$ is NP-hard. It was not until a decade and a half later that Ajtai [2] showed, under a randomized reduction, that SVP for the ℓ_2 norm is also NP-hard; in fact, Ajtai's hardness result holds not only for exact algorithms but also for $(1 + o(1))$-approximation algorithms as well. The $o(1)$ term in the inapproximability ratio was then improved in a subsequent work of Cai and Nerurkar [10]. Finally, Micciancio [33] managed to achieve a factor that is bounded away from one. Specifically, Micciancio [33] showed (again under randomized reductions) that SVP on the ℓ_p norm is NP-hard to approximate within a factor of $\sqrt[p]{2}$ for every $1 \leqslant p < \infty$. Khot [27] later improved the ratio to any constant, and even to $2^{\log^{1/2-\varepsilon}(nm)}$ under a stronger assumption. Haviv and Regev [26] subsequently simplified the gap amplification step of Khot and, in the process, improved the ratio to almost polynomial. We note that both Khot's and Haviv-Regev reductions are also randomized and it is still open to find a deterministic NP-hardness reduction for SVP in the ℓ_p norms for $1 \leqslant p < \infty$ (see [35]); we emphasize here that such a reduction is not known even for the exact (not approximate) version of the problem. For the ℓ_∞ norm, the following stronger result due to Dinur is known [16]: SVP in the ℓ_∞ norm is NP-hard to approximate to within $n^{\Omega(1/\log\log n)}$ factor (under a *deterministic* reduction).

Very recently, fine-grained studies of SVP have been initiated [7, 1]. The authors of [7, 1] showed that SVP for any ℓ_p norm cannot be solved (or even approximated to some constant strictly greater than one) in subexponential time assuming the existence of a certain family of lattices[9] and the (randomized) *Gap Exponential Time Hypothesis (Gap-ETH)* [17, 32], which states that no randomized subexponential time algorithm can distinguish between a satisfiable 3CNF formula and one which is only 0.99-satisfiable.

[9] This assumption is needed only for $p \leqslant 2$. For $p > 2$, their hardness is conditional only on Gap-ETH.

As with MDP, Downey et al. [21] were the first to question the parameterized tractability of k-SVP (for the ℓ_2 norm). Once again, Downey and Fellows included k-SVP as one of the open problems in both of their books [19, 20], albeit, in their second book, k-SVP was in the "tough customers" list instead of the "most infamous" list that k-MDP belonged to. And again, as with Open Question 1, this question remains unresolved to this day:

▶ **Open Question 2.** *Is k-SVP fixed parameter tractable?*

1.1 Our Results

The main result of this paper is a resolution to the previously mentioned Open Question 1 and 2: more specifically, we prove that k-MDP and k-SVP (on ℓ_p norm for any $p > 1$) do not admit any FPT algorithm, assuming the aforementioned (randomized) Gap-ETH. In fact, our result is slightly stronger than stated here in a couple of ways:

- We rule out not only exact FPT algorithms but also FPT approximation algorithms.
- Second, our result works even under the so-called *Parameterized Inapproximability Hypothesis (PIH)* [30], which asserts that no (randomized) FPT algorithm[10] can distinguish between a satisfiable 2CSP instance and one which is only 0.99-satisfiable, where the parameter is the number of variables. It is known that Gap-ETH implies PIH.

With this in mind, we can state our results starting with the parameterized intractability of k-MDP, more concretely (but still informally), as follows:

▶ **Theorem 3.** *Assuming PIH, for any $\gamma \geqslant 1$ and any computable function T, no $T(k) \cdot poly(nm)$-time algorithm, on input $(\mathbf{A}, k) \in \mathbb{F}_2^{n \times m} \times \mathbb{N}$, can distinguish between*

- *the distance of the code generated by $\mathbf{A}$ is at most k, and,*
- *the distance of the code generated by $\mathbf{A}$ is more than $\gamma \cdot k$.*

While our above result rules out FPT approximation algorithms with *any* constant approximation ratio for k-MDP, we can only prove FPT inapproximability with *some* constant ratio for k-SVP in ℓ_p norm for $p > 1$, with the exception of $p = 2$ for which the ratio in our result can be amplified to any constant. These are stated more precisely below.

▶ **Theorem 4.** *For any $p > 1$, there exists a constant $\gamma_p > 1$ such that, assuming PIH, for any computable function T, no $T(k) \cdot poly(nm)$-time algorithm, on input $(\mathbf{A}, k) \in \mathbb{Z}^{n \times m} \times \mathbb{N}$, can distinguish between*

- *the ℓ_p norm of the shortest vector of the lattice generated by $\mathbf{A}$ is $\leqslant k$, and,*
- *the ℓ_p norm of the shortest vector of the lattice generated by $\mathbf{A}$ is $> \gamma_p \cdot k$.*

▶ **Theorem 5.** *Assuming PIH, for any computable function T and constant $\gamma \geqslant 1$, no $T(k) \cdot poly(nm)$-time algorithm, on input $(\mathbf{A}, k) \in \mathbb{Z}^{n \times m} \times \mathbb{N}$, can distinguish between*

- *the ℓ_2 norm of the shortest vector of the lattice generated by $\mathbf{A}$ is $\leqslant k$, and,*
- *the ℓ_2 norm of the shortest vector of the lattice generated by $\mathbf{A}$ is $> \gamma \cdot k$.*

We remark that our results do not yield hardness for SVP in the ℓ_1 norm and this remains an interesting open question. Section 3 contains discussion on this problem. We also note that, for Theorem 4 and onwards, we are only concerned with $p \neq \infty$; this is because, for $p = \infty$, the problem is NP-hard to approximate even when $k = 1$ [45]!

[10] The original formulation from [30] is slightly different in that it states that the problem is W[1]-hard.

1.1.1 Nearest Codeword Problem and Nearest Vector Problem

As we shall see in Section 2, our proof proceeds by first showing FPT hardness of approximation of the non-homogeneous variants of k-MDP and k-SVP called the *k-Nearest Codeword Problem* (k-NCP) and the *k-Nearest Vector Problem* (k-NVP) respectively. For both k-NCP and k-NVP, we are given a target vector $\mathbf{y}$ (in $\mathbb{F}_2^n$ and $\mathbb{Z}^n$, respectively) in addition to $(\mathbf{A}, k)$, and the goal is to find whether there is any $\mathbf{x}$ (in $\mathbb{F}_2^m$ and $\mathbb{Z}^m$, respectively) such that the (Hamming and ℓ_p, respectively) norm of $\mathbf{Ax} - \mathbf{y}$ is at most k.

As an intermediate step of our proof, we show that the k-NCP and k-NVP problems are hard to approximate[11]. This should be compared to [21], in which the authors show that both problems are W[1]-hard. The distinction here is that our result rules out not only exact algorithms but also approximation algorithms, at the expense of the stronger assumption than that of [21]. Indeed, if one could somehow show that k-NCP and k-NVP are W[1]-hard to approximate (to some constant strictly greater than one), then our reduction would imply W[1]-hardness of k-MDP and k-SVP (under randomized reductions). Unfortunately, no such W[1]-hardness of approximation of k-NCP and k-NVP is known yet.

We end this section by remarking that the computational complexity of both (non-parameterized) NCP and NVP are also thoroughly studied (see e.g. [34, 18, 44, 5, 24] in addition to the references for MDP and SVP), and indeed the inapproximability results of these two problems form the basis of hardness of approximation for MDP and SVP.

2 Proof Overview

In the non-parameterized setting, all the aforementioned inapproximability results for both MDP and SVP are shown in two steps: first, one proves the inapproximability of their inhomogeneous counterparts (i.e. NCP and NVP), and then reduces them to MDP and SVP. We follow this general outline. That is, we first show, via relatively simple reductions from PIH, that both k-NCP and k-NVP are hard to approximate. Then, we reduce k-NCP and k-NVP to k-MDP and k-SVP respectively. In this second step, we employ Dumer et al.'s reduction [22] for k-MDP and Khot's reduction [27] for k-SVP. While the latter works almost immediately in the parameterized regime, there are several technical challenges in adapting Dumer et al.'s reduction to our setting. The remainder of this section is devoted to presenting all of our reductions and to highlight such technical challenges and changes in comparison with the non-parameterized settings.

The starting point of all the hardness results in this paper is Gap-ETH. As mentioned earlier, it is well-known that Gap-ETH implies PIH, i.e., PIH is weaker than Gap-ETH. Hence, for the rest of this section, we may start from PIH instead of Gap-ETH.

2.1 Parameterized Intractability of k-MDP from PIH

We start this subsection by describing the Dumer et al.'s (henceforth DMS) reduction [22]. The starting point of the DMS reduction is the NP-hardness of approximating NCP to any constant factor [5]. Let us recall that in NCP we are given a matrix $\mathbf{A} \in \mathbb{F}_2^{n \times m}$, an integer k, and a target vector $\mathbf{y} \in \mathbb{F}_2^n$, and the goal is to determine whether there is any $\mathbf{x} \in \mathbb{F}_2^m$ such that $\|\mathbf{Ax} - \mathbf{y}\|_0$ is at most k. Arora et al. [5] shows that for any constant $\gamma \geqslant 1$, it is NP-hard to distinguish the case when there exists $\mathbf{x}$ such that $\|\mathbf{Ax} - \mathbf{y}\|_0 \leqslant k$ from the case when for all $\mathbf{x}$ we have that $\|\mathbf{Ax} - \mathbf{y}\|_0 > \gamma k$. Dumer et al. introduce the notion of "locally

[11] While our k-MDP result only applies for $\mathbb{F}_2$, our result for k-NCP holds for any finite field $\mathbb{F}_q$ too.

dense codes" to enable a gadget reduction from NCP to MDP. Informally, a locally dense code is a linear code $\mathbf{L}$ with minimum distance d admitting a ball $\mathcal{B}(\mathbf{s}, r)$ centered at $\mathbf{s}$ of radius[12] $r < d$ and containing a large (exponential in the dimension) number of codewords. Moreover, for the gadget reduction to MDP to go through, we require not only the knowledge of the code, but also the center $\mathbf{s}$ and a linear transformation $\mathbf{T}$ used to index the codewords in $\mathcal{B}(\mathbf{s}, r)$, i.e., $\mathbf{T}$ maps $\mathcal{B}(\mathbf{s}, r) \cap \mathbf{L}$ onto a smaller subspace. Given an instance $(\mathbf{A}, \mathbf{y}, k)$ of NCP, and a locally dense code $(\mathbf{L}, \mathbf{T}, \mathbf{s})$ whose parameters (such as dimension and distance) we will fix later, Dumer et al. build the following matrix:

$$\mathbf{B} = \begin{bmatrix} \mathbf{ATL} & -\mathbf{y} \\ \vdots & \vdots \\ \mathbf{ATL} & -\mathbf{y} \\ \mathbf{L} & -\mathbf{s} \\ \vdots & \vdots \\ \mathbf{L} & -\mathbf{s} \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} \mathbf{ATL} & -\mathbf{y} \\ \vdots & \vdots \\ \mathbf{ATL} & -\mathbf{y} \end{matrix}}\right\} a \text{ copies} \\ \left.\vphantom{\begin{matrix} \mathbf{L} & -\mathbf{s} \\ \vdots & \vdots \\ \mathbf{L} & -\mathbf{s} \end{matrix}}\right\} b \text{ copies} \end{matrix}, \tag{1}$$

where a, b are some appropriately chosen positive integers. If there exists $\mathbf{x}$ such that $\|\mathbf{Ax} - \mathbf{y}\|_0 \leqslant k$ then consider $\mathbf{z}'$ such that $\mathbf{TLz}' = \mathbf{x}$ (we choose the parameters of $(\mathbf{L}, \mathbf{T}, \mathbf{s})$, in particular the dimensions of $\mathbf{L}$ and $\mathbf{T}$ such that all these computations are valid). Let $\mathbf{z} = \mathbf{z}' \circ 1$ (where $\circ$ is used to denote the concatenation operation on vectors), and note that $\|\mathbf{Bz}\|_0 = a\|\mathbf{Ax} - \mathbf{y}\|_0 + b\|\mathbf{Lz} - \mathbf{s}\|_0 \leqslant ak + br$. In other words, if $(\mathbf{A}, \mathbf{y}, k)$ is a YES instance of NCP then $(\mathbf{B}, ak + br)$ is a YES instance of MDP. On the other hand if we had that for all $\mathbf{x}$, the norm of $\|\mathbf{Ax} - \mathbf{y}\|_0$ is more than γk for some constant[13] $\gamma > 2$, then it is possible to show that for all $\mathbf{z}$ we have that $\|\mathbf{Bz}\|_0 > \gamma'(ak + br)$ for any $\gamma' < \frac{2\gamma}{2+\gamma}$. The proof is based on a case analysis of the last coordinate of $\mathbf{z}$. If that coordinate is 0, then, since $\mathbf{L}$ is a code of distance d, we have $\|\mathbf{Bz}\|_0 \geqslant bd > \gamma'(ak + br)$; if that coordinate is 1, then the assumption that $(\mathbf{A}, \mathbf{y}, k)$ is a NO instance of NCP implies that $\|\mathbf{Bz}\|_0 > a\gamma k > \gamma'(ak + br)$. Note that this gives an inapproximability for MDP of ratio $\gamma' < 2$; this gap is then further amplified by a simple tensoring procedure.

We note that Dumer et al. were not able to find a deterministic construction of locally dense codes with all of the above described properties. Specifically, they gave an efficient deterministic construction of $\mathbf{L}$, but only gave a randomized algorithm that finds $\mathbf{T}$ and $\mathbf{s}$ w.h.p. Therefore, their hardness result relies on the assumption that NP $\neq$ RP, instead of the more standard NP $\neq$ P assumption. Later, Cheng and Wan [11] and Micciancio [36] provided constructions for such (families of) locally dense codes with an explicit center, and thus showed the constant ratio inapproximability of MDP under the assumption of NP $\neq$ P.

Trying to follow the DMS reduction in order to show the parameterized intractability of k-MDP, we face the following three immediate obstacles. First, there is no inapproximability result known for k-NCP, for any constant factor greater than 1. Note that to use the DMS reduction, we need the parameterized inapproximability of k-NCP, for an approximation factor which is greater than two. Second, the construction of locally dense codes of Dumer et al. only works when the distance is linear in the block length (which is a function of the size of the input). However, we need codes whose distance are bounded above by a function of the parameter of the problem (and does not depend on the input size). This is because

[12] Note that for the ball to contain more than a single codeword, we must have $r \geqslant d/2$.

[13] Note that in the described reduction, we need the inapproximability of NCP to a factor greater than two, even to just reduce to the *exact* version of MDP.

the DMS reduction converts an instance $(\mathbf{A}, \mathbf{y}, k)$ of k-NCP to an instance $(\mathbf{B}, ak + br)$ of $(ak + br)$-MDP, and for this reduction to be an FPT reduction, we need $ak + br$ to be a function only depending on k, i.e., d, the distance of the code $\mathbf{L}$ (which is at most $2r$), must be a function only of k. Third, recall that the DMS reduction needs to identify the vectors in the ball $\mathcal{B}(\mathbf{s}, r) \cap \mathbf{L}$ with all the potential solutions of k-NCP. Notice that the number of vectors in the ball is at most $(nm)^{O(r)}$ but the number of potential solutions of k-NCP is exponential in m (i.e. all $\mathbf{x} \in \mathbb{F}_2^m$). However, this is impossible since $r \leqslant d$ is bounded above by a function of k!

We overcome the first obstacle by proving the constant inapproximability of k-NCP under PIH. Specifically, assuming PIH, we first show the parameterized inapproximability of k-NCP for some constant factor greater than 1, and then boost the gap using a composition operator (self-recursively). Note that in order to follow the DMS reduction, we need the inapproximability of k-NCP for some constant factor greater than 2; in other words, the gap amplification for k-NCP is necessary, even if we are not interested in showing the inapproximability of k-NCP for all constant factors.

We overcome the third obstacle by introducing an intermediate problem in the DMS reduction, which we call the *sparse nearest codeword problem*. The sparse nearest codeword problem is a promise problem which differs from k-NCP in only one way: in the YES case, we want to find $\mathbf{x} \in \mathcal{B}(\mathbf{0}, k)$ (rather than from the entire space $\mathbb{F}_2^m$), such that $\|\mathbf{Ax} - \mathbf{y}\|_0 \leqslant k$. In other words, we only allow sparse $\mathbf{x}$ as a solution. We show that k-NCP can be reduced to the sparse nearest codeword problem.

Finally, we overcome the second obstacle by introducing a variant of locally dense codes, which we call *sparse covering codes*. Roughly speaking, we show that any code which nears the Hamming bound (aka sphere-packing bound) in the high rate regime is a sparse covering code. Then we follow the DMS reduction with the new ingredient of sparse covering codes (replacing locally dense codes) to reduce the sparse nearest codeword problem to k-MDP.

We note that overcoming the second and third obstacles are our main technical contributions. Specifically, our result on sparse covering codes might be of independent interest.

The full reduction goes through several intermediate steps, which we will describe in more detail in the coming paragraphs. Throughout this section, for any gap problem, if we do not specify the gap in the subscript, then it implies that the gap can be any arbitrary constant. For every $\varepsilon \geqslant 0$, we denote by $\text{Gap2CSP}_\varepsilon$ the gap problem where we have to determine if a given 2CSP instance Γ, i.e., a graph $G = (V, E)$ and a set of constraints $\{C_{uv}\}_{(u,v)\in E}$ over an alphabet set Σ, has an assignment to its vertices that satisfies all the constraints or if every assignment violates more than ε fraction of the constraints. Here each C_{uv} is simply the set of all $(\sigma_u, \sigma_v) \in \Sigma \times \Sigma$ that satisfy the constraint. The parameter of the problem is $|V|$. PIH asserts that there exists some constant $\varepsilon > 0$ such that no randomized FPT algorithm can solve $\text{Gap2CSP}_\varepsilon$.

Reducing Gap2CSP$_\varepsilon$ to GapMLD$_\gamma$. We start by showing the parameterized inapproximability of k-NCP for some constant ratio. Instead of working with k-NCP, we work with its equivalent formulation (by converting the generator matrix given as input into a parity-check matrix) which in the literature is referred to as the *maximum likelihood decoding problem*[14]. We define the gap version of this problem (i.e., a promise problem), denoted by GapMLD_γ (for some constant $\gamma \geqslant 1$) as follows: on input $(\mathbf{A}, \mathbf{y}, k)$, distinguish between the YES case

[14] The two formulations are equivalent but we use different names for them to avoid confusion when we use *Sparse Nearest Codeword Problem* later on.

where there exists $\mathbf{x} \in \mathcal{B}(\mathbf{0}, k)$ such that $\mathbf{Ax} = \mathbf{y}$, and the NO case where for all $\mathbf{x} \in \mathcal{B}(\mathbf{0}, \gamma k)$ we have $\mathbf{Ax} \neq \mathbf{y}$. It is good to keep in mind that this is equivalent to asking whether there exist k columns of $\mathbf{A}$ whose sum is equal to $\mathbf{y}$ or whether any $\leqslant \gamma k$ columns of $\mathbf{A}$ do not sum up to $\mathbf{y}$.

Next, we sketch the reduction from an instance $(G = (V, E), \Sigma, \{C_{uv}\}_{(u,v)\in E})$ of Gap2CSP$_\varepsilon$ to an instance $(\mathbf{A}, \mathbf{y}, k)$ of GapMLD$_{1+\varepsilon/3}$. The matrix $\mathbf{A}$ has $|V||\Sigma| + \sum_{(u,v)\in E} |C_{uv}|$ columns and $|V| + |E| + 2|E||\Sigma|$ rows. The first $|V||\Sigma|$ columns of $\mathbf{A}$ are labelled with $(u, \sigma_u) \in V \times \Sigma$, and the remaining columns are labeled by (e, σ_u, σ_v) where $e = (u, v) \in E$ and $(\sigma_u, \sigma_v) \in C_{uv}$.

Before we continue with our description of $\mathbf{A}$, let us note that, in the YES case where there is a satisfying assignment $\phi : V \to \Sigma$, our intended solution for our GapMLD instance is to pick the $(u, \phi(u))$-column for every $u \in V$ and the $((u, v), \phi(u), \phi(v))$-column for every $(u, v) \in E$. Notice that $|V| + |E|$ columns are picked, and indeed we set $k = |V| + |E|$. Moreover, we set the first $|V| + |E|$ coordinates of $\mathbf{y}$ to be one and the rest to be zero.

We also identify the first $|V|$ rows of $\mathbf{A}$ with $u \in V$, the next $|E|$ rows with $e \in E$, and the remaining $2|E||\Sigma|$ rows with $(e, \sigma, b) \in E \times \Sigma \times \{0, 1\}$. Figure 1 provides an illustration of $\mathbf{A}$. The rows of $\mathbf{A}$ will be designed to serve the following purposes: the first $|V|$ rows will ensure that, for each $u \in V$, at least one column of the form $(u, \cdot)$ is picked, the next $|E|$ rows will ensure that, for each $e \in E$, at least one column of the form $(e, \cdot, \cdot)$ is picked, and finally the remaining $2|E||\Sigma|$ rows will "check" that the constraint is indeed satisfied.

Specifically, each u-row for $u \in V$ has only $|\Sigma|$ non-zero entries: those in column (u, σ_u) for all $\sigma_u \in \Sigma$. Since our target vector $\mathbf{y}$ has $\mathbf{y}_u = 1$, we indeed have that at least one column of the form $(u, \cdot)$ must be selected for every $u \in V$. Similarly, each e-row for $e = (u, v) \in E$ has $|C_{uv}|$ non-zero entries: those in column (e, σ_u, σ_v) for all $(\sigma_u, \sigma_v) \in C_{uv}$. Again, these make sure that at least one column of the form $(e, \cdot, \cdot)$ must be picked for every $e \in E$.

Finally, we will define the entries of the last $2|E||\Sigma|$ rows. To do so, let us recall that, in the YES case, we pick the columns $(u, \phi(u))$ for all $u \in V$ and $((u, v), \phi(u), \phi(v))$ for all $(u, v) \in E$. The goal of these remaining rows is to not only accept such a solution but also prevent any solution that picks columns $(u, \sigma_u), (v, \sigma_v)$ and $((u, v), \sigma'_u, \sigma'_v)$ where $\sigma_u \neq \sigma'_u$ or $\sigma_v \neq \sigma'_v$. In other words, these rows serve as a "consistency checker" of the solution. Specifically, the $|\Sigma|$ rows of the form $((u, v), \cdot, 0)$ will force σ_u and σ'_u to be equal whereas the $|\Sigma|$ rows of the form $((u, v), \cdot, 1)$ will force σ_v and σ'_v to be equal. For convenience, we will only define the entries for the $((u, v), \cdot, 0)$-rows; the $((u, v), \cdot, 1)$-rows can be defined similarly. Each $((u, v), \sigma, 0)$-row has only one non-zero entry within the first $|V||\Sigma|$ rows: the one in the (u, σ)-column. For the remaining columns, the entry in the $((u, v), \sigma, 0)$-row and the (e, σ_0, σ_1)-column is non-zero if and only if $e = (u, v)$ and $\sigma_0 = \sigma$.

It should be clear from the definition that our intended solution for the YES case is indeed a valid solution because, for each $((u, v), \phi(u), 0)$-row, the two non-zero entries from the columns $(u, \phi(u))$ and $((u, v), \phi(u), \phi(v))$ cancel each other out. On the other hand, for the NO case, the main observation is that, for each edge $(u, v) \in E$, if only one column of the form $(u, \cdot)$, one of the form $(v, \cdot)$ and one of the form $((u, v), \cdot, \cdot)$ are picked, then the assignment corresponding to the picked columns satisfy the constraint C_{uv}. In particular, it is easy to argue that, if we can pick $(1 + \varepsilon/3)(|V| + |E|)$ columns that sum up to $\mathbf{y}$, then all but ε fraction of all constraints fulfill the previous conditions, meaning that we can find an assignment that satisfies $1 - \varepsilon$ fraction of the constraints. Thus, we have also proved the soundness of the reduction.

Gap Amplification for GapMLD$_\gamma$. We have sketched the proof of the hardness of GapMLD$_\gamma$ for *some* constant $\gamma \geqslant 1$, assuming PIH. The next step is to amplify the gap and arrive at the

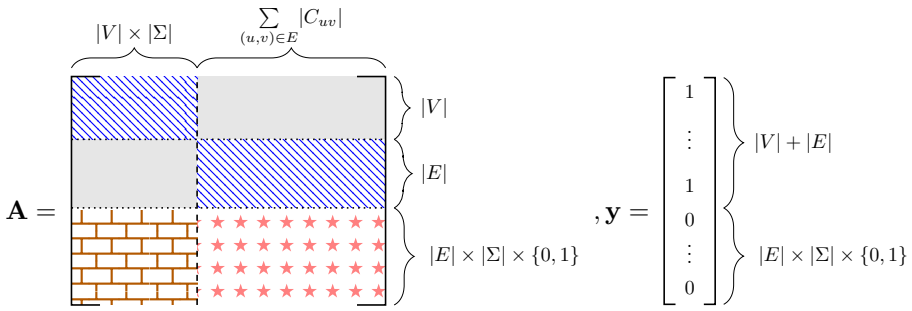


Figure 1 An illustration of $\mathbf{A}$ and $\mathbf{y}$. All entries in shaded areas are zero. Each *row* in the brick pattern area has one non-zero entry in that area, and each *column* in the star pattern area has two non-zero entries in the area. Finally, each column has one non-zero entry in the lines pattern areas.

hardness for GapMLD$_\gamma$ for *every* constant $\gamma \geqslant 1$. To do so, we define an operator $\oplus$ over every pair of instances of GapMLD$_\gamma$ with the following property: if two instances $(\mathbf{A}_1, \mathbf{y}_1, k_1)$ and $(\mathbf{A}_2, \mathbf{y}_2, k_2)$ are both YES instances, then $(\mathbf{A}, \mathbf{y}, k) := (\mathbf{A}_1, \mathbf{y}_1, k_1) \oplus (\mathbf{A}_2, \mathbf{y}_2, k_2)$ is a YES instance for GapMLD$_{\gamma'}$ where $\gamma' \approx \gamma^2$. On the other hand, if both $(\mathbf{A}_1, \mathbf{y}_1, k_1)$ and $(\mathbf{A}_2, \mathbf{y}_2, k_2)$ are NO instances, then $(\mathbf{A}, \mathbf{y}, k)$ is a NO instance for GapMLD$_{\gamma'}$. Hence, we can apply $\oplus$ repeatedly to the GapMLD$_\gamma$ instance from the previous step (with itself) and amplify the gap to be any arbitrarily large constant. The definition of $\oplus$, while simple, is slightly tedious to formalize and we defer it to the full version of this paper.

Reducing GapMLD to GapSNC. Now we introduce the *sparse nearest codeword problem* that we had briefly talked about. We define the gap version of this problem, denoted by GapSNC$_\gamma$ (for some constant $\gamma \geqslant 1$) as follows: on input $(\mathbf{A}', \mathbf{y}', k)$, distinguish between the YES case where there exists $\mathbf{x} \in \mathcal{B}(\mathbf{0}, k)$ such that $\|\mathbf{A}'\mathbf{x} - \mathbf{y}'\|_0 \leqslant k$, and the NO case where for all $\mathbf{x}$ (in the entire space), we have $\|\mathbf{A}'\mathbf{x} - \mathbf{y}'\|_0 > \gamma k$. We highlight that the difference between k-NCP and GapSNC$_\gamma$ is that, in the YES case of the latter, we are promised that $\mathbf{x} \in \mathcal{B}(\mathbf{0}, k)$. We sketch below the reduction from an instance $(\mathbf{A}, \mathbf{y}, k)$ of GapMLD$_\gamma$ to an instance $(\mathbf{A}', \mathbf{y}', k)$ of GapSNC$_\gamma$. Given $\mathbf{A}, \mathbf{y}$, let

$$\mathbf{A}' = \begin{bmatrix} \mathbf{A} \\ \vdots \\ \mathbf{A} \\ \mathrm{Id} \end{bmatrix} \Big\} \gamma k + 1 \text{ copies}, \qquad \mathbf{y}' = \begin{bmatrix} \mathbf{y} \\ \vdots \\ \mathbf{y} \\ \mathbf{0} \end{bmatrix} \Big\} \gamma k + 1 \text{ copies}.$$

Notice that for any $\mathbf{x}$ (in the entire space), we have $\|\mathbf{A}'\mathbf{x} - \mathbf{y}'\|_0 = (\gamma k + 1)\|\mathbf{A}\mathbf{x} - \mathbf{y}\|_0 + \|\mathbf{x}\|_0$, and thus both the completeness and soundness of the reduction easily follow.

Sparse Covering Codes. Before reducing GapSNC to GapMDP$_{1.99}$ we need to introduce in more detail the notion of *sparse covering codes* that we previously mentioned.

A sparse covering code (SCC) is a linear code $\mathbf{L}$ of block length h with minimum distance d admitting a ball $\mathcal{B}(\mathbf{s}, r)$ centered at $\mathbf{s}$ of radius $r < d$ and containing a large (i.e., about h^k, where $k = \Omega(d)$) number of codewords. Moreover, for our reduction to go through, we require not only $\mathbf{L}$ and $\mathbf{s}$, but also a linear transformation $\mathbf{T}$ used to index the codewords in $\mathcal{B}(\mathbf{s}, r)$, i.e., $\mathbf{T}(\mathcal{B}(\mathbf{s}, r) \cap \mathbf{L})$ needs to contains the ball of radius k centered at $\mathbf{0}$. Similar to

how Dumer et al. only managed to show the probabilistic existence of the center, we too cannot find an explicit $\mathbf{s}$ for the SCCs that we construct, but instead provide an efficiently samplable distribution such that, for any $\mathbf{x} \in \mathcal{B}(\mathbf{0}, k)$, the probability (over $\mathbf{s}$ sampled from the distribution) that $\mathbf{x} \in \mathbf{T}(\mathcal{B}(\mathbf{s}, r) \cap \mathbf{L})$ is non-negligible. This is what makes our reduction from GapSNC to $\text{GapMDP}_{1.99}$ randomized. We will not elaborate more on this issue here, but focus on the (probabilistic) construction of such codes. For convenience, we will assume throughout this overview that k is much smaller than d, i.e., $k = 0.001d$.

Recall that the Hamming (aka sphere-packing) bound states that a binary code of block length h and distance d can have at most $2^h/|\mathcal{B}(\mathbf{0}, \lceil \frac{d-1}{2} \rceil)|$ codewords, because the balls of radius $\lceil \frac{d-1}{2} \rceil$ centered at the codewords do not intersect. Our main theorem regarding the existence of SCC is that any code that is "near" the Hamming bound is a sparse covering code with $r = \lceil \frac{d-1}{2} \rceil + k \approx 0.501d$. Here "near" means that the number of codewords must be at least $2^h/|\mathcal{B}(\mathbf{0}, \lceil \frac{d-1}{2} \rceil)|$ divided by $f(d) \cdot \text{poly}(h)$ for some function f that depends only on d. (Equivalently, this means that the message length must be at least $h - (d/2 + O(1)) \log h$.) The BCH code over binary alphabet is an example of a code satisfying such a condition.

While we do not sketch the proof of the theorem here, we note that the idea is to set $\mathbf{T}$ and the distribution over $\mathbf{s}$ in such a way that the probability that $\mathbf{x}$ lies in $\mathbf{T}(\mathcal{B}(\mathbf{s}, r) \cap \mathbf{L})$ is at least the probability that a random point in $\mathbb{F}_2^h$ is within distance $r - k = \lceil \frac{d-1}{2} \rceil$ of some codeword. The latter is non-negligible since $\mathbf{L}$ nears the Hamming bound.

Finally, we remark that our proof here is completely different from the DMS proof of existence of locally dense codes. Specifically, DMS uses a group-theoretic argument to show that, when a code exceeds the Gilbert–Varshamov bound, there must be a center $\mathbf{s}$ such that $\mathcal{B}(\mathbf{s}, r)$ contains many codewords. Then, they pick a random linear map $\mathbf{T}$ and show that w.h.p. $\mathbf{T}(\mathcal{B}(\mathbf{s}, r) \cap \mathbf{L})$ is the entire space. Note that this second step does not use any structure of $\mathcal{B}(\mathbf{s}, r) \cap \mathbf{L}$; their argument is simply that, for any sufficiently large subset Y, a random linear map $\mathbf{T}$ maps Y to an entire space w.h.p. However, such an argument fails for us, due to the fact that, in SCC, we want to cover a ball $\mathcal{B}(\mathbf{0}, k)$ rather than the whole space, and it is not hard to see that there are very large subsets Y such that no linear map $\mathbf{T}$ satisfies $\mathbf{T}(Y) \supseteq \mathcal{B}(\mathbf{0}, k)$. A simple example of this is when Y is a subspace of $\mathbb{F}_2^h$; in this case, even when Y is as large as $\exp(\text{poly}(h))$, no desired linear map $\mathbf{T}$ exists.

Reducing GapSNC_γ to $\text{GapMDP}_{1.99}$. Next, we prove the hardness of $\text{GapMDP}_{\gamma'}$ for all constant $\gamma' \in [1, 2)$, assuming PIH, using a gadget constructed from sparse covering codes.

Given an instance $(\mathbf{A}, \mathbf{y}, k)$ of GapSNC_γ for some $\gamma > 2$ and a SCC $(\mathbf{L}, \mathbf{T}, \mathbf{s})$ we build an instance $(\mathbf{B}, ak + br)$ of $\text{GapMDP}_{\gamma'}$ where $\gamma' < \frac{2\gamma}{2+\gamma}$, by following the DMS reduction (which was previously described; see (1)). If $\|\mathbf{Ax} - \mathbf{y}\|_0 \leqslant k$ for some $\mathbf{x} \in \mathcal{B}(\mathbf{0}, k)$, then consider $\mathbf{z}'$ such that $\mathbf{TLz}' = \mathbf{x}$; the existence of such a $\mathbf{z}'$ is guaranteed by the definition of SCC. Consider $\mathbf{z} = \mathbf{z}' \circ 1$, and note that $\|\mathbf{Bz}\|_0 = a\|\mathbf{Ax} - \mathbf{y}\|_0 + b\|\mathbf{Lz} - \mathbf{s}\|_0 \leqslant ak + br$. In other words, as in the DMS reduction, if $(\mathbf{A}, \mathbf{y}, k)$ is a YES instance of NCP, then $(\mathbf{B}, ak + br)$ is a YES instance of MDP. On the other hand, similar to the DMS reduction, if we had that $\|\mathbf{Ax} - \mathbf{y}\|_0 > \gamma k$ for all $\mathbf{x}$, then $\|\mathbf{Bz}\|_0 > \gamma'(ak + br)$ for all $\mathbf{z}$. The parameterized intractability of $\text{GapMDP}_{1.99}$ is obtained by setting $\gamma = 400$ in the above reduction.

Gap Amplification for $\text{GapMDP}_{1.99}$. It is well known that the distance of the tensor product of two linear codes is the product of the distances of the individual codes. We can use this proposition to reduce GapMDP_γ to GapMDP_{γ^2} for any $\gamma \geqslant 1$. In particular, we can obtain, for any constant γ, the intractability of GapMDP_γ starting from $\text{GapMDP}_{1.99}$ by just recursively tensoring the input code $\lceil \log_{1.99} \gamma \rceil$ times.

2.2 Parameterized Intractability of k-SVP from PIH

We begin this subsection by briefly describing Khot's reduction. The starting point of Khot's reduction is the NP-hardness of approximating NVP in every ℓ_p norm to any constant factor [5]. Let us recall that in NVP in the ℓ_p norm, we are given a matrix $\mathbf{A} \in \mathbb{Z}^{n\times m}$, an integer k, and a target vector $\mathbf{y} \in \mathbb{Z}^n$, and the goal is to determine whether there is any $\mathbf{x} \in \mathbb{Z}^m$ such that$\|\mathbf{Ax} - \mathbf{y}\|_p^p$ is at most k. The result of Arora et al. [5] states that for any constant $\gamma \geqslant 1$, it is NP-hard to distinguish the case when there exists $\mathbf{x}$ such that $\|\mathbf{Ax} - \mathbf{y}\|_p^p \leqslant k$ from the case when for all (integral) $\mathbf{x}$ we have that $\|\mathbf{Ax} - \mathbf{y}\|_p^p > \gamma k$. Khot's reduction proceeds in four steps. First, he constructs a gadget lattice called the "BCH Lattice" using BCH Codes. Next, he reduces NVP in the ℓ_p norm (where $p \in (1, \infty)$) to an instance of SVP on an intermediate lattice by using the BCH Lattice. This intermediate lattice has the following property. For any YES instance of NVP the intermediate lattice contains multiple copies of the witness of the YES instance; For any NO instance of NVP there are also many "annoying vectors" (but far less than the total number of YES instance witnesses) which look like witnesses of a YES instance. However, since the annoying vectors are outnumbered, Khot reduces this intermediate lattice to a proper SVP instance, by randomly picking a sub-lattice via a random homogeneous linear constraint on the coordinates of the lattice vectors (this annihilates all the annoying vectors while retaining at least one witness for the YES instance). Thus he obtains some constant factor hardness for SVP. Finally, the gap is amplified via "Augmented Tensor Product". It is important to note that Khot's reduction is randomized, and thus his result of inapproximability of SVP is based on NP $\neq$ RP.

Trying to follow Khot's reduction, in order to show the parameterized intractability of k-SVP, we face only one obstacle: there is no known parameterized inapproximability of k-NVP for any constant factor greater than 1. Let us denote by $\text{GapNVP}_{p,\eta}$ for any constant $\eta \geqslant 1$ the gap version of k-NVP in the ℓ_p norm. Recall that in $\text{GapNVP}_{p,\eta}$ we are given a matrix $\mathbf{A} \in \mathbb{Z}^{n\times m}$, a target vector $\mathbf{y} \in \mathbb{Z}^n$, and a parameter k, and we would like to distinguish the case when there exists $\mathbf{x} \in \mathbb{Z}^m$ such that $\|\mathbf{Ax} - \mathbf{y}\|_p^p \leqslant k$ from the case when for all $\mathbf{x} \in \mathbb{Z}^m$ we have that $\|\mathbf{Ax} - \mathbf{y}\|_p^p > \eta k$. As it turns out, our reduction from $\text{Gap2CSP}_\varepsilon$ to GapSNC (with arbitrary constant gap), having GapMLD_γ and GapMLD as intermediate steps, can be translated to show the constant inapproximability of GapNVP_p (under PIH) in a straightforward manner. We will not elaborate on this part of the proof any further here and defer the detailed proof to the full version of this paper.

Once we have established the constant parameterized inapproximability of GapNVP_p, we follow Khot's reduction, and everything goes through as it is to establish the inapproximability for some factor of the gap version of k-SVP in the ℓ_p norm (where $p \in (1, \infty)$). We denote by $\text{GapSVP}_{p,\gamma}$ for some constant $\gamma(p) \geqslant 1$ the gap version of k-SVP (in the ℓ_p norm) where we are given a matrix $\mathbf{B} \in \mathbb{Z}^{n\times m}$ and a parameter $k \in \mathbb{N}$, and we would like to distinguish the case when there exists a non-zero $\mathbf{x} \in \mathbb{Z}^m$ such that $\|\mathbf{Bx}\|_p^p \leqslant k$ from the case when for all $\mathbf{x} \in \mathbb{Z}^m \setminus \{\mathbf{0}\}$ we have that $\|\mathbf{Bx}\|_p^p > \gamma k$. Let $\gamma^* := \frac{2^p}{2^{p-1}+1}$. Following Khot's reduction, we obtain the inapproximability of $\text{GapSVP}_{p,\gamma^*}$ (under PIH). To obtain inapproximability of GapSVP_2 for all constant ratios, we use the tensor product of lattices; the argument needed here is slightly more subtle than the similar step in MDP because, unlike distances of codes, the ℓ_2 norm of the shortest vector of the tensor product of two lattices is not necessarily equal to the product of the ℓ_2 norm of the shortest vector of each lattice. Fortunately, Khot's construction is tailored so that the resulting lattice is "well-behaved" under tensoring [27, 26], and gap amplification is indeed possible for such instances.

We remark here that, for the (non-parameterized) inapproximability of SVP, the techniques of [27, 26] allow one to successfully amplify gaps for ℓ_p norm where $p \neq 2$ as well. Unfortunately, this does not work in our settings, as it requires the distance k to be dependent on nm which is not possible for us since k is the parameter of the problem.

3 Discussion and Open Questions

While our results give an evidence of intractability of k-MDP and k-SVP, there are still many questions that remain open. First and foremost, it is still open whether the hardness of both problems can be based on more standard assumptions, such as ETH or W[1] $\neq$ FPT. On this front, we would like to note that the only reason we need PIH is to arrive at the inapproximability of the non-homogeneous variants of the problems, which is needed for us even if we want to only rule out exact FPT algorithms for k-MDP and k-SVP. Hence, if one could prove the hardness of approximation for these problems under weaker assumptions, then the inapproximability of k-MDP and k-SVP would still follow.

Another obvious question is whether k-SVP in the ℓ_1 norm is in FPT. Khot's reduction unfortunately does not work for ℓ_1; indeed, in [26], the hardness of approximating SVP in the ℓ_1 norm is shown by embedding SVP instances in ℓ_2 to instances in ℓ_1 using an earlier result of Regev and Rosen [43]. This embedding inherently does not work in the FPT regime either, as it produces non-integral lattices. Similar issue applies to an earlier hardness result for SVP on ℓ_1 of [33], whose reduction produces irrational bases.

An additional question regarding k-SVP is whether we can prove inapproximability for *every* constant factor for $p \neq 2$. As described earlier, the gap amplification techniques of [27, 26] require the distance k to be dependent on the input size nm, and hence are not applicable for us. To the best of our knowledge, it is unknown whether this dependency is necessary. If they are indeed required, it would be interesting to come up with different gap amplification techniques that also work for our settings.

Furthermore, k-MDP can be defined for linear codes in $\mathbb{F}_p$ for any larger field of size $p > 2$ as well. It turns out that our result does not rule out FPT algorithms for k-MDP over $\mathbb{F}_p$ with $p > 2$. The issue here is that, in our proof of existence of Sparse Covering Codes, we need the co-dimension of the code to be small compared to its distance. In particular, the co-dimension $h - m$ has to be at most $(d/2 + O(1)) \log_p h$ where d is the distance. While the BCH code over binary alphabet satisfies this property, we are not aware of any linear codes that satisfy this for larger fields. It is an intriguing open question to determine whether such codes exist, or whether the reduction can be made to work without existence of such codes.

Since the current reductions for both k-MDP and k-SVP are randomized, it is still an intriguing open question whether we can find deterministic reductions from PIH to these problems. As stated in the introduction, even in the non-parameterized setting, NP-hardness of SVP through deterministic reductions is not known. On the other hand, MDP is known to be NP-hard even to approximate under deterministic reductions; in fact, even the DMS reduction [22] that we employ can be derandomized, as long as one has a deterministic construction for Locally Dense Codes [11, 36]. In our settings, if one can deterministically construct Sparse Covering Codes, we would also get a deterministic reduction for k-MDP.

Finally, another interesting research direction is to prove more concrete running time lower bounds for k-MDP and k-SVP. For instance, k-MDP can be trivially solved (exactly) in $N^{O(k)}$ time, where $N = nm$ is the input size. On the other hand, while not stated explicitly above, our proof implies that k-MDP cannot be solved (or even approximated) in time $N^{o(k^c)}$ for some small constant $c > 0$, assuming Gap-ETH. Would it be possible to improve this running time lower bound to the tight $N^{o(k)}$? Similar questions also apply to k-SVP.

References

1. Divesh Aggarwal and Noah Stephens-Davidowitz. (gap/s)eth hardness of SVP. *CoRR*, abs/1712.00942, 2017. arXiv:1712.00942.
2. Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 99–108, 1996. doi:10.1145/237814.237838.
3. Miklós Ajtai. The shortest vector problem in ℓ_2 is NP-hard for randomized reductions (extended abstract). In *STOC*, pages 10–19, 1998. doi:10.1145/276698.276705.
4. Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *STOC*, pages 284–293, 1997. doi:10.1145/258533.258604.
5. Sanjeev Arora, László Babai, Jacques Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. *J. Comput. Syst. Sci.*, 54(2):317–331, 1997. doi:10.1006/jcss.1997.1472.
6. Per Austrin and Subhash Khot. A simple deterministic reduction for the gap minimum distance of code problem. *IEEE Trans. Information Theory*, 60(10):6636–6645, 2014. doi:10.1109/TIT.2014.2340869.
7. Huck Bennett, Alexander Golovnev, and Noah Stephens-Davidowitz. On the quantitative hardness of CVP. In *FOCS*, pages 13–24, 2017. doi:10.1109/FOCS.2017.11.
8. Elwyn R. Berlekamp, Robert J. McEliece, and Henk C. A. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Trans. Information Theory*, 24(3):384–386, 1978. doi:10.1109/TIT.1978.1055873.
9. Arnab Bhattacharyya, Ameet Gadekar, Suprovat Ghoshal, and Rishi Saket. On the hardness of learning sparse parities. In *ESA*, pages 11:1–11:17, 2016. doi:10.4230/LIPIcs.ESA.2016.11.
10. Jin-yi Cai and Ajay Nerurkar. Approximating the SVP to within a factor $(1 + 1/\dim^{\xi})$ is NP-hard under randomized reductions. *J. Comput. Syst. Sci.*, 59(2):221–239, 1999. doi:10.1006/jcss.1999.1649.
11. Qi Cheng and Daqing Wan. A deterministic reduction for the gap minimum distance problem. *IEEE Trans. Information Theory*, 58(11):6935–6941, 2012. doi:10.1109/TIT.2012.2209198.
12. Marek Cygan, Fedor Fomin, Bart MP Jansen, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, and Saket Saurabh. Open problems for fpt school 2014, 2014.
13. Marek Cygan, Fedor V. Fomin, Danny Hermelin, and Magnus Wahlström. Randomization in parameterized complexity (dagstuhl seminar 17041). *Dagstuhl Reports*, 7(1):103–128, 2017. doi:10.4230/DagRep.7.1.103.
14. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
15. Erik D. Demaine, Gregory Gutin, Dániel Marx, and Ulrike Stege. 07281 open problems – structure theory and FPT algorithmcs for graphs, digraphs and hypergraphs. In Erik D. Demaine, Gregory Z. Gutin, Dániel Marx, and Ulrike Stege, editors, *Structure Theory and FPT Algorithmics for Graphs, Digraphs and Hypergraphs, 08.07. - 13.07.2007*, volume 07281 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007. URL: http://drops.dagstuhl.de/opus/volltexte/2007/1254.
16. Irit Dinur. Approximating SVP_∞ to within almost-polynomial factors is NP-hard. *Theor. Comput. Sci.*, 285(1):55–71, 2002. doi:10.1016/S0304-3975(01)00290-0.
17. Irit Dinur. Mildly exponential reduction from gap 3SAT to polynomial-gap label-cover. *ECCC*, 23:128, 2016. URL: http://eccc.hpi-web.de/report/2016/128.

18 Irit Dinur, Guy Kindler, Ran Raz, and Shmuel Safra. Approximating CVP to within almost-polynomial factors is NP-hard. *Combinatorica*, 23(2):205–243, 2003. doi:10.1007/s00493-003-0019-y.

19 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

20 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

21 Rodney G. Downey, Michael R. Fellows, Alexander Vardy, and Geoff Whittle. The parametrized complexity of some fundamental problems in coding theory. *SIAM J. Comput.*, 29(2):545–570, 1999. doi:10.1137/S0097539797323571.

22 Ilya Dumer, Daniele Micciancio, and Madhu Sudan. Hardness of approximating the minimum distance of a linear code. *IEEE Trans. Information Theory*, 49(1):22–37, 2003. doi:10.1109/TIT.2002.806118.

23 Michael R. Fellows, Jiong Guo, Dániel Marx, and Saket Saurabh. Data reduction and problem kernels (dagstuhl seminar 12241). *Dagstuhl Reports*, 2(6):26–50, 2012. doi:10.4230/DagRep.2.6.26.

24 Oded Goldreich, Daniele Micciancio, Shmuel Safra, and Jean-Pierre Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Inf. Process. Lett.*, 71(2):55–61, 1999. doi:10.1016/S0020-0190(99)00083-6.

25 Petr A. Golovach, Jan Kratochvíl, and Ondrej Suchý. Parameterized complexity of generalized domination problems. *Discrete Applied Mathematics*, 160(6):780–792, 2012. doi:10.1016/j.dam.2010.11.012.

26 Ishay Haviv and Oded Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In *STOC*, pages 469–477, 2007. doi:10.1145/1250790.1250859.

27 Subhash Khot. Hardness of approximating the shortest vector problem in lattices. *J. ACM*, 52(5):789–808, 2005.

28 Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

29 Hendrik Willem Lenstra. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/moor.8.4.538.

30 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. *CoRR*, abs/1704.04249, 2017. arXiv:1704.04249.

31 Ruhollah Majdoddin. Parameterized complexity of CSP for infinite constraint languages. *CoRR*, abs/1706.10153, 2017. arXiv:1706.10153.

32 Pasin Manurangsi and Prasad Raghavendra. A birthday repetition theorem and complexity of approximating dense CSPs. *CoRR*, abs/1607.02986, 2016. URL: http://arxiv.org/abs/1607.02986, arXiv:1607.02986.

33 Daniele Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. *SIAM J. Comput.*, 30(6):2008–2035, 2000. doi:10.1137/S0097539700373039.

34 Daniele Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Trans. Information Theory*, 47(3):1212–1215, 2001. doi:10.1109/18.915688.

35 Daniele Micciancio. Inapproximability of the shortest vector problem: Toward a deterministic reduction. *Theory of Computing*, 8(1):487–512, 2012. doi:10.4086/toc.2012.v008a022.

36 Daniele Micciancio. Locally dense codes. In *CCC*, pages 90–97. IEEE Computer Society, 2014. doi:10.1109/CCC.2014.17.

37 Daniele Micciancio and Oded Regev. Lattice-based cryptography. In *Post-quantum cryptography*, pages 147–191. Springer, 2009.

38 Phong Q. Nguyen and Brigitte Vallée, editors. *The LLL Algorithm - Survey and Applications*. Information Security and Cryptography. Springer, 2010. doi:10.1007/978-3-642-02295-1.

39 Oded Regev. New lattice based cryptographic constructions. In Lawrence L. Larmore and Michel X. Goemans, editors, *STOC*, pages 407–416. ACM, 2003. doi:10.1145/780542.780603.

40 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005. doi:10.1145/1060590.1060603.

41 Oded Regev. Lattice-based cryptography. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 131–141. Springer, 2006. doi:10.1007/11818175_8.

42 Oded Regev. The learning with errors problem (invited survey). In *CCC*, pages 191–204, 2010. doi:10.1109/CCC.2010.26.

43 Oded Regev and Ricky Rosen. Lattice problems and norm embeddings. In *STOC*, pages 447–456, 2006. doi:10.1145/1132516.1132581.

44 Jacques Stern. Approximating the number of error locations within a constant ratio is NP-complete. In Gérard D. Cohen, Teo Mora, and Oscar Moreno, editors, *AAECC*, volume 673 of *Lecture Notes in Computer Science*, pages 325–331. Springer, 1993. doi:10.1007/3-540-56686-4_54.

45 Peter van Emde-Boas. *Another NP-complete partition problem and the complexity of computing short vectors in a lattice*. Report. Department of Mathematics. University of Amsterdam. Department, Univ., 1981.

46 Alexander Vardy. Algorithmic complexity in coding theory and the minimum distance problem. In *STOC*, pages 92–109, 1997. doi:10.1145/258533.258559.

47 Alexander Vardy. The intractability of computing the minimum distance of a code. *IEEE Trans. Information Theory*, 43(6):1757–1766, 1997. doi:10.1109/18.641542.

Rollercoasters and Caterpillars

Therese Biedl[1]
School of Computer Science, University of Waterloo, Canada
biedl@uwaterloo.ca

Ahmad Biniaz[2]
School of Computer Science, University of Waterloo, Canada
ahmad.biniaz@gmail.com

Robert Cummings
School of Computer Science, University of Waterloo, Canada
rcummings000@gmail.com

Anna Lubiw[3]
School of Computer Science, University of Waterloo, Canada
alubiw@uwaterloo.ca

Florin Manea[4]
Department of Computer Science, Kiel University, D-24098 Kiel, Germany
flm@zs.uni-kiel.de

Dirk Nowotka[5]
Department of Computer Science, Kiel University, D-24098 Kiel, Germany
dn@zs.uni-kiel.de

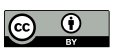
Jeffrey Shallit[6]
School of Computer Science, University of Waterloo, Canada
shallit@cs.uwaterloo.ca

Abstract

A *rollercoaster* is a sequence of real numbers for which every maximal contiguous subsequence – increasing or decreasing – has length at least three. By translating this sequence to a set of points in the plane, a rollercoaster can be defined as an x-monotone polygonal path for which every maximal sub-path, with positive- or negative-slope edges, has at least three vertices. Given a sequence of distinct real numbers, the rollercoaster problem asks for a maximum-length (not necessarily contiguous) subsequence that is a rollercoaster. It was conjectured that every sequence of n distinct real numbers contains a rollercoaster of length at least $\lceil n/2 \rceil$ for $n > 7$, while the best known lower bound is $\Omega(n/\log n)$. In this paper we prove this conjecture. Our proof is constructive and implies a linear-time algorithm for computing a rollercoaster of this length. Extending the $O(n \log n)$-time algorithm for computing a longest increasing subsequence, we show how to compute a maximum-length rollercoaster within the same time bound. A maximum-length rollercoaster in a permutation of $\{1, \ldots, n\}$ can be computed in $O(n \log \log n)$ time.

The search for rollercoasters was motivated by orthogeodesic point-set embedding of caterpillars. A *caterpillar* is a tree such that deleting the leaves gives a path, called the *spine*. A *top-view caterpillar* is one of maximum degree 4 such that the two leaves adjacent to each vertex lie on opposite sides of the spine. As an application of our result on rollercoasters, we are able to

[1] Supported by NSERC.
[2] Supported by NSERC Postdoctoral Fellowship.
[3] Supported by NSERC.
[4] Supported by DFG.
[5] Supported by DFG.
[6] Supported by NSERC Grant # 105829/2013.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 18; pp. 18:1–18:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

find a planar drawing of every n-vertex top-view caterpillar on every set of $\frac{25}{3}(n+4)$ points in the plane, such that each edge is an orthogonal path with one bend. This improves the previous best known upper bound on the number of required points, which is $O(n \log n)$. We also show that such a drawing can be obtained in linear time, when the points are given in sorted order.

2012 ACM Subject Classification Theory of computation → Algorithm design techniques

Keywords and phrases sequences, alternating runs, patterns in permutations, caterpillars

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.18

Related Version A full version of the paper is available at [3], `https://arxiv.org/abs/1801.08565`.

1 Introduction

A *run* in a sequence of real numbers is a maximal contiguous subsequence that is increasing or decreasing. A *rollercoaster* is a sequence of real numbers such that every run has length at least three.[7] For example the sequence $(8, 5, 1, 3, 4, 7, 6, 2)$ is a rollercoaster with runs $(8, 5, 1)$, $(1, 3, 4, 7)$, $(7, 6, 2)$, which have lengths 3, 4, 3, respectively. The sequence $(8, 5, 1, 7, 6, 2, 3, 4)$ is not a rollercoaster because its run $(1, 7)$ has length 2. Given a sequence $S = (s_1, s_2, \ldots, s_n)$ of n distinct real numbers, the rollercoaster problem is to find a maximum-size set of indices $i_1 < i_2 < \cdots < i_k$ such that $(s_{i_1}, s_{i_2}, \ldots, s_{i_k})$ is a rollercoaster. In other words, this problem asks for a longest rollercoaster in S, i.e., a longest subsequence of S that is a rollercoaster.

One can interpret S as a set P of points in the plane by translating each number $s_i \in S$ to a point $p_i = (i, s_i)$. With this translation, a rollercoaster in S translates to a "rollercoaster" in P, which is a polygonal path whose vertices are points of P and such that every maximal sub-path, with positive- or negative-slope edges, has at least three points. See Figure 1(a). Conversely, for any point set in the plane, the y-coordinates of the points, ordered by their x-coordinates, form a sequence of numbers. Therefore, any rollercoaster in P translates to a rollercoaster of the same length in S.

The best known lower bound on the length of a longest rollercoaster is $\Omega(n/\log n)$ due to Biedl et al. [4], who posed the following conjecture; see Appendix B in the full version of [4].

▶ **Conjecture 1.** *Every sequence of $n > 7$ distinct real numbers contains a rollercoaster of length at least $\lceil n/2 \rceil$.*

Conjecture 1 can be viewed as a statement about patterns in permutations, a topic with a long history, and the subject of much current research. For example, the Eulerian polynomials, introduced by Euler in 1749, are the generating function for the number of descents in permutations. For surveys of recent work, see, for example, Linton et al. [13] and Kitaev [12]. Specifically, Conjecture 1 is related to the following seminal result of Erdős and Szekeres [7] in the sense that they prove the existence of an increasing or a decreasing subsequence of length at least $a + 1$ for $n = a^2 + 1$, which is essentially a rollercoaster with one run.

[7] The term "rollercoaster permutation" has also been used to refer to a permutation that, together with all its subsequences, has maximum number of changes from increasing to decreasing and vice versa; see e.g. [1].

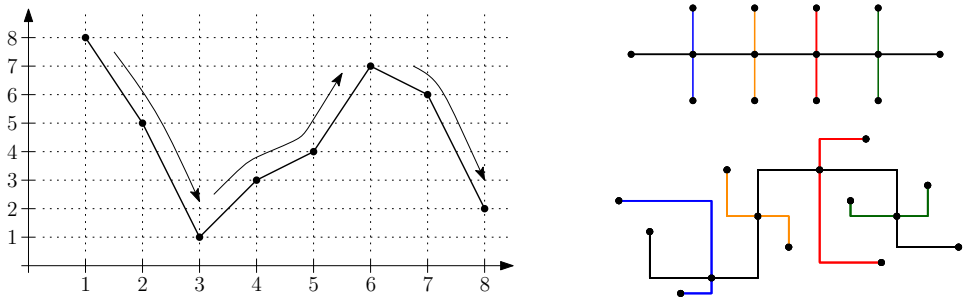


Figure 1 (a) Translating the sequence $(8, 5, 1, 3, 4, 7, 6, 2)$ to a set of points. (b) A planar L-shaped drawing of a top-view caterpillar.

▶ **Theorem 2** (Erdős and Szekeres, 1935). *Every sequence of $ab + 1$ distinct real numbers contains an increasing subsequence of length at least $a + 1$ or a decreasing subsequence of length at least $b + 1$.*

Hammersley [11] gave an elegant proof of the Erdős–Szekeres theorem that is short, simple, and based on the pigeonhole principle. The Erdős–Szekeres theorem also follows from the well-known decomposition of Dilworth (see [17]). The following is a restatement of Dilworth's decomposition for sequences of numbers.

▶ **Theorem 3** (Dilworth, 1950). *Any finite sequence S of distinct real numbers can be partitioned into k ascending sequences where k is the maximum length of a descending sequence in S.*

Besides its inherent interest, the study of rollercoasters is motivated by point-set embedding of caterpillars [4]. A *caterpillar* is a tree such that deleting the leaves gives a path, called the *spine*. An *ordered caterpillar* is a caterpillar in which the cyclic order of the edges incident to each vertex is specified. A *top-view caterpillar* is an ordered caterpillar where all vertices have degree 4 or 1 such that the two leaves adjacent to each spine vertex lie on opposite sides of the spine; see Figure 1(b). Planar orthogonal drawings of trees on a fixed set of points in the plane have been explored recently, see e.g., [4, 10, 15]; in these drawings every edge is drawn as an orthogonal path between two points, and the edges are non-intersecting. A *planar L-shaped drawing* is a simple type of planar orthogonal drawing in which every edge is an orthogonal path of exactly two segments. Such a path is called an *L-shaped edge*. For example see the top-view caterpillar in Figure 1(b) together with a planar L-shaped drawing on a given point set. Biedl et al. [4] proved that every top-view caterpillar on n vertices has a planar L-shaped drawing on every set of $O(n \log n)$ points in the plane that is in *general orthogonal position*, meaning that no two points have the same x- or y-coordinate.

1.1 Our Contributions

In Section 2 we study rollercoasters and prove Conjecture 1. In fact we prove something stronger: every sequence of n distinct numbers contains two rollercoasters of total length n. Our proof is constructive and yields a linear-time algorithm for computing such rollercoasters. We also extend our result to rollercoasters whose runs are of length at least k, for $k > 3$. Then we present an $O(n \log n)$-time algorithm for computing a longest rollercoaster, extending

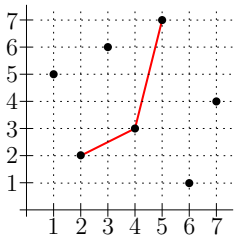
the classical algorithm for computing a longest increasing subsequence. This algorithm can be implemented in $O(n \log \log n)$ time if each number in the input sequence is an integer that fits in a constant number of memory words. Then we give an estimate on the number of permutations of $\{1, \ldots, n\}$ that are rollercoasters. In Section 3 we prove, by using Conjecture 1, that every n-vertex top-view caterpillar has a planar L-shaped drawing on every set of $\frac{25}{3}(n+4)$ points in the plane in general orthogonal position.

2 Rollercoasters

In this section we investigate lower bounds for the length of a longest rollercoaster in a sequence of numbers. We also study algorithmic aspects of computing such rollercoasters. First we prove Conjecture 1: any sequence of n distinct real numbers contains a rollercoaster of length at least $\lceil n/2 \rceil$. Observe that the length 4 sequence $(3, 4, 1, 2)$ has no rollercoaster, so we will restrict to $n \geqslant 5$ in the remainder of this section. Also, due to the following proposition we assume that $n \geqslant 8$.

▶ **Proposition 1.** *Every sequence of $n \in \{5, 6, 7\}$ distinct real numbers contains a rollercoaster of length at least 3. This bound is tight in the worst case.*

Proof. By applying Theorem 2 with $a = b = 2$ we get that every sequence of at least $ab + 1 = 5$ distinct numbers contains an increasing or a decreasing subsequence of length at least 3. This subsequence is a rollercoaster of length at least 3. For the tightness of this bound, consider the sequence $(5, 2, 6, 3, 7, 1, 4)$, depicted in the figure below. It has length 7 and its longest rollercoaster has length 3.



◀

We refer to a polygonal path as a *chain*. We define an *ascent* (resp., a *descent*) as an increasing (resp., a decreasing) sequence. We define a *k-ascent* (resp., a *k-descent*) as an ascent (resp., a descent) with at least k elements. We also use k-ascent and k-descent to refer to increasing and decreasing chains with at least k points, respectively. With this definition, a rollercoaster is a sequence in which every run is either a 3-ascent or a 3-descent. We refer to the rightmost run of a rollercoaster as its *last run*.

2.1 A Proof of Conjecture 1

In this section we prove the following theorem, which is a restatement of Conjecture 1. Our proof is constructive, and yields a linear-time algorithm for finding such a rollercoaster.

▶ **Theorem 4.** *Every sequence of $n \geqslant 8$ distinct real numbers contains a rollercoaster of length at least $\lceil n/2 \rceil$; such a rollercoaster can be computed in linear time. The lower bound of $\lceil n/2 \rceil$ is tight in the worst case.*

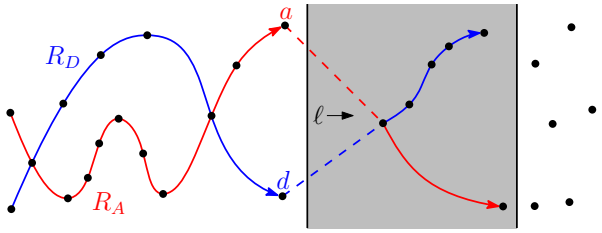


Figure 2 One iteration of algorithm: Constructing two pseudo-rollercoasters.

Consider a sequence with $n \geqslant 8$ distinct real numbers, and let P be its point-set translation with points $p_1, \ldots, p_n$ that are ordered from left to right. We define a *pseudo-rollercoaster* as a sequence in which every run is a 3-ascent or a 3-descent, except possibly the first run. That is, the first run of a pseudo-rollercoaster could be of length at most two, while the other runs are of length at least three. We present an algorithm that computes two pseudo-rollercoasters R_1 and R_2 in P such that $|R_1| + |R_2| \geqslant n$; the length of the longer one is at least $\lceil n/2 \rceil$. Then with a more involved proof we show how to extend this longer pseudo-rollercoaster to obtain a rollercoaster of length at least $\lceil n/2 \rceil$; this will prove the lower bound.

2.1.1 An Algorithm

First we provide a high-level description of our algorithm as depicted in Figure 2. Our algorithm is iterative, and proceeds by sweeping the plane by a vertical line ℓ from left to right. We maintain the following invariant:

▶ **Invariant.** *At the beginning of every iteration we have two pseudo-rollercoasters whose union is the set of all points to the left of ℓ and such that the last run of one of them is an ascent and the last run of the other one is a descent. Furthermore, these two last runs have a point in common.*

During every iteration we move ℓ forward and try to extend the current pseudo-rollercoasters. If this is not immediately possible with the next point, then we move ℓ farther and stop as soon as we are able to split all the new points into two chains that can be appended to the current pseudo-rollercoasters to obtain two new pseudo-rollercoasters that satisfy the invariant. See Figure 2.

Now we present our iterative algorithm in detail.

The First Iteration: We take the leftmost point p_1, and initialize each of the two pseudo-rollercoasters by p_1 alone. We may consider one of the pseudo-rollercoasters to end in an ascent and the other pseudo-rollercoaster to end in a descent. The two runs share p_1.

An Intermediate Iteration: By the above invariant we have two pseudo-rollercoasters R_A and R_D whose union is the set of all points to the left of ℓ and such that the last run of one of them, say R_A, is an ascent and the last run of R_D is a descent. Furthermore, the last run of R_A and the last run of R_D have a point in common. During the current iteration we make sure that every swept point will be added to R_A or R_D or both. We also make sure that at the end of this iteration the invariant will hold for the next iteration. Let a and d denote the rightmost points of R_A and R_D, respectively; see Figure 2. Observe that a lies above d. Let p_i be the first point to the right of ℓ. If p_i is above a, we add p_i to R_A to complete this iteration. Similarly, if p_i is below d, we add p_i to R_D to complete this iteration. In either

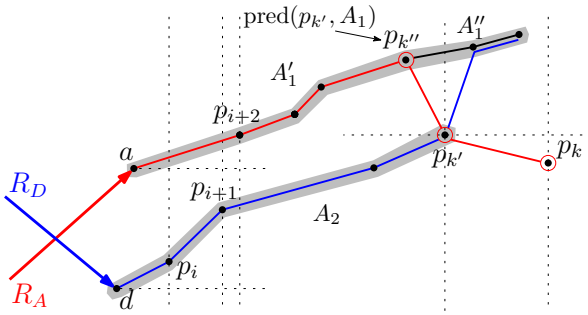


Figure 3 Illustration of an intermediate iteration of the algorithm.

case we get two pseudo-rollercoasters that satisfy the invariant for the next iteration. Thus we may assume that p_i lies below a and above d.

Consider the next point p_{i+1}. (If there is no such point, go to the last iteration; see below.) Suppose by symmetry that p_{i+1} lies above p_i as depicted in Figure 3. Then d, p_i, p_{i+1} forms a 3-ascent. Continue considering points $p_{i+2}, \ldots, p_k$ until for the first time, there is a 3-descent in $a, p_i, \ldots, p_k$. In other words, k is the smallest index for which $a, p_i, \ldots, p_k$ contains a descending chain of length 3. (If we run out of points before finding a 3-descent, then go to the last iteration.)

Without p_k there is no descending chain of length 3. Thus the longest descending chain has two points, and by Theorem 3, the sequence $P' = a, p_i, p_{i+1}, \ldots, p_{k-1}$ is the union of two ascending chains. We give an algorithm to find two such chains A_1 and A_2 with A_1 starting at a and A_2 starting at p_i. The algorithm also finds the 3-descent ending with p_k. For every point $q \in A_2$ we define its A_1-*predecessor* to be the rightmost point of A_1 that is to the left of q. We denote the A_1-predecessor of q by $\text{pred}(q, A_1)$.

The algorithm is as follows: While moving ℓ forward, we denote by r_1 and r_2 the rightmost points of A_1 and A_2, respectively; at the beginning $r_1 = a$, $r_2 = p_i$, and $\text{pred}(p_i, A_1) = a$. Moreover, we maintain this invariant that $\text{pred}(r_2, A_1)$ is above r_2. Let p be the next point to be considered. If p is above r_1 then we add p to A_1. If p is below r_1 and above r_2, then we add p to A_2 and set $\text{pred}(p, A_1) = r_1$; notice that this assignment satisfies the invariant. If p is below r_2, then we find our desired first 3-descent formed by (in backwards order) $p_k = p$, $p_{k'} = r_2$, and $p_{k''} = \text{pred}(r_2, A_1)$. See Figure 3. This algorithm runs in time $O(k - i)$, which is proportional to the number of swept points.

We add point d to the start of chain A_2. The resulting chains A_1 and A_2 are shaded in Figure 3. Observe that A_2 ends at $p_{k'}$. Also, all points of P' that are to the right of $p_{k'}$ (if there are any) belong to A_1, and lie to the right of $p_{k''}$, and form an ascending chain. Let A_1'' be this ascending chain. Let A_1' be the sub-chain of A_1 up to $p_{k''}$; see Figure 3. Now we form one pseudo-rollercoaster (shown in red) consisting of R_A followed by A_1' and then by the descending chain $p_{k''}, p_{k'}, p_k$. We form another pseudo-rollercoaster (shown in blue) consisting of R_D followed by A_2 and then by A_1''. We need to verify that the ascending chain added after d has length at least 3. This chain contains d, p_i and $p_{k'}$. This gives a chain of length at least 3 unless $k' = i$, but in this case $p_{k''} = a$, so p_{i+1} is part of A_1'' (because p_{i+1} is above p_i) and consequently part of this ascending chain. Thus we have constructed two longer pseudo-rollercoasters whose union is the set of all points up to point p_k, one ending with a 3-ascent and one with a 3-descent and such that the last two runs share the point $p_{k'}$. Figure 4(a) shows an intermediate iteration.

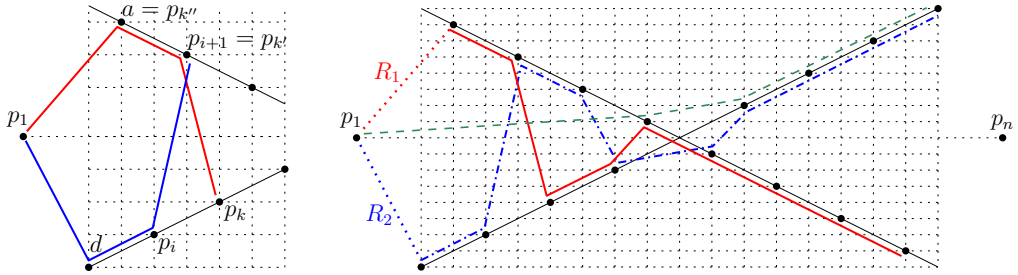


Figure 4 (a) An intermediate iteration. (b) A point set for which any rollercoaster of length at least $n/4 + 3$ does not contain p_1 and p_n. The green (dashed) rollercoaster, which contains p_1, has length $n/4 + 2$. The red (solid) and blue (dash-dotted) chains are the two rollercoasters returned by our algorithm.

The Last Iteration: If there are no points left, then we terminate the algorithm. Otherwise, let p_i be the first point to the right of ℓ. Let a and d be the endpoints of the two pseudo-rollercoasters obtained so far, such that a is the endpoint of an ascent and d is the endpoint of a descent. Notice that p_i is below a and above d, because otherwise this iteration would be an intermediate one. Moreover, the remaining points $p_i, \ldots, p_n$ do not contain a 3-ascent together with a 3-descent, again, because otherwise this iteration would be an intermediate one. If p_i is the last point, i.e., $i = n$, then we discard this point and terminate this iteration. Assume that $i \neq n$, and suppose by symmetry that the next point p_{i+1} lies above p_i. In this setting, by Theorem 3 and as described in an intermediate iteration, with the remaining points, we can get two ascending chains A_1 and A_2 such that A_2 contains at least two points. By connecting A_1 to a and A_2 to d we get two pseudo-rollercoasters whose union is all the points (in this iteration we do not need to maintain the invariant).

Final Refinement: At the end of algorithm, we obtain two pseudo-rollercoasters R_1 and R_2 that share p_1 and such that their union contains all points of P, except possibly p_n. Thus, $|R_1| + |R_2| \geqslant n$, and the length of the longer one is at least $\lceil \frac{n}{2} \rceil$.

Recall that every run of pseudo-rollercoasters R_1 and R_2 is a 3-ascent or a 3-descent, except possibly the first run. If the first run of R_1 (resp., R_2) contains only two points, then we remove p_1 to obtain a rollercoaster $\mathcal{R}_1$ (resp., $\mathcal{R}_2$). Therefore, we obtain two rollercoasters whose union contains all points, except possibly p_1 and p_n.

This is the end of our algorithm. In the next section we analyze the length of the resulting rollercoaster, the tightness of the claimed lower bound, and the running time of the algorithm.

2.1.2 Length and Running-Time Analysis

Our algorithm computes two rollercoasters $\mathcal{R}_1$ and $\mathcal{R}_2$ consisting of all points of P, except possibly p_1 and p_n. Thus, the total length of these rollercoasters is at least $n - 2$, and the length of the longer one is at least $\lceil \frac{n-2}{2} \rceil$. In the full version of the paper (see [3]) we show how to improve this bound to $\lceil \frac{n}{2} \rceil$ by revisiting the first and last iterations of our algorithm with some case analysis.

We note that there are point sets, with n points, for which every rollercoaster of length at least $n/4 + 3$ does not contain any of p_1 and p_n; see e.g., the point set in Figure 4(b). To verify the tightness of the $\lceil n/2 \rceil$ lower bound, consider a set of n points in the plane where

$\lceil n/2 \rceil$ of the points lie on a positive-slope line segment in the $(-,+)$-quadrant and the other $\lfloor n/2 \rfloor$ points lie on a positive-slope line segment in the $(+,-)$-quadrant.

To verify the running time, notice that the first iteration and final refinement take constant time, and the last iteration is essentially similar to an intermediate iteration. As described in an intermediate iteration the time complexity to find a 3-ascent and a 3-descent for the first time, together with the time complexity to compute chains A_1', A_1'', and A_2 is $O(k-i)$, which is linear in the number of swept points $p_i, \ldots, p_k$. Based on this and the fact that every point is considered only in one iteration, our algorithm runs in $O(n)$ time.

2.2 An Extension

In this section we extend our result to k-rollercoasters. A *k-rollercoaster* is a sequence of real numbers in which every run is either a k-ascent or a k-descent.

▶ **Theorem 5.** *Let $k \geqslant 4$ be an integer. Then every sequence of $n \geqslant (k-1)^2+1$ distinct real numbers contains a k-rollercoaster of length at least $\frac{n}{2(k-1)} - \frac{3k}{2}$. Moreover, for every $n > 0$ there exists a sequence of n distinct real numbers whose longest k-rollercoaster has length at most $\lceil \frac{n}{k-1} \rceil$.*

Proof. Our proof of the lower bound follows the same iterative approach of the proof of Theorem 4. Consider a sequence of n distinct real numbers and its point-set translation $p_1, \ldots, p_n$. We sweep the plane by a line ℓ, and maintain two k-rollercoasters R_A and R_D to the left of ℓ such that the last run of R_A is an ascent and the last run of R_D is a descent. In each iteration, except the last one, we move ℓ forward and stop as soon as we see a k-ascent A and a k-descent D in the swept points. Then we attach D to R_A, and A to R_D. To achieve the claimed lower bound, we make sure that the total length of A and D is at least $1/(k-1)$ times the number of swept points.

Consider an intermediate iteration where p_i lies below the rightmost point of R_A and above the rightmost point of R_D. Let m be the number of swept points in this iteration and let $P' = (p_i, p_{i+1} \ldots, p_{i+m-2}, p_{i+m-1})$ be the sequence of these points. Notice that $m \geqslant 2k-1$ because we need to sweep at least $2k-1$ points to get a k-ascent and a k-descent, which may share one point. Our strategy for stopping ℓ ensures that P' contains a k-ascent and a k-descent, while $P'' = (p_i, \ldots, p_{i+m-2})$ may contain only one of them but not both. Without loss of generality assume that P'' does not contain a k-descent. Since $m-1 \geqslant 2k-2$, there exists an integer $\alpha \geqslant 2$ for which

$$(\alpha-1)(k-1) < m-1 \leqslant \alpha(k-1). \tag{1}$$

The left-hand side of Inequality (1) implies that P'' has at least $(\alpha-1)(k-1)+1$ points. Having this and our assumption that P'' does not contain a k-descent, Theorem 2 implies that P'' contains an increasing subsequence of length at least α. We take the longest increasing and the longest decreasing subsequences in P' as A and D, respectively. Observe that $|A| \geqslant \max\{k, \alpha\}$ and $|D| = k$. This and the right-hand side of Inequality (1) imply that

$$|A| + |D| \geqslant \alpha + k \geqslant \frac{m-1}{k-1} + k > \frac{m}{k-1},$$

which means that the total length of A and D is at least $1/(k-1)$ times the number of swept points. In the last iteration if we sweep at most $(k-1)^2$ points then we discard all of them. But if we sweep $m > (k-1)^2$ points then by an argument similar to the one above there exists an integer α, with $\alpha \geqslant m/(k-1)$, for which we get either an α-ascent or an α-descent, which contains at least $1/(k-1)$ fraction of the swept points.

The first iteration is similar to the one in the proof of Theorem 4: we assume the existence of an ascent and a descent that end at the first point. At the end of algorithm if the first run of any of R_A and R_D contains k' points, for some $k' < k$, then by removing $k' - 1 (\leqslant k-2)$ points from its first run we get a valid k-rollercoaster. The total length of the resulting two k-rollercoasters is

$$|R_A| + |R_D| \geqslant \frac{n-(k-1)^2}{k-1} - 2(k-2),$$

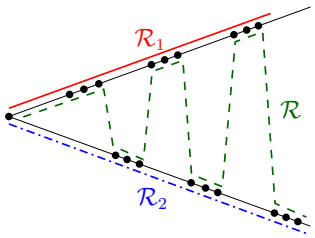
where the length of the longer one is at least

$$\frac{n-(k-1)^2}{2(k-1)} - (k-2) > \frac{n}{2(k-1)} - \frac{3(k-1)}{2}.$$

This finishes our proof of the lower bound. To verify the upper bound, consider a set of n points that are placed in the main-diagonal cells of a $(k-1) \times (k-1)$ grid, such that every cell contains at most $\lceil \frac{n}{k-1} \rceil$ points that are placed on a positive-slope line. ◀

2.3 Algorithms for a Longest Rollercoaster

In this section we study algorithmic aspects of computing a longest rollercoaster in a given sequence S of n distinct real numbers. By Theorem 4 we can compute a rollercoaster of length at least $\lceil n/2 \rceil$ in $O(n)$ time. However this rollercoaster may not necessarily be a longest one. If we run our algorithm of Section 2.1.1 on the point set in the figure below, then it returns two rollercoasters $\mathcal{R}_1$ and $\mathcal{R}_2$ each of length at most $\lceil \frac{n}{2} \rceil$ while the longest rollercoaster $\mathcal{R}$ has length n. In this section, first we adapt the existing $O(n \log n)$-time algorithm for computing a longest increasing subsequence in S to compute a longest rollercoaster in S within the same time bound. Then we show that if S is a permutation of $\{1, \ldots, n\}$, then we can compute a longest rollercoaster in $O(n \log \log n)$ time.



First we recall Fredman's version of the $O(n \log n)$-time algorithm for computing a longest increasing subsequence [9]; for more information about longest increasing subsequence, see Romik [14]. We maintain an array $R[i]$, which initially has $R[1] = S[1]$ and is empty otherwise. Then as i proceeds from 2 to n, we find the largest l for which $R[l] < S[i]$, and set $R[l+1] = S[i]$; if $S[i]$ is smaller than all elements of R, then $l = 0$. This insertion ensures that every element $R[l]$ stores the smallest element of $S[1..i]$ in which an increasing subsequence of length l ends. After all elements of S have been processed, the index of the last non-empty element of R is the largest length of an increasing sequence; the corresponding sequence can also be retrieved from R. Notice that R is always sorted during the above process. So, the proper location of $S[i]$ in R can be computed in $O(\log n)$ time by a predecessor search, which can be implemented as a binary search. Therefore, this algorithm runs in $O(n \log n)$ time.

To compute a longest rollercoaster we need to extend this approach. We maintain six arrays $R(w, h)$ with $w \in \{\text{inc}, \text{dec}\}$ and $h \in \{2, 3_+, 3'_+\}$ where inc stands for "increasing", dec stands for "decreasing", and both 3_+ and $3'_+$ stand for any integer that is at least 3 (we will see the difference between 3_+ and $3'_+$ later when we fill the arrays). We define a

w-h-rollercoaster to be a rollercoaster whose last run has h points and is increasing if $w = \text{inc}$ and decreasing if $w = \text{dec}$. We insert $S[i]$ into arrays $R(\text{inc}, h)$ such that after this insertion the following invariants hold:

- The array $R(\text{inc}, 2)[l]$ stores the smallest element of $S[1..i]$ in which an inc-2-rollercoaster of length l ends. The array $R(\text{dec}, 2)[l]$ stores the largest element of $S[1..i]$ in which a dec-2-rollercoaster of length l ends.
- The array $R(\text{inc}, 3_+)[l]$ stores the smallest element of $S[1..i]$ in which an inc-3_+-rollercoaster of length l ends. The array $R(\text{dec}, 3_+)[l]$ stores the largest element of $S[1..i]$ in which an dec-3_+-rollercoaster of length l ends.
- The array $R(\text{inc}, 3'_+)[l]$ stores the largest element of $S[1..i]$ in which an inc-3_+-rollercoaster of length l ends. The array $R(\text{dec}, 3'_+)[l]$ stores the smallest element of $S[1..i]$ in which a dec-3_+-rollercoaster of length l ends. These arrays will be used when the last run of the current rollercoaster changes from an ascent to a descent, and vice versa.

We insert $S[i]$ into arrays $R(\text{dec}, h)$ so to maintain analogous aforementioned invariants. To achieve these invariants we insert $S[i]$ as follows:

- $R(\text{inc}, 2)$: Find the largest index l such that $R(\text{dec}, 3'_+)[l] < S[i]$. If $S[i] < R(\text{inc}, 2)[l+1]$ then update $R(\text{inc}, 2)[l+1] = S[i]$.
- $R(\text{inc}, 3_+)$: Find the largest indices l_1 and l_2 such that $R(\text{inc}, 2)[l_1] < S[i]$ and $R(\text{inc}, 3_+)[l_2] < S[i]$. Let $l = \max\{l_1, l_2\}$. If $S[i] < R(\text{inc}, 3_+)[l+1]$ then update $R(\text{inc}, 3_+)[l+1] = S[i]$.
- $R(\text{inc}, 3'_+)$: Find the largest index l_1 and l_2 such that $R(\text{inc}, 2)[l_1] < S[i]$ and $R(\text{inc}, 3'_+)[l_2] < S[i]$. Let $l = \max\{l_1, l_2\}$. If $S[i] > R(\text{inc}, 3'_+)[l+1]$ then update $R(\text{inc}, 3'_+)[l+1] = S[i]$.
- The arrays $R(\text{dec}, h)$ are updated in a similar fashion.

Since our arrays $R(w, h)$ are not necessarily sorted, we cannot perform a predecessor search to find proper locations of $S[i]$. To insert $S[i]$ we need to *find* the largest index l such that $R(w, h)[l]$ is smaller (or, alternatively, larger) than $S[i]$ for some w and h, and also need to *update* contents of these arrays. Thereby, if A is an $R(w, h)$ array, we need to perform the following two operations on A:

- FindMax$(A, S[i])$: Find the largest index l such that $A[l] > S[i]$ (or $A[l] < S[i]$).
- Update$(A, l, S[i])$: Set $A[l] = S[i]$.

We implement each $R(w, h)$ as a Fenwick tree [8], which supports FindMax and Update in $O(\log n)$ time. Thus, the total running time of our algorithm is $O(n \log n)$. After all elements of S have been processed, the largest length of a rollercoaster is the largest value l for which $R(w, 3_+)[l]$ or $R(w, 3'_+)[l]$ is not empty; the corresponding rollercoaster can also be retrieved from arrays $R(w, h)$, by keeping the history of the way the elements of these arrays were computed, and then rolling back the computation.

A Longest Rollercoaster in Permutations: Here we consider a special case where our input sequence S consists of n distinct integers, each of which can be represented using at most c memory words for some constant $c \geqslant 1$, in a RAM model with logarithmic word size. In linear time, we can sort S, using Radix Sort, and then hash it to a permutation of $\{1, \ldots, n\}$. This reduces the problem to finding a longest rollercoaster in a permutation of $\{1, \ldots, n\}$. The longest increasing subsequence of such a sequence can be computed in $O(n \log \log n)$ time by using a van Emde Boas tree [18], which supports predecessor search and updates in $O(\log \log n)$ time.[8] To compute a longest rollercoaster in the same time, we need a data

[8] We note that a longest increasing subsequence of a permutation can also be computed in $O(n \log \log k)$ time (see [5]) where k is the largest length of an increasing sequence. However, in our case, the largest

structures that supports FindMax and Update in permutations in $O(\log\log n)$ time. In the full version of the paper (see [3]) we show how to obtain such a data structure by using van Emde Boas trees combined with some other structures.

▶ **Lemma 6.** *Let A be an array with n elements from the set $\{0, 1, \ldots, n\}$ such that each non-zero number occurs at most once in A. We can construct, in linear time, a data structure that performs* FindMax *and* Update *operations in $O(\log\log n)$ amortized time.*

With Lemma 6 in hand, we can compute a longest rollercoaster in S in $O(n \log\log n)$ time. We note that this algorithm can also compute a longest increasing subsequence by maintaining only the array $R(\text{inc}, 3_+)$.

Notice that both of our algorithms (for general sequences and for permutations) can be generalized to compute a longest k-rollercoaster in $O(kn \log n)$ time and in $O(kn \log\log n)$ time, respectively. A straightforward way is to maintain $2k$ arrays $R(w, h)$ with $w \in \{\text{inc}, \text{dec}\}$ and $h \in \{2, \ldots, k-1, k_+, k'_+\}$ and fill them in a way analogous to what we did for rollercoasters. The following theorem summarizes our results in this section.

▶ **Theorem 7.** *Let $k \geqslant 3$ be an integer. Then a longest k-rollercoaster in every sequence of n distinct real numbers can be computed in $O(kn \log n)$ time, and a longest k-rollercoaster in every permutation of $\{1, \ldots, n\}$ can be computed in $O(kn \log\log n)$ time.*

2.4 Counting Rollercoaster Permutations

In this section we estimate the number $r(n)$ of permutations of $\{1, 2, \ldots, n\}$ that are rollercoasters. A brief table follows:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$r(n)$	1	0	2	2	14	42	244	1208	7930	52710	40580	3310702	29742388	285103536

This is sequence A277556 in the *On-Line Encyclopedia of Integer Sequences* [16].

The first step is to rephrase the condition that a permutation is a rollercoaster in the language of ascents and descents. Given a length-n permutation $\pi = \pi_1\pi_2\cdots\pi_n$, its *descent word* $u(\pi)$ is defined to be $u_1u_2\cdots u_{n-1}$ where $u_i = \mathbf{a}$ if $\pi_i < \pi_{i+1}$ and $\mathbf{b}$ otherwise. For example if $\pi = 2, 4, 6, 1, 3, 5$, then $u(\pi) = \mathbf{aabaa}$. Notice that π is a rollercoaster if and only if every maximal contiguous subsequence of $u(\pi)$, that consists of only $\mathbf{a}$'s or $\mathbf{b}$'s, has length at least two. In other words, π is a rollercoaster if and only if $u(\pi)$ does not have an isolated $\mathbf{a}$ or an isolated $\mathbf{b}$; in fact $u(\pi)$ does not contain patterns $\{\mathbf{aba}, \mathbf{bab}\}$, and also begins and ends with either $\mathbf{aa}$ or $\mathbf{bb}$. The set of all such descent words $u(\pi)$ is given by the expression

$$(\mathbf{aaa}^* + \mathbf{bbb}^*)^*.$$

This expression specifies that every increasing run and every decreasing run must contain at least three elements. Since this description is a regular expression, one can, in principle, obtain the asymptotic behavior of $r(n)$ using the techniques of [2], but the calculations appear to be formidable.

Instead, we follow the approach of Ehrenborg and Jung [6]. This is based on specifying sets of permutations through pattern avoidance. We say a word w *avoids* a set of words S if no contiguous subword of w belongs to S. Although rollercoasters are not specifiable in terms of a finite set of avoidable patterns, they "almost are". Consider the patterns

length of a rollercoaster is $\Omega(n)$.

$\{\mathbf{aba}, \mathbf{bab}\}$. Every descent word of a rollercoaster must avoid both these patterns, and every word avoiding these patterns that *also* begins and ends with either **aa** or **bb** is the descent word of some rollercoaster. Let $s(n)$ be the number of permutations of length n whose descent word avoids $\{\mathbf{aba}, \mathbf{bab}\}$. Then $r(n) = \Theta(s(n))$. From [6, Prop. 5.2] we know that $s(n) \sim c \cdot n! \cdot \lambda^{n-3}$ where $\lambda \doteq 0.6869765032\cdots$ is the root of a certain equation. It follows that $r(n) \sim c' \cdot n! \cdot \lambda^{n-3}$ where c' is a constant, approximately 0.204.

3 Caterpillars

In this section we study the problem of drawing a top-view caterpillar, with L-shaped edges, on a set of points in the plane that is in general orthogonal position. Recall that a top-view caterpillar is an ordered caterpillar of degree 4 such that the two leaves adjacent to each vertex lie on opposite sides of the spine; see Figure 1(b) for an example. The best known upper bound on the number of required points for a planar L-shaped drawing of the n-vertex top-view caterpillar is $O(n \log n)$, for all n; this bound is due to Biedl et al. [4]. We use Theorem 4 and improve this bound to $\frac{25}{3}n+O(1)$.

In every planar L-shaped drawing of a top-view caterpillar, every node of the spine, except for the two endpoints, must have its two incident spine edges aligned either horizontally or vertically. Such a drawing of the spine (which is essentially a path) is called a *straight-through* drawing. It has been proved in [4] that every n-vertex path has an x-monotone straight-through drawing on any set of at least $c \cdot n \log n$ points, for some constant c. The following theorem improves this bound.

▶ **Theorem 8.** *Any path of n vertices has an x-monotone straight-through drawing on any set of at least $3n-3$ points in the plane that is in general orthogonal position.*

Proof. Fix an arbitrary set of $3n-3$ points. As in the proof of Theorem 4, find two pseudo-rollercoasters that together cover all but the last point and that both contain the first point. Append the last point to both sets; we hence obtain two subsequences R_1, R_2 with $|R_1| + |R_2| \geqslant 3n-1$ and for which all but the first and last run have length at least 3.

We may assume $|R_1| \geqslant \frac{3}{2}n - \frac{1}{2}$, and will find the straight-through drawing within it. To do so, consider any run r of R_1 that is neither the first nor the last run, and that has even length (hence length at least 4). By removing from r one point that is not shared with an adjacent run, we turn it into a run of odd length. Let R' be the subsequence that results after applying this to every such run of R_1; then R' satisfies that every run except the first and last one has odd length. Observe that we can find an x-monotone straight-through drawing of length $|R'|$ on this, see e.g. the black path in Figure 5 that is drawn on the black points.

It remains to argue that $|R'| \geqslant n$. Let $r_1, \ldots, r_\ell$ be the runs of R_1, and assign to each run r_i all but the last point of r_i (the last point of r_i is counted with r_{i+1}, or not counted at all if $i = \ell$). Therefore $|r_1| + \cdots + |r_\ell| = |R_1|-1 \geqslant \frac{3}{2}n - \frac{3}{2}$. For each r_i with $2 \leqslant i \leqslant \ell-1$, we remove a point only if $|r_i| \geqslant 3$, hence we keep at least $\frac{2}{3}|r_i|$ points. Therefore $|R'| \geqslant |r_1|+|r_\ell|+\sum_{2\leqslant i\leqslant \ell-1} \frac{2}{3}|r_i| = \sum_{1\leqslant i\leqslant \ell} \frac{2}{3}|r_i| + \frac{1}{3}(|r_1|+|r_\ell|) \geqslant \frac{2}{3}\cdot(\frac{3}{2}n - \frac{3}{2}) + \frac{1}{3}(2+1) = n$ as desired. ◀

To draw top-view caterpillars, we essentially use Theorem 8 and place the spine on the resulting straight-through x-monotone path. But we will get a slightly better factor if we analyze the number of points directly.

▶ **Theorem 9.** *For every n, the n-vertex top-view caterpillar has a planar L-shaped drawing on any set of $\frac{25}{3}(n+4)$ points in the plane that is in general orthogonal position. Such a*

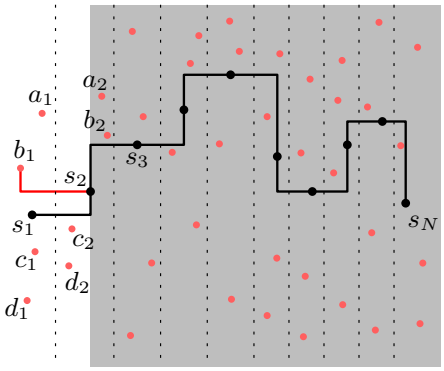


Figure 5 An x-monotone straight-through drawing of an N-vertex path. Red (lighter shade) points are reserved.

drawing can be computed in $O(n)$ time, provided that the input points are given in sorted order.

Proof. Fix any $\frac{25}{3}(n+4)$ points P in general orthogonal position. Partition P, by vertical lines, into $\frac{5}{3}(n+4)$ sets, each of them containing five points. We call every such set a *5-set*. Let P' be the set of the mid-points (with respect to y-coordinates) of every 5-set. We have $|P'| \geqslant \frac{5}{3}(n+4)$, so by Theorem 4 it contains a rollercoaster R of length at least $\frac{5}{6}(n+4)$.

Let $s_1, \ldots, s_N$ (for $N \geqslant \frac{5}{6}(n+4)$) be the points of R, ordered from left to right. For every $i \in \{1, \ldots, N\}$, consider the 5-set containing s_i and let its other points be a_i, b_i, c_i, d_i from top to bottom; we call these the *reserved points*. The main idea is to draw the spine of the caterpillar along R and the leaves at reserved points, though we will deviate from this occasionally. Let the spine consist of vertices $v_1, v_2, \ldots, v_\ell$, where v_1 and v_ℓ are leaves while $v_2, v_3, \ldots$ are vertices of degree 4. We process the vertices in order along the spine, and maintain the following invariant:

▶ **Invariant.** *At time $k \geqslant 1$, vertex v_{2k} is drawn on a point s_i that is not an extremal vertex of a run of R. Edge (v_{2k}, v_{2k-1}) attaches vertically at v_{2k}. All vertices $v_1, \ldots, v_{2k-1}$, all their incident leaves, and one incident leaf of v_{2k} are drawn on points to the left of s_i.*

By a suitable reflection we may assume that s_2 is above s_1. To initiate this process, we draw v_1 on s_1, v_2 on s_2, and one leaf incident to s_2 on b_1. See Figure 5. Clearly the invariant holds for v_2. Now assume that vertex v_{2k} has been placed at s_i, and we want to place v_{2k+1} and v_{2k+2} next. We know that s_i is in the middle of some run of R; up to symmetry we may assume that it is an ascending run. Let s_j be the last point of this run of R; by the invariant $j > i$. We distinguish cases:

Case 1: $j \leqslant i + 4$. See Figure 6(a). We will completely ignore the 5-sets containing $s_{i+1}, \ldots, s_{j-1}$. Recall that there are two reserved points a_j and b_j above s_j. We place v_{2k+1} at b_j and v_{2k+2} at s_{j+1}. We connect leaves as follows: The leaves incident to v_{2k+1} are placed at a_j and s_j. To place one leaf each incident to v_{2k} and v_{2k+2}, we use the two points c_j and d_j, using the one farther left for v_{2k}. Clearly the invariant holds.

Observe that there are at most five 5-sets (corresponding to $s_{i+1}, \ldots, s_{j+1}$) that were parsed, and we have used two for placing spine-vertices. Therefore, we have used at least $\frac{2}{5}$th of the parsed 5-sets.

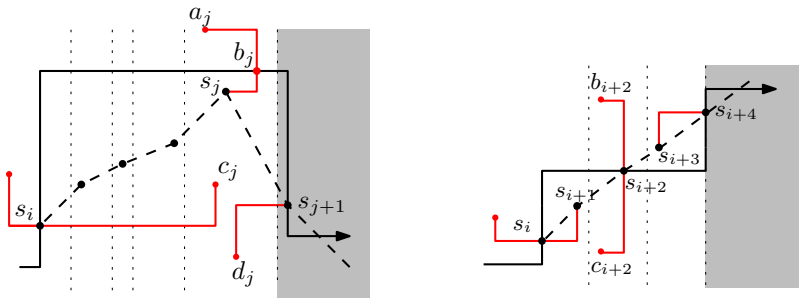


Figure 6 Placing the next two spine-vertices. (a) $j \leqslant i+4$. (b) $j > i+4$. The dashed line indicates R, the solid line is the spine.

Case 2: $j > i+4$. See Figure 6(b). We ignore the reserved points corresponding to s_{i+1} and s_{i+3}. We place v_{2k+1} at s_{i+2} and v_{2k+2} at s_{i+4}. Note that by case-assumption s_{i+4} is *not* the end of the run, so this satisfies the invariant. We connect, as leaves, s_{i+1} to v_{2k} (at s_i), and s_{i+3} to v_{2k+2} (at s_{i+4}). The two leaves of v_{2k+1} can be placed in the 5-set of s_{i+2}. We have used four 5-sets and placed two spine-vertices, and have therefore used half of the parsed 5-sets.

This is the end of one iteration. In every iteration, we have used at least $\frac{2}{5}$th of the parsed 5-sets. Since there were $\frac{5}{6}(n+4)$ 5-sets, we hence can place $\frac{1}{3}(n+4)$ spine-vertices. Since the spine of the n-vertex top-view caterpillar has $\frac{1}{3}(n+4)$ vertices, our claim about the size of the input point set follows. If the input points are given in sorted order, we can find the rollercoaster in linear time, and then we do one scan of the points to find a planar L-shaped drawing. Thus, our claim about the running time follows. ◀

References

1 Tanbir Ahmed and Hunter Snevily. Some properties of roller coaster permutations. *Bulletin of the Institute of Combinatorics and its Applications*, 68:55–69, 2013.

2 Nicolas Basset. Counting and generating permutations in regular classes. *Algorithmica*, 76(4):989–1034, 2016.

3 Therese C. Biedl, Ahmad Biniaz, Robert Cummings, Anna Lubiw, Florin Manea, Dirk Nowotka, and Jeffrey Shallit. Rollercoasters and caterpillars. *CoRR*, abs/1801.08565, 2018. `arXiv:1801.08565`.

4 Therese C. Biedl, Timothy M. Chan, Martin Derka, Kshitij Jain, and Anna Lubiw. Improved bounds for drawing trees on fixed points with L-shaped edges. In *Proceedings of the 25th International Symposium on Graph Drawing and Network Visualization* (*GD*), volume 10692 of *Lecture Notes in Computer Science*, pages 305–317. Springer, 2017. Full version in arXiv:1709.01456.

5 Maxime Crochemore and Ely Porat. Fast computation of a longest increasing subsequence and application. *Information and Computation*, 208(9):1054–1059, 2010.

6 R. Ehrenborg and J. Jung. Descent pattern avoidance. *Advances in Applied Mathematics*, 49:375–390, 2012.

7 Paul Erdős and George Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.

8 Peter M. Fenwick. A new data structure for cumulative frequency tables. *Software—Practice and Experience*, 24(3):327–336, 1994.

9 Michael L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975.

10 Emilio Di Giacomo, Fabrizio Frati, Radoslav Fulek, Luca Grilli, and Marcus Krug. Orthogeodesic point-set embedding of trees. *Computational Geometry: Theory and Applications*, 46(8):929–944, 2013.

11 John M. Hammersley. A few seedlings of research. In *Proceedings of the 6th Berkeley Symposium on Mathematical Statistics and Probability*, pages 345–394. University of California Press, 1972.

12 Sergey Kitaev. *Patterns in Permutations and Words*. Springer, 2011.

13 S. Linton, N. Ruškuc, and V. Vatter, editors. *Permutation Patterns*. London Mathematical Society Lecture Note Series, vol. 376, Cambridge, 2010.

14 Dan Romik. *The Surprising Mathematics of Longest Increasing Subsequences*. Cambridge, 2015.

15 Manfred Scheucher. Orthogeodesic point set embeddings of outerplanar graphs. Master's thesis, Graz University of Technology, 2015.

16 N. J. A. Sloane et al. The On-Line Encylopedia of Integer Sequences. URL: `https://oeis.org`.

17 J. Michael Steele. Variations on the monotone subsequence theme of Erdős and Szekeres. In David Aldous, Persi Diaconis, Joel Spencer, and J. Michael Steele, editors, *Discrete Probability and Algorithms*, pages 111–131. Springer New York, 1995.

18 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80–82, 1977.

New algorithms for Steiner tree reoptimization

Davide Bilò
Department of Humanities and Social Sciences, University of Sassari
Via Roma 151, 07100 Sassari (SS), Italy
davide.bilo@uniss.it
https://orcid.org/0000-0003-3169-4300

Abstract

Reoptimization is a setting in which we are given an (near) optimal solution of a problem instance and a local modification that slightly changes the instance. The main goal is that of finding an (near) optimal solution of the modified instance.

We investigate one of the most studied scenarios in reoptimization known as *Steiner tree reoptimization*. Steiner tree reoptimization is a collection of strongly NP-hard optimization problems that are defined on top of the classical Steiner tree problem and for which several constant-factor approximation algorithms have been designed in the last decade. In this paper we improve upon all these results by developing a novel technique that allows us to design *polynomial-time approximation schemes*. Remarkably, prior to this paper, no approximation algorithm better than recomputing a solution from scratch was known for the elusive scenario in which the cost of a single edge decreases. Our results are best possible since none of the problems addressed in this paper admits a fully polynomial-time approximation scheme, unless P = NP.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Steiner tree problem, reoptimization, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.19

Related Version A full version of the paper is available at `https://arxiv.org/abs/1804.10791`.

1 Introduction

Reoptimization is a setting in which we are given an instance I of an optimization problem together with an (near) optimal solution S for I and a local modification that, once applied to I, generates a new instance I' which slightly differs from I. The main goal is that of finding an (near) optimal solution of I'. Since reoptimization problems defined on top of NP-hard problems are usually NP-hard, the idea beyond reoptimization is to compute a good approximate solution of I' by exploiting the structural properties of S. This approach leads to the design of reoptimization algorithms that, when compared to classical algorithms that compute feasible solutions from scratch, are less expensive in terms of running time or output solutions which guarantee a better approximation ratio. Thus, reoptimization finds applications in many real settings (like scheduling problems or network design problems) where prior knowledge is often at our disposal and a problem instance can arise from a local modification of a previous problem instance (for example, a scheduled job is canceled or some links of the network fail).

The term reoptimization was mentioned for the first time in the paper of Schäffter [33], where the author addressed an NP-hard scheduling problem with forbidden sets in the scenario of adding/removing a forbidden set. Since then, reoptimization has been successfully applied to other NP-hard problems including: the Steiner tree problem [8, 11, 13, 14, 15, 29, 30, 35], the traveling salesman problem and some of its variants [1, 5, 7, 12, 15, 16, 17, 18, 19, 32],

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 19; pp. 19:1–19:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

scheduling problems [6, 21], the minimum latency problem [27], the minimum spanning tree problem [24], the rural postman problem [3], many covering problems [10], the shortest superstring problem [9], the knapsack problem [2], the maximum-weight induced hereditary subgraph problems [22], and the maximum P_k subgraph problem [23]. Some overviews on reoptimization can be found in [4, 25, 34].[1]

1.1 Our results

In this paper we address *Steiner tree reoptimization*. Steiner tree reoptimization is a collection of optimization problems that are built on top of the famous *Steiner tree problem* (*STP* for short), a problem that given an edge-weighted graph as input, where each vertex can be either terminal or Steiner, asks to find a minimum-cost tree spanning all the terminals. More precisely, we study the four local modifications in which the cost of an edge changes (it can either increase or decrease) or a vertex changes its status (from Steiner to terminal and viceversa).[2] All the problems addressed are strongly NP-hard [14, 15], which implies that none of them admits a fully polynomial-time approximation scheme, unless $\mathsf{P} = \mathsf{NP}$.

In this paper we improve upon all the constant-factor approximation algorithms that have been developed in the last 10 years by designing *polynomial-time approximation schemes*. More precisely, if S is a ρ-approximate solution of I, then all our algorithms compute a $(\rho + \epsilon)$-approximate solution of I'. For the scenarios in which the cost of an edge decreases or a terminal vertex becomes Steiner, our polynomial-time approximation schemes hold under the assumption that $\rho = 1$, i.e., the provided solution is optimal. We observe that this assumption is somehow necessary since Goyal and Mömke (see [30]) proved that in both scenarios, unless $\rho = 1$, any σ-approximation algorithm for the reoptimization problem would be also a σ-approximation algorithm for the *STP*! Remarkably, prior to this paper, no approximation algorithm better than recomputing a solution from scratch was known for the elusive scenario in which the cost of a single edge decreases.

The state of the art of Steiner tree reoptimization is summarized in Table 1. We observe that the only problem that remains open is the design of a reoptimization algorithm for the scenario in which a vertex, that can be either terminal or Steiner, is added to the graph. In fact, as proved by Goyal and Mömke [30], the scenario in which a vertex is removed from the graph is as hard to approximate as the *STP* (think of the removal of a vertex that is connected to each terminal by an edge of cost 0).

1.2 Used techniques

All the polynomial-time approximation schemes proposed in this paper make a clever use of the following three algorithms:

- the algorithm Connect that computes a Steiner tree in polynomial time by augmenting a Steiner forest having a constant number of trees with a minimum-cost set of edges. This algorithm has been introduced by Böckenhauer et al. in [14] and has been subsequently used in several other papers on Steiner tree reoptimization [11, 13, 30, 35];
- the algorithmic proof of Borchers and Du [20] that, for every $\xi > 0$, converts a Steiner tree S into a $f(\xi)$-*restricted Steiner tree* S_ξ, for some function $f(\xi)$ that depends on ξ only, whose cost is a $(1 + \xi)$-factor away from the cost of S. This algorithm has been used for the first time in Steiner tree reoptimization by Goyal and Mömke [30];

[1] In this paper we focus only on reoptimization problems that are NP-hard.

[2] We observe that the insertion of an edge can be modeled by decreasing the cost of the edge from $+\infty$ to its real value. Analogously, the deletion of an edge can be modeled by increasing the cost of the edge to $+\infty$.

Table 1 The state of the art of the approximability of Steiner tree reoptimization before and after our paper. The value σ is the best approximation ratio we can achieve for *STP* (actually $\sigma = \ln 4 + \epsilon \approx 1.387$ [26]). If we substitute $\sigma = \ln 4 + \epsilon$, we get $\frac{10\sigma-7}{7\sigma-4} \approx 1.204$, $\frac{7\sigma-4}{4\sigma-1} \approx 1.256$, and $\frac{5\sigma-3}{3\sigma-1} \approx 1.246$. The bound of $\frac{5\sigma-3}{3\sigma-1}$ for the scenario in which the cost of an edge decreases holds when the modified edge can affect only its cost in the shortest-path metric closure. To increase readability, all the bounds are provided for the case in which $\rho = 1$. Local modifications marked with an asterisk are those for which the condition $\rho = 1$ is necessary as otherwise the reoptimization problem would be as hard to approximate as the *STP*. Finally, the question mark means that the problem is still open.

Local modification	**before our paper**	**after our paper**
a terminal becomes a Steiner vertex*	$\frac{10\sigma-7}{7\sigma-4} + \epsilon$ [30]	$1 + \epsilon$
a Steiner vertex becomes terminal	$\frac{10\sigma-7}{7\sigma-4} + \epsilon$ [30]	$1 + \epsilon$
an edge cost increases	$\frac{7\sigma-4}{4\sigma-1} + \epsilon$ [30]	$1 + \epsilon$
an edge cost decreases*	$\frac{5\sigma-3}{3\sigma-1}$ [8]	$1 + \epsilon$
a vertex is added to the graph*	?	?
a vertex is deleted from the graph	as hard as the *STP* [30]	as hard as the *STP* [30]

- a novel algorithm, that we call BuildST, that takes a $f(\xi)$-restricted Steiner forest S_ξ of S with q trees, generated with the algorithm of Borchers and Du, and a positive integer h as inputs and computes a minimum-cost Steiner tree S' w.r.t. the set of all the feasible solutions that can be obtained by swapping up to h *full components* of S_ξ with a minimum-cost set of edges (that is computed using the algorithm Connect). The running time of this algorithm is polynomial when all the three parameters $f(\xi), q$, and h are bounded by a constant.

We prove that the approximation ratio of the solution S' returned by the algorithm BuildST is $\rho+\epsilon$ (i) by showing that the cost of S' is at most the cost of other feasible solutions $S_1, \ldots, S_\ell$ and (ii) by proving useful upper bounds to the cost of each S_i's. Intuitively, each S_i is obtained by swapping up to h suitable full components, say H_i, of a suitable $f(\xi)$-restricted version of a forest in S, say S_ξ, with a minimal set of full components, say H'_i, of a $g(\xi)$-restricted version of a fixed optimal solution OPT'. This implies that we can easily bound the cost of each S_i w.r.t. the cost of S_ξ, the cost of H_i, and the cost of H'_i. Unfortunately, none of these bounds can be used by itself to prove the claim. We address this issue by carefully defining H_{i+1} as a function of both OPT' and H'_i, and H'_i as a function of both S_ξ and H_i. This trick allows us to derive better upper bounds: we prove that the cost of each S_i is at most $\rho g(\xi)^2$ times the cost of OPT' plus the i-th term of a telescoping sum. As each term of the telescoping sum can be upper bounded by $g(\xi)$ times the cost of OPT', the bound of $(\rho + \epsilon)$ on the approximation ratio of S' then follows by averaging the costs of the S_i's and because of the choices of the parameters.

The paper is organized as follows: in Section 2 we provide the basic definitions and some preliminaries; in Section 3 we describe the three main tools that are used in all our algorithms; in Sections 4 and 5 we describe and analyze the algorithms for the cases in which a Steiner vertex becomes terminal and the cost of an edge increases, respectively. Due to the lack of space, the local modifications in which a terminal becomes a Steiner vertex and the cost of an edge decreases can be found in the full version of the paper.

2 Basic definitions and preliminaries

The Steiner tree problem (*STP* for short) is defined as follows:

- the input is a triple $I = \langle G, c, R\rangle$ where $G = (V(G), E(G))$ is a connected undirected graph with $|V(G)| = n$ vertices, c is a function that associates a real value $c(e) \geq 0$ to each edge $e \in E(G)$, and $R \subseteq V(G)$ is a set of *terminal* vertices;
- the problem asks to compute a minimum-cost Steiner tree of I, i.e., a tree that spans R and that minimizes the overall sum of its edge costs.

The vertices in $V(G) \setminus R$ are also called *Steiner* vertices. With a slight abuse of notation, for any subgraph H of G, we denote by $c(H) := \sum_{e\in E(H)} c(e)$ the cost of H.

In *Steiner tree reoptimization* we are given a triple $\langle I, S, I'\rangle$ where I is an *STP* instance, S is a ρ-approximate Steiner tree of I, and I' is another *STP* instance which slightly differs from I. The problem asks to find a ρ-approximate Steiner tree of I'.

For a forest F of G (i.e., with $E(F) \subseteq E(G)$) and a set of edges $E' \subseteq E(G)$, we denote by $F + E'$ a (fixed) forest yielded by the addition of all the edges in $E' \setminus E(F)$ to F, i.e., we scan all the edges of E' one by one in any (fixed) order and we add the currently scanned edge to the forest only if the resulting graph remains a forest. Analogously, we denote by $F - E'$ the forest yielded by the removal of all the edges in $E' \cap E(F)$ from F. We use the shortcuts $F + e$ and $F - e$ to denote $F + \{e\}$ and $F - \{e\}$, respectively. Furthermore, if F' is another forest of G we also use the shortcuts $F + F'$ and $F - F'$ to denote $F + E(F')$ and $F - E(F')$, respectively. For a set of vertices $U \subseteq V(G)$, we define $F + U := (V(F) \cup U, E(F))$ and use the shortcut $F + v$ to denote $F + \{v\}$.

W.l.o.g., in this paper we tacitly assume that all the leaves of a Steiner tree are terminals: if a Steiner tree S contains a leaf which is not a terminal, then we can always remove such a leaf from S to obtain another Steiner tree whose cost is upper bounded by the cost of S. Analogously, we tacitly assume that all the leaves of a Steiner forest (i.e., a forest spanning all the terminals) are terminals. A *full component* of a Steiner forest F – so a Steiner tree as well – is a maximal (w.r.t. edge insertion) subtree of F whose leaves are all terminals and whose internal vertices are all Steiner vertices. We observe that a Steiner forest can always be decomposed into full components. A *k-restricted* Steiner forest is a Steiner forest where each of its full components has at most k terminals.

In the rest of the paper, for any superscript s (the empty string is included), we denote by I^s an *STP* instance (we tacitly assume that $I^s = \langle G^s, c^s, R^s\rangle$ as well as that G^s has n vertices), by OPT^s a fixed optimal solution of I^s, by d^s the shortest-path metric induced by c^s in G^s (i.e, $d^s(u, v)$ is equal to the cost of a shortest path between u and v in G^s w.r.t. cost function c^s), by K^s the complete graph on $V(G^s)$ with edge costs d^s, and by K^s_n the complete graph obtained by adding $n - 1$ copies of K^s to K^s and where the cost of an edge between a vertex and any of its copies is equal to 0. With a little abuse of notation, we use d^s to denote the cost function of K^s_n. Moreover, we say that a subgraph H of K^s_n *spans* a vertex $v \in V(G^s)$ if H spans – according to the standard terminology used in graph theory – any of the n copies of v. Finally, with a little abuse of notation, we denote by I^s_n the *STP* instance $I^s_n = \langle K^s_n, d^s, R^s\rangle$.

Let I be an *STP* instance. We observe that any forest F of G is also a forest of K such that $d(F) \leq c(F)$. Moreover, any forest $\bar{F}$ of K can be transformed in polynomial time into a forest F of G spanning $V(\bar{F})$ and such that $c(F) \leq d(\bar{F})$. In fact, F can be build from an empty graph on $V(G)$ by iteratively adding – according to our definition of graph addition – a shortest path between u and v in G, for every edge $(u, v) \in E(\bar{F})$.

We also observe that any forest F of K can be viewed as a forest of K_n. Conversely, any forest F of K_n can be transformed in polynomial time into a forest of K spanning $V(F)$ and having the same cost of F: it suffices to identify each of the 0-cost edges of F between any vertex and any of its copies. Therefore, we have polynomial-time algorithms that convert any forest F of any graph in $\{G, K, K_n\}$ into a forest F' of any graph in $\{G, K, K_n\}$ such that F' always spans $V(F)$ and the cost of F' (according to the cost function associated with the corresponding graph) is less than or equal to the cost of F (according to the cost function associated with the corresponding graph). All these polynomial-time algorithms will be implicitly used in the rest of the paper; for example, we can say that a Steiner tree of I_n is also a Steiner tree of I or that a forest of I_n that spans v is also a forest of I that spans v. However, we observe that some of these polynomial-time algorithms do no preserve useful structural properties; for example, when transforming a k-restricted Steiner tree of I_n into a Steiner tree of I we may lose the property that the resulting tree is k-restricted. Therefore, whenever we refer to a structural property of a forest, we assume that such a property holds only w.r.t. the underlying graph the forest belongs to.

3 General tools

In this section we describe the three main tools that are used by all our algorithms.

The first tool is the algorithm Connect that has been introduced by Böckenhauer et al. in [14]. This algorithm takes an *STP* instance I together with a Steiner forest F of I as inputs and computes a minimum-cost set of edges of G whose addition to F yields a Steiner tree of I. The algorithm Connect reduces the *STP* instance to a smaller *STP* instance, where each terminal vertex corresponds to a tree of F, and then uses the famous Dreyfus-Wagner algorithm (see [28]) to find an optimal solution of the reduced instance.[3] If F contains q trees, the running time of the algorithm Connect is $O^*(3^q)$;[4] this implies that the algorithm has polynomial running time when q is constant. The call of the algorithm Connect with input parameters I and F is denoted by Connect(I, F).

The second tool is the algorithmic proof of Borchers and Du on k-restricted Steiner trees [20] that has already been used in Steiner tree reoptimization in the recent paper of Goyal and Mömke [30]. In their paper, Borchers and Du proved that, for every *STP* instance I and every $\xi > 0$, there exists a k-restricted Steiner tree S_ξ of I_n, with $k = 2^{\lceil 1/\xi \rceil}$, such that $d(S_\xi) \leq (1+\xi)c(OPT)$. As the following theorem shows, the algorithmic proof of Borchers and Du immediately extends to any (not necessarily optimal!) Steiner tree of I_n.

▶ **Theorem 1** ([20]). *Let I be an STP instance, S a Steiner tree of I, and $\xi > 0$ a real value. There is a polynomial time algorithm that computes a k-restricted Steiner tree S_ξ of I_n, with $k = 2^{\lceil 1/\xi \rceil}$, such that $d(S_\xi) \leq (1+\xi)c(S)$.*

The call of Borchers and Du algorithm with input parameters I, S, and ξ, is denoted by RestrictedST(I, S, ξ). By construction, the algorithm of Borchers and Du also guarantees the following properties:

(a) if v is a Steiner vertex that has degree at least 3 in S, then S_ξ spans v;

(b) if the degree of a terminal t in S is 2, then the degree of t in S_ξ is at most 4.

As shown in the following corollary, these two properties are useful if we want that a specific vertex of S would also be a vertex of the k-restricted Steiner tree S_ξ of I_n.

[3] We refer to Hougardy [31] for further exact algorithms for the *STP*.

[4] The O^* notation hides polynomial factors.

Algorithm 1: The pseudocode of the algorithm BUILDST(I, F, h).

```
1 if F contains less than h full components then return CONNECT(I, (R, ∅));
2 S' ←⊥;
3 for every set H of up to h full components of F do
4 |   F̄ ← F − H;
5 |   S̄ ← CONNECT(I_n, F̄);
6 |   if S' =⊥ or c(S̄) < c(S') then S' ← S̄;
7 return S';
```

▶ **Corollary 2.** *Let I be an STP instance, S a Steiner tree of I, v a vertex of S, and $\xi > 0$ a real value. There is a polynomial time algorithm that computes a k-restricted Steiner tree S_ξ of I_n, with $k = 2^{2+\lceil 1/\xi \rceil}$, such that $d(S_\xi) \leq (1+\xi)c(S)$ and $v \in V(S_\xi)$.*

Proof. The cases in which $v \in R$ has already been proved in Theorem 1. Moreover, the claim trivially holds by property (a) when v is a Steiner vertex of degree greater than or equal to 3 in S. Therefore we assume that v is a Steiner vertex of degree 2 in S. Let $I' = \langle G, c, R \cup \{v\} \rangle$. In this case, the call of RESTRICTEDST(I', S, ξ) outputs a k'-restricted Steiner tree S_ξ of I'_n, with $k' = 2^{\lceil 1/\xi \rceil}$, such that $d(S_\xi) \leq (1+\xi)c(S)$. Since v is a terminal in I', by property (b) the degree of v in S_ξ is at most 4; in other words, v is contained in at most 4 full components of S_ξ. Therefore, S_ξ is a k-restricted Steiner tree of I, with $k = 4k' = 2^{2+\lceil 1/\xi \rceil}$. ◀

The call of Borchers and Du algorithm with parameters I, S, ξ, and v is denoted by RESTRICTEDST(I, S, ξ, v).

The third tool, which is the novel idea of this paper, is the algorithm BUILDST (see Algorithm 1 for the pseudocode). The algorithm BUILDST takes as inputs an *STP* instance I, a Steiner forest F of I_n, and a positive integer value h. The algorithm computes a Steiner tree of I by selecting, among several feasible solutions, the one of minimum cost. More precisely, the algorithm computes a feasible solution for each forest $\bar{F}$ that is obtained by removing from F up to h of its full components. The Steiner tree of I associated with $\bar{F}$ is obtained by the call of CONNECT$(I_n, \bar{F})$. If F contains a number of full components which is strictly less than h, then the algorithm computes a minimum-cost Steiner tree of I from scratch. The call of the algorithm BUILDST with input parameters I, F, and h is denoted by BUILDST(I, F, h). The proof of the following lemma is immediate.

▶ **Lemma 3.** *Let I be an STP instance, F a Steiner forest of I_n, $\bar{F}$ a Steiner forest of I_n that is obtained from F by removing up to h of its full components, and H a minimum-cost subgraph of K_n such that $\bar{F} + H$ is a Steiner tree of I. Then the cost of the Steiner tree of I returned by the call of* BUILDST(I, F, h) *is at most $c(\bar{F}) + c(H)$.*

Moreover, the following lemmas hold.

▶ **Lemma 4.** *Let I be an STP instance and F a forest of G spanning R. If F contains q trees, then every minimal (w.r.t. the deletion of entire full components) subgraph H of any graph in $\{G, K, K_n\}$ such that $F + H$ is a Steiner tree of I contains at most $q - 1$ full components.*

Proof. Since H is minimal, each full component of H contains at least 2 vertices each of which respectively belongs to one of two distinct trees of F. Therefore, if we add the full components of H to F one after the other, the number of connected components of the resulting graph strictly decreases at each iteration because of the minimality of H. Hence, H contains at most $q - 1$ full components. ◀

Algorithm 2: A Steiner vertex $t \in V(G) \setminus R$ becomes a terminal.

```
1 ξ ← ε/10;
2 h ← 2^{2⌈2/ε⌉⌈1/ξ⌉};
3 S_ξ ← RestrictedST(I, S, ξ);
4 S' ← BuildST(I', S_ξ + t, h);
5 return S';
```

▶ **Lemma 5.** *Let I be an STP instance and F a Steiner forest of I_n of q trees. If each tree of F is k-restricted, then the call of* BuildST(I, F, h) *outputs a Steiner tree of I in time $O^*(|R|^h 3^{q+hk})$.*

Proof. If F contains at most h full components, then the number of terminals is at most $q + hk$ and the call of Connect$(I, (R, \emptyset))$ outputs a Steiner tree of I in time $O^*(3^{q+hk})$. Therefore, we assume that F contains a number of full components that is greater than or equal to h. In this case, the algorithm BuildST(I, F, h) computes a feasible solution for each forest that is obtained by removing up to h full components of F from itself. As the number of full components of F is at most $|R|$, the overall number of feasible solutions evaluated by the algorithm is $O(|R|^{h+1})$. Let $\bar{F}$ be any forest that is obtained from F by removing h of its full components. Since F is k-restricted, $\bar{F}$ contains at most $q + hk$ trees. Therefore the call of Connect$(I_n, \bar{F})$ requires $O^*(3^{q+hk})$ time to output a solution. Hence, the overall time needed by the call of BuildST(I, F, h) to output a solution is $O^*(|R|^h 3^{q+hk})$. ◀

For the rest of the paper, we denote by $\langle I, S, I' \rangle$ an instance of the Steiner tree reoptimization problem. Furthermore, we also assume that $\rho \leq 2$ as well as $\epsilon \leq 1$, as otherwise we can use classical time-efficient algorithms to compute a 2-approximate solution of I'.

4 A Steiner vertex becomes a terminal

In this section we consider the local modification in which a Steiner vertex $t \in V(G) \setminus R$ becomes a terminal. Clearly, $G' = G$, $c' = c$, $d' = d$, whereas $R' = R \cup \{t\}$. Therefore, for the sake of readability, we will drop the superscripts from G', c', and d'.

Let $\xi = \epsilon/10$ and $h = 2^{2\lceil 2/\epsilon \rceil \lceil 1/\xi \rceil}$. The algorithm, whose pseudocode is reported in Algorithm 2, computes a Steiner tree of I' first by calling RestrictedST(I, S, ξ) to obtain a k-restricted Steiner tree S_ξ of I_n, with $k = 2^{\lceil 1/\xi \rceil}$, and then by calling BuildST$(I', S_\xi + t, h)$. Intuitively, the algorithm *guesses* the full components of S_ξ to be replaced with the corresponding full components of a suitable k'-restricted version of a new (fixed) optimal solution, for suitable values of k'.

▶ **Theorem 6.** *Let $\langle I, S, I' \rangle$ be an instance of Steiner tree reoptimization, where S is a ρ-approximate solution of I and I' is obtained from I by changing the status of a single vertex from Steiner to terminal. Then Algorithm 2 computes a $(\rho + \epsilon)$-approximate Steiner tree of I' in polynomial time.*

Proof. Theorem 1 implies that the computation of S_ξ requires polynomial time. Since S_ξ is a Steiner tree of I_n, $S_\xi + t$ is a Steiner forest of I'_n of at most two trees (observe that S_ξ may already contain t). Therefore, using Lemma 5 and the fact that h is a constant value, the call of BuildST$(I', S_\xi + t, h)$ outputs a solution S' in polynomial time. Hence, the overall time complexity of Algorithm 2 is polynomial.

In the following we show that Algorithm 2 returns a $(\rho + \epsilon)$-approximate solution. Let S'_ξ denote the Steiner tree of I'_n that is returned by the call of RestrictedST(I', OPT', ξ). Let

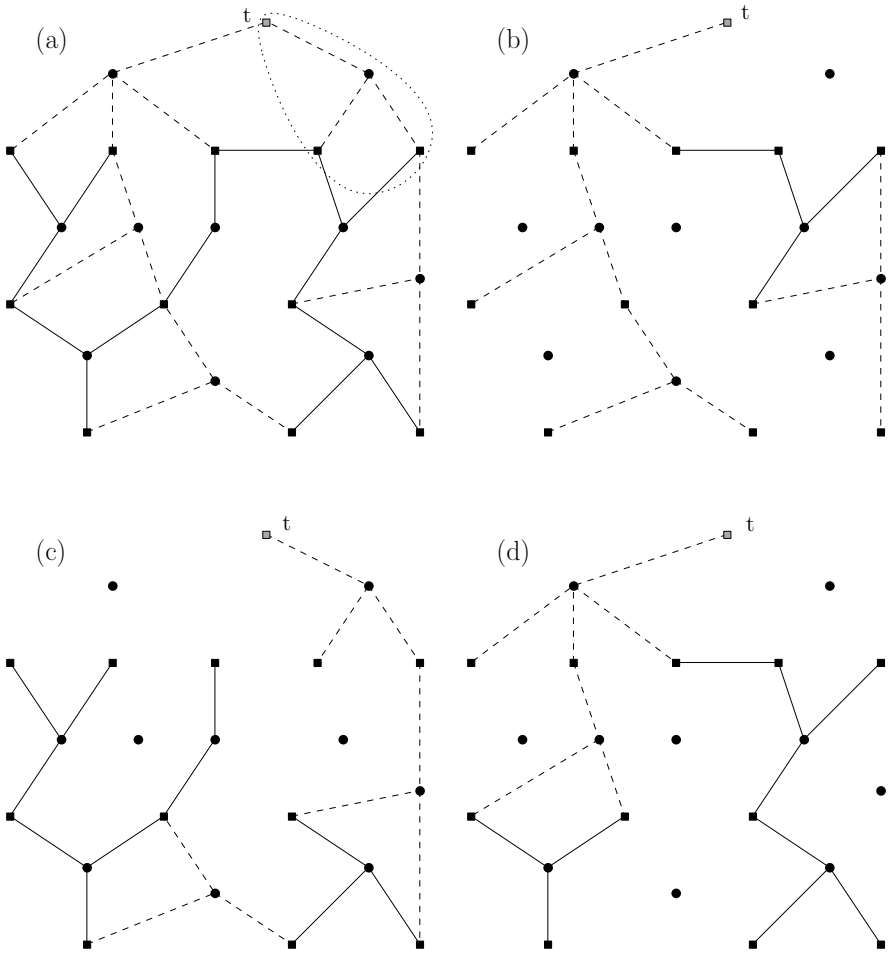


Figure 1 An example that shows how H_i and H'_i are defined. In (a) it is depicted a Steiner tree reoptimization instance where square vertices represent the terminals, while circle vertices represent the Steiner vertices. The Steiner vertex t of I that becomes a terminal in I' is represented as a terminal vertex of grey color. Solid edges are the edges of S_ξ, while dashed edges are the edges of S'_ξ. The full component H'_0 of S'_ξ is highlighted in (a). The forests $S'_\xi - H'_0$ and H_1 are shown in (b). The forests $S_\xi - H_1$ and H'_1 are shown in (c). Finally, the forests $S'_\xi - H'_1$ and H_2 are shown in (d).

H'_0 be a full component of S'_ξ that spans t (ties are broken arbitrarily). For every $i = 1, \ldots, \ell$, with $\ell = \lceil 2/\epsilon \rceil$, we define:

- H_i as the forest consisting of a minimal set of full components of S_ξ whose addition to $S'_\xi - H'_{i-1}$ yields a Steiner tree of I_n (see Figure 1 (b) and (d));
- H'_i as the forest consisting of a minimal set of full components of S'_ξ whose addition to $S_\xi - H_i$ yields a Steiner tree of I'_n (see Figure 1 (c)).

For the rest of the proof, we denote by S_i the Steiner tree of I' yielded by the addition of H'_i to $S_\xi - H_i$.

Let $k = 2^{\lceil 1/\xi \rceil}$ and observe that both S_ξ and S'_ξ are k-restricted Steiner trees of I_n and I'_n, respectively (see Theorem 1). Therefore H'_0 spans at most k terminals. Furthermore, by repeatedly using Lemma 4, H_i contains at most k^{2i-1} full components and spans at most k^{2i} terminals, while H'_i contains at most k^{2i} full components and spans at most k^{2i+1} terminals.

This implies that the number of the full components of H_i, with $i = 1, \ldots, \ell$, is at most

$$k^{2i-1} \leq k^{2\ell} = 2^{2\lceil 2/\epsilon \rceil \lceil 1/\xi \rceil} = h.$$

Therefore, by Lemma 3, the cost of the solution returned by the call of $\textsc{BuildST}(I', S_\xi + t, h)$ is at most $c(S_i)$. As a consequence

$$c(S') \leq \frac{1}{\ell} \sum_{i=1}^{\ell} c(S_i). \tag{1}$$

Now we prove an upper bound to the cost of each S_i. Let $\Delta_{OPT} = (1+\xi)c(OPT') - c(OPT)$. As any Steiner tree of I' is also a Steiner tree of I, $c(OPT') \geq c(OPT)$; therefore $\Delta_{OPT} \geq 0$. Since the addition of H_i to $S'_\xi - H'_{i-1}$ yields a Steiner tree of I, the cost of this tree is lower bounded by the cost of OPT. As a consequence, using Theorem 1 in the second inequality that follows, $c(OPT) \leq d(S'_\xi) - d(H'_{i-1}) + d(H_i) \leq (1+\xi)c(OPT') - d(H'_{i-1}) + d(H_i)$, i.e.,

$$d(H_i) \geq d(H'_{i-1}) - \Delta_{OPT}. \tag{2}$$

Using Theorem 1 in the first inequality that follows we have that

$$d(S_\xi) \leq (1+\xi)c(S) \leq \rho(1+\xi)c(OPT) \leq \rho(1+\xi)^2 c(OPT') - \Delta_{OPT}. \tag{3}$$

Using both (2) and (3) in the second inequality that follows, we can upper bound the cost of S_i with

$$c(S_i) \leq d(S_\xi) - d(H_i) + d(H'_i) \leq \rho(1+\xi)^2 c(OPT') - d(H'_{i-1}) + d(H'_i). \tag{4}$$

Observe that the overall sum of the last two terms on the right-hand side of (4) for every $i \in \{1, \ldots, \ell\}$ is a telescoping sum. As a consequence, if for every $i \in \{1, \ldots, \ell\}$ we substitute the term $c(S_i)$ in (1) with the corresponding upper bound in (4), and use Theorem 1 to derive that $d(H'_\ell) \leq d(S'_\xi) \leq (1+\xi)c(OPT')$, we obtain

$$c(S') \leq \frac{1}{\ell} \sum_{i=1}^{\ell} c(S_i) \leq \rho(1+\xi)^2 c(OPT') + \frac{1+\xi}{\ell} c(OPT') \leq (\rho + \epsilon) c(OPT'),$$

where last inequality holds by the choice of ξ and ℓ, and because $\rho \leq 2$ and $\epsilon \leq 1$. This completes the proof. ◀

5 The cost of an edge increases

In this section we consider the local modification in which the cost of an edge $e = (u, v) \in E(G)$ increases by $\Delta > 0$. More formally, we have that $G' = G$, $R' = R$, whereas, for every $e' \in E(G)$,

$$c'(e') = \begin{cases} c(e) + \Delta & \text{if } e' = e; \\ c(e') & \text{otherwise.} \end{cases}$$

For the sake of readability, in this section we drop the superscripts from G' and R'. Furthermore, we denote by c_{-e} the edge-cost function c restricted to $G - e$ and by d_{-e} the corresponding shortest-path metric.

Algorithm 3: The cost of an edge $e = (u, v)$ increases by $\Delta > 0$.

$\xi \leftarrow \epsilon/10$;
$h \leftarrow 2^{2(1+\lceil 1/\xi \rceil)\lceil 2/\epsilon \rceil}$;
if $e \notin E(S)$ **then**
  $S' \leftarrow S$;
else
  let S_u and S_v be the tree of $S - e$ containing u and v, respectively;
  $I_u \leftarrow \langle G - e, c_{-e}, V(S_u) \cap R \rangle$;
  $I_v \leftarrow \langle G - e, c_{-e}, V(S_v) \cap R \rangle$;
  $S_{u,\xi} \leftarrow \textsc{RestrictedST}(I_u, S_u, \xi, u)$;
  $S_{v,\xi} \leftarrow \textsc{RestrictedST}(I_v, S_v, \xi, v)$;
  $S_\xi \leftarrow S_{u,\xi} + S_{v,\xi}$;
  $S' \leftarrow \textsc{BuildST}(I', S_\xi, h)$;
  if $c'(S) \leq c'(S')$ **then** $S' \leftarrow S$;
return S';

Let $\xi = \epsilon/10$ and $h = 2^{2(1+\lceil 1/\xi \rceil)\lceil 2/\epsilon \rceil}$. The algorithm, whose pseudocode is reported in Algorithm 3, first checks whether $e \in E(S)$. If this is not the case, then the algorithm returns S. If $e \in E(S)$, then let S_u and S_v denote the two trees of $S - e$ containing u and v, respectively. The algorithm first computes a k-restricted Steiner tree $S_{u,\xi}$ (resp., $S_{v,\xi}$), with $k = 2^{2+\lceil 1/\xi \rceil}$, of S_u (resp., S_v) such that $u \in V(S_{u,\xi})$ (resp., $v \in V(S_{v,\xi})$). Then the algorithm computes a solution S' via the call of $\textsc{BuildST}(I', S_{u,\xi} + S_{v,\xi}, h)$, and, finally, it returns the cheapest solution between S and S'. As we will see, the removal of e from S is necessary to guarantee that the cost of the processed Steiner forest of I' (i.e., $S - e$) is upper bounded by the cost of OPT'.

▶ **Theorem 7.** *Let $\langle I, S, I' \rangle$ be an instance of Steiner tree reoptimization, where S is a ρ-approximate solution of I and I' is obtained from I by increasing the cost of a single edge. Then Algorithm 3 computes a $(\rho + \epsilon)$-approximate Steiner tree of I' in polynomial time.*

Proof. Corollary 2 implies that the computation of both $S_{u,\xi}$ and $S_{v,\xi}$ requires polynomial time. Therefore, using Lemma 5 and the fact that h is a constant value, the call of $\textsc{BuildST}(I', S_\xi, h)$ outputs a solution S' in polynomial time. Hence, the overall time complexity of Algorithm 3 is polynomial.

In the following we show that Algorithm 3 returns a $(\rho+\epsilon)$-approximate solution. Observe that the local modification does not change the set of feasible solutions, i.e., a tree is a Steiner tree of I iff it is a Steiner tree of I'. This implies that OPT' is a Steiner tree of I as well. Thanks to this observation, if $e \notin E(S)$ we have that

$$c'(S) = c(S) \leq \rho c(OPT) \leq \rho c(OPT') \leq \rho c'(OPT'),$$

i.e., the solution returned by the algorithm is a ρ-approximate solution. Moreover, if $e \in E(OPT')$, then $c'(OPT') = c(OPT') + \Delta$ or, equivalently, $c(OPT') = c'(OPT') - \Delta$. As a consequence,

$$c'(S) \leq c(S) + \Delta \leq \rho c(OPT) + \Delta \leq \rho c(OPT') + \Delta \leq \rho c'(OPT'),$$

i.e., once again, the solution returned by the algorithm is a ρ-approximate solution. Therefore, in the rest of the proof, we assume that $e \in E(S)$ as well as $e \notin E(OPT')$.

Let S'_ξ denote the Steiner tree of $\langle K_n, d_{-e}, R \rangle$ that is returned by the call of $\textsc{RestrictedST}(\langle G - e, c_{-e}, R \rangle, OPT', \xi)$. Using Theorem 1 and the fact that $e \notin E(OPT')$ we have that

$$d_{-e}(S'_\xi) \leq (1 + \xi) c_{-e}(OPT') = (1 + \xi) c'(OPT'). \tag{5}$$

Let H'_0 be a full component of S'_ξ whose addition to S_ξ yields a Steiner tree of I'_n (ties are broken arbitrarily). For every $i = 1, \ldots, \ell$, with $\ell = \lceil 2/\epsilon \rceil$, we define:

- H_i as the forest consisting of a minimal set of full components of S_ξ such that the addition of H_i and e to $S'_\xi - H'_{i-1}$ yields a Steiner tree of I_n;[5]
- H'_i as the forest consisting of a minimal set of full components of S'_ξ whose addition to $S_\xi - H_i$ yields a Steiner tree of I'_n.

For the rest of the proof, we denote by S_i the Steiner tree of I' obtained by augmenting $S_\xi - H_i$ with H'_i. Observe that S_i does not contain e. Therefore

$$c'(S_i) \leq c_{-e}(S_i) \leq d_{-e}(S_\xi) - d_{-e}(H_i) + d_{-e}(H'_i). \tag{6}$$

Let $k = 2^{2+\lceil 1/\xi \rceil}$ and $r = 2^{\lceil 1/\xi \rceil}$. Observe that S_ξ is a k-restricted Steiner forest of I_n (see Corollary 2), while S'_ξ is an r-restricted Steiner tree of I'_n (see Theorem 1). Therefore, H'_0 spans at most r terminals. Moreover, by repeatedly using Lemma 4, H_i contains at most $k^{i-1}r^i$ full components and spans at most $k^i r^i$ terminals, while H'_i contains at most $k^i r^i$ full components and spans at most $k^i r^{i+1}$ terminals. This implies that the number of full components of H_i, with $i = 1, \ldots, \ell$, is at most

$$k^{i-1}r^i \leq k^\ell r^\ell = 2^{2(1+\lceil 1/\xi \rceil)\lceil 2/\epsilon \rceil} = h.$$

Therefore, by Lemma 3, the cost of the solution returned by the call of BuildST(I', S_ξ, h) is at most the cost of S_i. As a consequence

$$c'(S') \leq \frac{1}{\ell} \sum_{i=1}^{\ell} c'(S_i). \tag{7}$$

Now we prove an upper bound to the cost of each S_i, with $i \in \{1, \ldots, \ell\}$. Let $\Delta_{OPT} = (1+\xi)c'(OPT') - c(OPT)$ and observe that $\Delta_{OPT} \geq 0$. Since the addition of H_i and e to $S'_\xi - H'_{i-1}$ yields a Steiner tree of I, the cost of this tree is lower bounded by the cost of OPT. As a consequence, using (5) together with Theorem 1 in the third inequality that follows,

$$\begin{aligned} c(OPT) &\leq d(S'_\xi - H'_{i-1}) + d(H_i) + d(e) \leq d_{-e}(S'_\xi - H'_{i-1}) + d_{-e}(H_i) + c(e) \\ &= d_{-e}(S'_\xi) - d_{-e}(H'_{i-1}) + d_{-e}(H_i) + c(e) \\ &\leq (1+\xi)c'(OPT') - d_{-e}(H'_{i-1}) + d_{-e}(H_i) + c(e), \end{aligned}$$

from which we derive

$$d_{-e}(H_i) \geq d_{-e}(H'_{i-1}) - \Delta_{OPT} - c(e). \tag{8}$$

Using Corollary 2 twice in the first inequality that follows we have that

$$\begin{aligned} d_{-e}(S_\xi) &= d_{-e}(S_{u,\xi}) + d_{-e}(S_{v,\xi}) \leq (1+\xi)c_{-e}(S_u) + (1+\xi)c_{-e}(S_v) \\ &= (1+\xi)c(S) - (1+\xi)c(e) \leq \rho(1+\xi)c(OPT) - c(e) \\ &\leq \rho(1+\xi)^2 c'(OPT') - \Delta_{OPT} - c(e). \end{aligned} \tag{9}$$

Starting from (6) and using both (8) and (9) in the second inequality that follows, we can upper bound the cost of S_i, for every $i \in \{1, \ldots, \ell\}$, with

$$c'(S_i) \leq d_{-e}(S_\xi) - d_{-e}(H_i) + d_{-e}(H'_i) \leq \rho(1+\xi)^2 c'(OPT') - d_{-e}(H'_{i-1}) + d_{-e}(H'_i). \tag{10}$$

[5] We observe that the existence of H_i is guarantee by the fact that $S_\xi + e$ is a Steiner tree of I_n: in fact, $S_\xi = S_{u,\xi} + S_{v,\xi}$ is a forest of two trees such that $u \in S_{u,\xi}$ and $v \in S_{v,\xi}$.

As a consequence, if for every $i \in \{1, \ldots, \ell\}$ we substitute the term $c'(S_i)$ in (7) with the corresponding upper bound in (10), and use (5) to derive $d_{-e}(H'_\ell) \leq d_{-e}(S'_\xi) \leq (1 + \xi)c'(OPT')$, we obtain

$$c'(S') \leq \frac{1}{\ell} \sum_{i=1}^{\ell} c'(S_i) \leq \rho(1+\xi)^2 c'(OPT') + \frac{1+\xi}{\ell} c'(OPT') \leq (\rho + \epsilon)c'(OPT'),$$

where last inequality holds by the choice of ξ and ℓ, and because $\rho \leq 2$ and $\epsilon \leq 1$. This completes the proof. ◀

References

1 Claudia Archetti, Luca Bertazzi, and Maria Grazia Speranza. Reoptimizing the traveling salesman problem. *Networks*, 42(3):154–159, 2003. doi:10.1002/net.10091.

2 Claudia Archetti, Luca Bertazzi, and Maria Grazia Speranza. Reoptimizing the 0-1 knapsack problem. *Discrete Applied Mathematics*, 158(17):1879–1887, 2010. doi:10.1016/j.dam.2010.08.003.

3 Claudia Archetti, Gianfranco Guastaroba, and Maria Grazia Speranza. Reoptimizing the rural postman problem. *Computers & OR*, 40(5):1306–1313, 2013. doi:10.1016/j.cor.2012.12.010.

4 Giorgio Ausiello, Vincenzo Bonifaci, and Bruno Escoffier. *Complexity and Approximation in Reoptimization*, pages 101–129. Imperial College Press, 2012.

5 Giorgio Ausiello, Bruno Escoffier, Jérôme Monnot, and Vangelis Th. Paschos. Reoptimization of minimum and maximum traveling salesman's tours. *J. Discrete Algorithms*, 7(4):453–463, 2009. doi:10.1016/j.jda.2008.12.001.

6 Michael A. Bender, Martin Farach-Colton, Sándor P. Fekete, Jeremy T. Fineman, and Seth Gilbert. Reallocation problems in scheduling. *Algorithmica*, 73(2):389–409, 2015. doi:10.1007/s00453-014-9930-4.

7 Tobias Berg and Harald Hempel. Reoptimization of traveling salesperson problems: Changing single edge-weights. In Adrian-Horia Dediu, Armand-Mihai Ionescu, and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications, Third International Conference, LATA 2009, Tarragona, Spain, April 2-8, 2009. Proceedings*, volume 5457 of *Lecture Notes in Computer Science*, pages 141–151. Springer, 2009. doi:10.1007/978-3-642-00982-2_12.

8 Davide Bilò, Hans-Joachim Böckenhauer, Juraj Hromkovic, Richard Královic, Tobias Mömke, Peter Widmayer, and Anna Zych. Reoptimization of Steiner trees. In Joachim Gudmundsson, editor, *Algorithm Theory - SWAT 2008, 11th Scandinavian Workshop on Algorithm Theory, Gothenburg, Sweden, July 2-4, 2008, Proceedings*, volume 5124 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2008. doi:10.1007/978-3-540-69903-3_24.

9 Davide Bilò, Hans-Joachim Böckenhauer, Dennis Komm, Richard Královic, Tobias Mömke, Sebastian Seibert, and Anna Zych. Reoptimization of the shortest common superstring problem. *Algorithmica*, 61(2):227–251, 2011. doi:10.1007/s00453-010-9419-8.

10 Davide Bilò, Peter Widmayer, and Anna Zych. Reoptimization of weighted graph and covering problems. In Evripidis Bampis and Martin Skutella, editors, *Approximation and Online Algorithms, 6th International Workshop, WAOA 2008, Karlsruhe, Germany, September 18-19, 2008. Revised Papers*, volume 5426 of *Lecture Notes in Computer Science*, pages 201–213. Springer, 2008. doi:10.1007/978-3-540-93980-1_16.

11 Davide Bilò and Anna Zych. New advances in reoptimizing the minimum Steiner tree problem. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012,*

Bratislava, Slovakia, August 27-31, 2012. Proceedings, volume 7464 of *Lecture Notes in Computer Science*, pages 184–197. Springer, 2012. doi:10.1007/978-3-642-32589-2_19.

12 Hans-Joachim Böckenhauer, Luca Forlizzi, Juraj Hromkovic, Joachim Kneis, Joachim Kupke, Guido Proietti, and Peter Widmayer. On the approximability of TSP on local modifications of optimally solved instances. *Algorithmic Operations Research*, 2(2):83–93, 2007. URL: http://journals.hil.unb.ca/index.php/AOR/article/view/2803.

13 Hans-Joachim Böckenhauer, Karin Freiermuth, Juraj Hromkovic, Tobias Mömke, Andreas Sprock, and Björn Steffen. Steiner tree reoptimization in graphs with sharpened triangle inequality. *J. Discrete Algorithms*, 11:73–86, 2012. doi:10.1016/j.jda.2011.03.014.

14 Hans-Joachim Böckenhauer, Juraj Hromkovic, Richard Královic, Tobias Mömke, and Peter Rossmanith. Reoptimization of Steiner trees: Changing the terminal set. *Theor. Comput. Sci.*, 410(36):3428–3435, 2009. doi:10.1016/j.tcs.2008.04.039.

15 Hans-Joachim Böckenhauer, Juraj Hromkovic, Tobias Mömke, and Peter Widmayer. On the hardness of reoptimization. In Viliam Geffert, Juhani Karhumäki, Alberto Bertoni, Bart Preneel, Pavol Návrat, and Mária Bieliková, editors, *SOFSEM 2008: Theory and Practice of Computer Science, 34th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 19-25, 2008, Proceedings*, volume 4910 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2008. doi:10.1007/978-3-540-77566-9_5.

16 Hans-Joachim Böckenhauer, Juraj Hromkovic, and Andreas Sprock. Knowing all optimal solutions does not help for TSP reoptimization. In Jozef Kelemen and Alica Kelemenová, editors, *Computation, Cooperation, and Life - Essays Dedicated to Gheorghe Paun on the Occasion of His 60th Birthday*, volume 6610 of *Lecture Notes in Computer Science*, pages 7–15. Springer, 2011. doi:10.1007/978-3-642-20000-7_2.

17 Hans-Joachim Böckenhauer, Juraj Hromkovic, and Andreas Sprock. On the hardness of reoptimization with multiple given solutions. *Fundam. Inform.*, 110(1-4):59–76, 2011. doi:10.3233/FI-2011-528.

18 Hans-Joachim Böckenhauer, Joachim Kneis, and Joachim Kupke. Approximation hardness of deadline-TSP reoptimization. *Theor. Comput. Sci.*, 410(21-23):2241–2249, 2009. doi:10.1016/j.tcs.2009.02.016.

19 Hans-Joachim Böckenhauer and Dennis Komm. Reoptimization of the metric deadline TSP. *J. Discrete Algorithms*, 8(1):87–100, 2010. doi:10.1016/j.jda.2009.04.001.

20 Al Borchers and Ding-Zhu Du. The k-Steiner ratio in graphs. *SIAM J. Comput.*, 26(3):857–869, 1997. doi:10.1137/S0097539795281086.

21 Nicolas Boria and Federico Della Croce. Reoptimization in machine scheduling. *Theor. Comput. Sci.*, 540:13–26, 2014. doi:10.1016/j.tcs.2014.04.004.

22 Nicolas Boria, Jérôme Monnot, and Vangelis Th. Paschos. Reoptimization of maximum weight induced hereditary subgraph problems. *Theor. Comput. Sci.*, 514:61–74, 2013. doi:10.1016/j.tcs.2012.10.037.

23 Nicolas Boria, Jérôme Monnot, and Vangelis Th. Paschos. Reoptimization under vertex insertion: Max p_k-free subgraph and max planar subgraph. *Discrete Math., Alg. and Appl.*, 5(2), 2013. doi:10.1142/S1793830913600045.

24 Nicolas Boria and Vangelis Th. Paschos. Fast reoptimization for the minimum spanning tree problem. *J. Discrete Algorithms*, 8(3):296–310, 2010. doi:10.1016/j.jda.2009.07.002.

25 Nicolas Boria and Vangelis Th. Paschos. A survey on combinatorial optimization in dynamic environments. *RAIRO - Operations Research*, 45(3):241–294, 2011. doi:10.1051/ro/2011114.

26 Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33, 2013. doi:10.1145/2432622.2432628.

27 Wenkai Dai. Reoptimization of minimum latency problem. In Yixin Cao and Jianer Chen, editors, *Computing and Combinatorics - 23rd International Conference, COCOON 2017, Hong Kong, China, August 3-5, 2017, Proceedings*, volume 10392 of *Lecture Notes in Computer Science*, pages 162–174. Springer, 2017. doi:10.1007/978-3-319-62389-4_14.

28 S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971. doi:10.1002/net.3230010302.

29 Bruno Escoffier, Martin Milanic, and Vangelis Th. Paschos. Simple and fast reoptimizations for the Steiner tree problem. *Algorithmic Operations Research*, 4(2):86–94, 2009. URL: http://journals.hil.unb.ca/index.php/AOR/article/view/5653.

30 Keshav Goyal and Tobias Mömke. Robust reoptimization of Steiner trees. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, volume 45 of *LIPIcs*, pages 10–24. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.10.

31 Stefan Hougardy, Jannik Silvanus, and Jens Vygen. Dijkstra meets Steiner: a fast exact goal-oriented Steiner tree algorithm. *Math. Program. Comput.*, 9(2):135–202, 2017. doi:10.1007/s12532-016-0110-1.

32 Jérôme Monnot. A note on the traveling salesman reoptimization problem under vertex insertion. *Inf. Process. Lett.*, 115(3):435–438, 2015. doi:10.1016/j.ipl.2014.11.003.

33 Markus W. Schäffter. Scheduling with forbidden sets. *Discrete Applied Mathematics*, 72(1-2):155–166, 1997. doi:10.1016/S0166-218X(96)00042-X.

34 Anna Zych. *Reoptimization of NP-hard problems.* PhD thesis, Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 20257, 2012.

35 Anna Zych and Davide Bilò. New reoptimization techniques applied to Steiner tree problem. *Electronic Notes in Discrete Mathematics*, 37:387–392, 2011. doi:10.1016/j.endm.2011.05.066.

Efficient Shortest Paths in Scale-Free Networks with Underlying Hyperbolic Geometry

Thomas Bläsius
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
thomas.blaesius@hpi.de

Cedric Freiberger
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
cedric.freiberger@student.hpi.de

Tobias Friedrich
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
tobias.friedrich@hpi.de

Maximilian Katzmann
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
maximilian.katzmann@hpi.de

Felix Montenegro-Retana
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
felix.montenegro-retana@student.hpi.de

Marianne Thieffry
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
marianne.thieffry@student.hpi.de

Abstract

A common way to accelerate shortest path algorithms on graphs is the use of a bidirectional search, which simultaneously explores the graph from the start and the destination. It has been observed recently that this strategy performs particularly well on scale-free real-world networks. Such networks typically have a heterogeneous degree distribution (e.g., a power-law distribution) and high clustering (i.e., vertices with a common neighbor are likely to be connected themselves). These two properties can be obtained by assuming an underlying hyperbolic geometry.

To explain the observed behavior of the bidirectional search, we analyze its running time on hyperbolic random graphs and prove that it is $\tilde{\mathcal{O}}(n^{2-1/\alpha} + n^{1/(2\alpha)} + \delta_{\max})$ with high probability, where $\alpha \in (0.5, 1)$ controls the power-law exponent of the degree distribution, and $\delta_{\max}$ is the maximum degree. This bound is sublinear, improving the obvious worst-case linear bound. Although our analysis depends on the underlying geometry, the algorithm itself is oblivious to it.

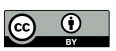
2012 ACM Subject Classification Theory of computation → Shortest paths, Theory of computation → Random network models, Theory of computation → Computational geometry

Keywords and phrases random graphs, hyperbolic geometry, scale-free networks, bidirectional shortest path

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.20

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.03253`.

Acknowledgements This research has received funding from the German Research Foundation (DFG) under grant agreement no. FR 2988 (ADLON, HYP).



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 20; pp. 20:1–20:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Finding shortest paths between nodes in a network is among the most basic graph problems. Besides being of independent interest, many algorithms use shortest path queries as a subroutine. On unweighted graphs, such queries can be answered in linear time using a *breadth-first search (BFS)*. Though this is optimal in the worst case, it is not efficient enough when dealing with large networks or problems involving many shortest path queries.

A way to heuristically improve the run time, is to use a *bidirectional BFS* [16]. It runs two searches, simultaneously exploring the graph from the start and the destination. The shortest path is then found once the two search spaces touch. Though this heuristic does not improve the worst-case running time, recent experiments by Borassi and Natale [6] suggest that it achieves a significant speedup on scale-free real-world networks. They also try to explain the observed run times by proving that the bidirectional BFS runs in sublinear expected time on different random graph models. Though this is a great result, we do not think that it provides a satisfying explanation for the good practical performance for two reasons.

First, the bidirectional search performs particularly well on networks with a heterogeneous degree distribution (i.e., few vertices with high degree, many vertices with low degree). A common assumption is that the degree distribution follows a *power-law*, i.e., the number of vertices of degree k is proportional to $k^{-\beta}$. The constant β is called the *power-law exponent* and is typically between 2 and 3. The above mentioned proof predicts a shorter execution time for homogeneous graphs (e.g., for Erdős-Rényi graphs) than for heterogeneous graphs (e.g., for Chung-Lu graphs), which contradicts the observed behavior.

Second, the proof relies on the independence of edges. In fact, this is the only assumption, which makes the same proof hold for multiple different models. However, this assumption is unrealistic for most real-world networks. The dependence between edges is typically measured with the *clustering coefficient*. The *local clustering coefficient* of a vertex v is the probability that two randomly chosen neighbors of v are adjacent. The clustering coefficient of the graph is the average of all local coefficients. The assumption of independent edges thus implies a clustering coefficient close to 0. In contrast, the three best performing instances in [6, Figure 2] have comparatively high clustering coefficients 0.47, 0.49, and 0.57 [14].

In this paper, we analyze the bidirectional BFS on hyperbolic random graphs, which are generated by randomly placing vertices in the hyperbolic plane and connecting each pair that is geometrically close. This model was introduced by Krioukov et al. [13] with the aim to generate graphs that closely resemble real-world networks. Hyperbolic random graphs in particular have a power-law degree distribution and high clustering [12, 13]. Moreover, as these properties emerge naturally from the hyperbolic geometry, the model is conceptually simple, which makes it accessible to mathematical analysis. It has thus gained popularity in different research areas and has been studied from different perspectives.

From the network-science perspective, the goal is to gather knowledge about real-world networks. This is for example achieved by assuming that a real-world network has a hidden underlying hyperbolic geometry, which can be revealed by embedding it into the hyperbolic plane [1, 5]. From the mathematical perspective, the focus lies on studying structural properties. Beyond degree distribution and clustering [12], the diameter [11], the component structure [4], the clique size [3], and separation properties [2] have been studied successfully.

Finally, there is the algorithmic perspective, which is the focus of this paper. Usually algorithms are analyzed by proving worst-case running times. Though this is the strongest possible performance guarantee, it is rather pessimistic as practical instances rarely resemble worst-case instances. Techniques leading to a more realistic analysis include parameterized

or average case complexity. The latter is based on the assumption that instances are drawn from a certain probability distribution. Thus, its explanatory power depends on how realistic the distribution is. For hyperbolic random graphs, the maximum clique can be computed in polynomial time [3], and there are several algorithmic results based on the fact that hyperbolic random graphs have sublinear tree width [2]. Moreover, there is a compression algorithm that can store a hyperbolic random graph using on $O(n)$ bits in expectation [8, 15]. Finally, a close approximation of the shortest path between two nodes can be found using greedy routing, which visits only $\mathcal{O}(\log \log n)$ nodes for most start-destination pairs [9]. The downside of all these algorithms is that they need to know the underlying geometry, i.e., the coordinates of each vertex. Unfortunately, this is a rather unrealistic assumption for real-world networks. To the best of our knowledge, we present the first analysis of an algorithm on hyperbolic random graphs that is oblivious to the underlying geometry.

Contribution and Outline. After an introduction to hyperbolic random graphs in Section 2, we analyze the bidirectional BFS in Section 3. We first prove in Section 3.1 that a certain greedy strategy for deciding when to alternate between the forward and the backward search is not much worse than any other alternation strategy. We note that this result is interesting in its own right and does not depend on the input. In Section 3.2 we analyze the bidirectional BFS on hyperbolic random graphs. We show that, for any pair of vertices, it computes a shortest path in $\tilde{\mathcal{O}}(n^{2-1/\alpha} + n^{1/(2\alpha)} + \delta_{\max})$ time with high probability[1], where $\alpha \in (0.5, 1)$ controls the power-law exponent and $\delta_{\max}$ is the maximum degree of the graph (which is $\tilde{\mathcal{O}}(n^{1/(2\alpha)})$ almost surely [12]). We note that drawing the hyperbolic random graph is the only random choice here; once this is done our analysis always assumes the worst case. Thus, the bound in particular holds for every start-destination pair. Section 4 contains concentration bounds that were left out in Section 3 to improve readability. In Section 5, we conclude by comparing our theoretical results to empirical data.

2 Preliminaries

Let $G = (V, E)$ be an undirected, unweighted, and connected graph. We denote the number of vertices and edges with n and m, respectively. With $N(v) = \{w \in V \mid \{v, w\} \in E\}$, we denote the *neighborhood* of a vertex $v \in V$. The *degree* of v is $\deg(v) = |N(v)|$. We denote the maximum degree with $\delta_{\max}$. The soft O-notation $\tilde{\mathcal{O}}$ suppresses poly-logarithmic factors.

The Hyperbolic Plane. The major difference between hyperbolic and Euclidean geometry is the exponential expansion of the hyperbolic plane. A circle of radius r has area $2\pi(\cosh(r)-1)$ and circumference $2\pi \sinh(r)$, with $\cosh(x) = (e^x + e^{-x})/2$ and $\sinh(x) = (e^x - e^{-x})/2$, both growing as $e^x/2 \pm o(1)$. To identify points, we use radial coordinates with respect to a designated origin O and a ray starting at O. A point p is uniquely determined by its *radius* r, which is the distance to O, and the *angle* (or *angular coordinate*) φ between the reference ray and the line through p and O. In illustrations, we use the *native representation*, obtained by interpreting the hyperbolic coordinates as polar coordinates in the Euclidean plane, see Figure 1 (left). Due to the exponential expansion, line segments bend towards the origin O. Let $p_1 = (r_1, \varphi_1)$ and $p_2 = (r_2, \varphi_2)$ be two points. The *angular distance* between p_1 and p_2 is the angle between the rays from the origin through p_1 and p_2. Formally, it is

[1] *With high probability* and *almost surely* refer to probabilities $1 - O(1/n)$ and $1 - o(1)$, respectively.

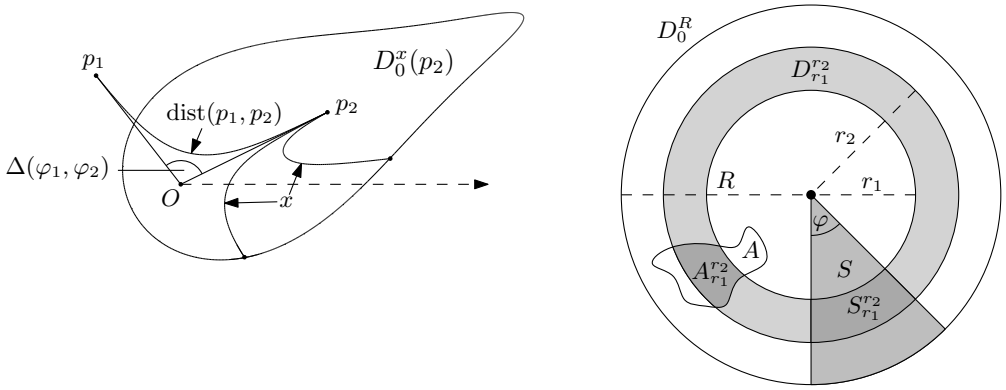


Figure 1 Left: A circle and several line segments in the native representation of the hyperbolic plane. A disk of radius x is centered at p_2. Right: Geometric shapes and their intersections.

$\Delta(\varphi_1, \varphi_2) = \pi - |\pi - |\varphi_1 - \varphi_2||$. The hyperbolic distance $\text{dist}(p_1, p_2)$ is given by

$$\cosh(\text{dist}(p_1, p_2)) = \cosh(r_1)\cosh(r_2) - \sinh(r_1)\sinh(r_2)\cos(\Delta(\varphi_1, \varphi_2)).$$

Note how the angular coordinates make simple definitions cumbersome as angles are considered modulo 2π, leading to a case distinction depending on where the reference ray lies. Whenever possible, we implicitly assume that the reference ray was chosen such that we do not have to compute modulo 2π. Thus, the above angular distance between p_1 and p_2 simplifies to $|\varphi_1 - \varphi_2|$. A third point $p = (r, \varphi)$ *lies between* p_1 and p_2 if $\varphi_1 \leq \varphi \leq \varphi_2$ or $\varphi_2 \leq \varphi \leq \varphi_1$.

Throughout the paper, we regularly use different geometric shapes, which are mostly based on disks centered at the origin O, as can be seen in Figure 1 (right). With $D_{r_1}^{r_2}$ we denote the set of points that have radius between r_1 and r_2. Note that D_0^r is the disk of radius r centered at O. The restriction of a disk D_0^r to all points with angular coordinates in a certain interval is called *sector*, which we usually denote with the letter S. Its *angular width* is the length of this interval. For an arbitrary set of points A, $A_{r_1}^{r_2}$ denotes the restriction of A to points with radii in $[r_1, r_2]$, i.e., $A_{r_1}^{r_2} = A \cap D_{r_1}^{r_2}$.

Hyperbolic Random Graphs. A *hyperbolic random graph* is generated by drawing n points uniformly at random in a disk of the hyperbolic plane and connecting pairs of points whose distance is below a threshold. More precisely, the model depends on two parameters C and α. The generated graphs have a power-law degree distribution with power-law exponent $\beta = 2\alpha + 1$ and with an average degree depending on C. The n points are sampled within the disk D_0^R of radius $R = 2\log n + C$. For each vertex, the angular coordinate is drawn uniformly from $[0, 2\pi]$. The radius r is sampled according to the probability density function

$$f(r) = \frac{1}{2\pi}\frac{\alpha\sinh(\alpha r)}{\cosh(\alpha R) - 1} = \Theta(e^{\alpha(r-R)}), \tag{1}$$

for $r \in [0, R]$. For $r > R$, $f(r) = 0$. Two vertices are connected by an edge if and only if their hyperbolic distance is less than R. The above probability distribution is a natural choice as the probability for a vertex ending up in a certain region is proportional to its area (at least for $\alpha = 1$). Note that the exponential growth in r reflects the fact that the area of a disk grows exponentially with the radius. It follows that a hyperbolic random graph typically has few vertices with high degree close to the center of the disk and many vertices with low degree near its boundary. The following lemma is common knowledge; see the full version of the paper for a proof.

▶ **Lemma 1.** *Let G be a hyperbolic random graph. Furthermore, let v_1, v_2 be two nodes with radii $r_1 \leq r_2 \leq R$, respectively, and with the same angular coordinate. Then $N(v_2) \subseteq N(v_1)$.*

Given two vertices with fixed radii r_1 and r_2, their hyperbolic distance grows with increasing angular distance. The maximum angular distance such that they are still adjacent [12, Lemma 3.1] is

$$\theta(r_1, r_2) = \arccos\left(\frac{\cosh(r_1)\cosh(r_2) - \cosh(R)}{\sinh(r_1)\sinh(r_2)}\right) = 2e^{\frac{R-r_1-r_2}{2}}(1 + \Theta(e^{R-r_1-r_2})). \tag{2}$$

The probability that a sampled node falls into a given subset $A \subseteq D_0^R$ of the disk is given by its probability measure $\mu(A) = \int_A f(r)\,\mathrm{d}r$, which can be thought of as the area of A. There are two types of regions we encounter regularly: disks D_0^r with radius r centered at the origin and disks $D_0^R(r, \varphi)$ of radius R centered at a point (r, φ). Note that the measure of $D_0^R(r, \varphi)$ gives the probability that a random vertex lies in the neighborhood of a vertex with position (r, φ). Gugelmann et al. [12, Lemma 3.2] showed that

$$\mu(D_0^r) = e^{\alpha(r-R)}(1 + o(1)), \text{ and} \tag{3}$$

$$\mu(D_0^R(r, \varphi)) = \Theta(e^{-r/2}). \tag{4}$$

For a given region $A \subseteq D_0^R$ of the disk, let $X_1, \dots, X_n$ be random variables with $X_i = 1$ if $i \in A$ and $X_i = 0$ otherwise. Then $X = \sum_{i=1}^n X_i$ is the number of vertices lying in A. By the linearity of expectation, we obtain that the expected number of vertices in A is $\mathbb{E}[X] = \sum_{i=1}^n \mathbb{E}[X_i] = n\mu(A)$. To bound the number of vertices in A *with high probability*, we regularly use the following Chernoff bound.

▶ **Theorem 2** (Chernoff Bound [10, A.1])**.** *Let $X_1, \dots, X_n$ be n independent random variables with $X_i \in \{0, 1\}$ and let X be their sum. For any $\delta > 0$,*

$$\Pr[X > (1 + \delta)\mathbb{E}[X]] < \exp\left(-\frac{\delta^2}{3}\mathbb{E}[X]\right).$$

▶ **Corollary 3.** *Let $X_1, \dots, X_n$ be n independent random variables with $X_i \in \{0, 1\}$ and let X be their sum. Let $f(n) = \Omega(\log n)$. If $f(n)$ is an upper bound for $\mathbb{E}[X]$, then for any constant c there is a constant c' such that $X \leq c' f(n)$ holds with probability $1 - \mathcal{O}(n^{-c})$.*

3 Bidirectional BFS

In this section, we analyze the running time of the bidirectional BFS on hyperbolic random graphs. Our results are summarized in the following main theorem.

▶ **Theorem 4.** *Let G be a hyperbolic random graph. With high probability the shortest path between any two vertices in G can be computed in $\tilde{\mathcal{O}}(n^{2-1/\alpha} + n^{1/(2\alpha)} + \delta_{\max})$ time.*

To prove this, we make use of the hyperbolic geometry in the following way; see Figure 2. As long as the two searches visit only low-degree vertices, all explored vertices lie within a small region, i.e., the searches operate locally. Once the searches visit high-degree vertices closer to the center of the hyperbolic disk (gray area in Figure 2), it takes only few steps to complete the search, as hyperbolic random graphs have a densely connected core. Thus, we split our analysis in two phases: a first phase in which both searches advance towards the center and a second phase in which both searches meet in the center. Note that this strategy

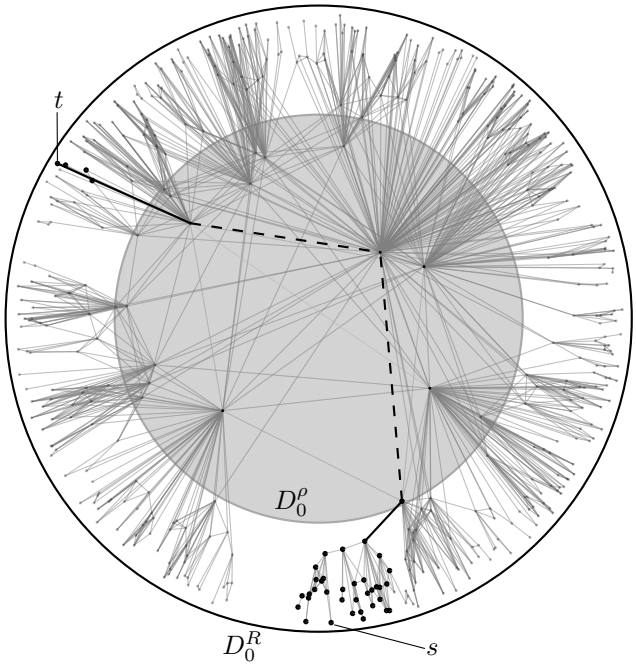


Figure 2 Visualization of the two phases of each BFS in a hyperbolic random graph. Nodes that are visited during the first phase are bold. The bold black edges denote the first encounter of a node in the inner disk D_0^ρ (gray region). This corresponds to the first step in the second phase. The last step then leads to a common neighbor via the dashed edges.

assumes that we know the coordinates of the vertices as we would like to stop a search once it reached the center. To resolve this issue, we first show in Section 3.1 that there exists an alternation strategy that is oblivious to the geometry but performs not much worse than any other alternation strategy. We note that this result is independent of hyperbolic random graphs and thus interesting in its own right. Afterwards, in Section 3.2, we actually analyze the bidirectional search in hyperbolic random graphs.

3.1 Bidirectional Search and Alternation Strategies

In an unweighted and undirected graph $G = (V, E)$, a BFS finds the shortest path between two vertices $s, t \in V$ by starting at s and exploring the graph level after level, where the ith level L_i^s contains the vertices with distance i to s. More formally, the BFS starts with the set $L_0^s = \{s\}$ on level 0. Assuming the first i levels $L_1^s, \dots, L_i^s$ have been computed already, one obtains the next level L_{i+1}^s as the set of neighbors of vertices in level L_i^s that are not contained in earlier layers. Computing L_{i+1}^s from L_i^s is called an *exploration step*, which is obtained by *exploring the edges* between vertices in L_i^s and L_{i+1}^s.

The bidirectional BFS runs two BFSs simultaneously. The *forward search* starts at s and the *backward search* starts at t. The shortest path between the two vertices can then be obtained, once the search spaces of the forward and backward search touch. Since the two searches cannot actually be run simultaneously, they alternate depending on their progress. When exactly the two searches alternate is determined by the *alternation strategy*. Note that we only swap after full exploration steps, i.e., we never explore only half of level i of one search before continuing with the other. This has the advantage that we can be certain to know the shortest path once a vertex is found by both searches.

In the following we define the *greedy alternation strategy* as introduced by Borassi and Natale [6] and show that it is not much worse than any other alternation strategy. Assume the latest layers of the forward and backward searches are L_i^s and L_j^t, respectively. Then the next exploration step of the forward search would cost time proportional to $c_i^s := \sum_{v \in L_i^s} \deg(v)$, while the cost for the backward search is $c_j^t := \sum_{v \in L_j^t} \deg(v)$. The *greedy alternation strategy* then greedily continues with the search that causes the fewer cost in the next exploration step, i.e., it continues with the forward search if $c_i^s \leq c_j^t$ and with the backward search otherwise.

▶ **Theorem 5.** *Let G be a graph with diameter d. If there exists an alternation strategy such that the bidirectional BFS explores $f(n)$ edges, then the bidirectional BFS with greedy alternation strategy explores at most $d \cdot f(n)$ edges.*

Proof. Let A be the alternation strategy that explores only $f(n)$ edges. First note that the number of explored edges only depends on the number of layers explored by the two different searches and not on the actual order in which they are explored. Thus, if the greedy alternation strategy is different from A, we can assume without loss of generality that the greedy strategy performed more exploration steps in the forward search and fewer in the backward search compared to A. Let c^s and c^t be the number of edges explored by the forward and backward search, respectively, when using the greedy strategy. Moreover, let j be the last layer of the backward search (which is actually not explored) and, accordingly, let c_j^t be the number of edges the next step in the backward search would have explored. Then $c^t + c_j^t \leq f(n)$ as, when using A, the backward search still explores layer j. Moreover, the forward search with the greedy strategy explores at most $c^t + c_j^t$ (and therefore at most $f(n)$) edges in each step, as exploring the backward search would be cheaper otherwise. Consequently, each step in the forward and backward search costs at most $f(n)$. As there are at most d steps in total, we obtain the claimed bound. ◀

3.2 Bidirectional Search in Hyperbolic Random Graphs

To analyze the size of the search space of the bidirectional BFS in hyperbolic random graphs, we separate the whole disk D_0^R into two partitions. One is the *inner disk* D_0^ρ centered at the origin. Its radius ρ is chosen in such a way that any two vertices in D_0^ρ have a common neighbor with high probability. The second part is the *outer band* D_ρ^R, the remainder of the whole disk. A single BFS now explores the graph in two phases. In the first phase, the BFS explores vertices in the outer band. The phase ends, when the next vertex to be encountered lies in the inner disk. Once both BFSs completed the first phase, they only need at most two more steps for their search spaces to share a vertex. One step to encounter the vertex in the inner disk and another step to meet at their common neighbor that any two vertices in the inner disk have with high probability; see Figure 2.

For our analysis we assume an alternation strategy in which each search stops once it explored one additional layer after finding the first vertex in the inner disk D_0^ρ. Of course, this cannot be implemented without knowing the underlying geometry of the network. However, by Theorem 5 the search space explored using the greedy alternation strategy is only a poly-logarithmic factor larger, as the diameter of hyperbolic random graphs is poly-logarithmic with high probability [11]. The following lemma shows for which choice of ρ the above sketched strategy works.

▶ **Lemma 6.** *Let G be a hyperbolic random graph. With high probability, G contains a vertex that is adjacent to every other vertex in D_0^ρ, for $\rho = \frac{1}{\alpha}(\log n - \log\log n)$.*

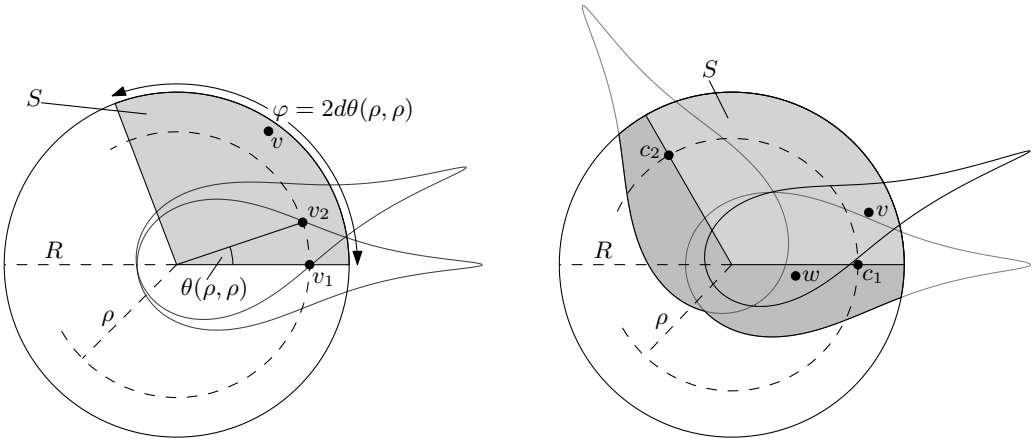


Figure 3 Left: The sector S of angular width φ contains the search space of a BFS in the outer band D_ρ^R starting at v. The vertices v_1 and v_2 are at maximum angular distance to still be adjacent. Right: Neighbor w of vertex v is in S (gray) or a neighbor of c_1 or c_2 (dark gray).

Proof. Assume v is a vertex with radius at most $R-\rho$. Note that the distance between two points is upper bounded by the sum of their radii. Thus, every vertex in D_0^ρ has distance at most R to v, and is therefore adjacent to v. Hence, to prove the claim, it suffices to show the existence of this vertex v with radius at most $R-\rho$. As described in Section 2, the probability for a single vertex to have radius at most $R-\rho$ is given by the measure $\mu(D_0^{R-\rho})$. Using Equation (3) we obtain

$$\begin{aligned}\mu(D_0^{R-\rho}) &= e^{-\alpha\rho}(1+o(1)) \\ &= \frac{\log n}{n}(1+o(1)).\end{aligned}$$

Thus, the probability that none of the n vertices lies in $D_0^{R-\rho}$ is $O((1-\frac{\log n}{n})^n) = O(\frac{1}{n})$. Hence, there is at least one vertex with radius at most $R-\rho$ with high probability. ◀

In the following, we first bound the search space explored in the first phase, i.e., before we enter the inner disk D_0^ρ. Afterwards we bound the search space explored in the second phase, which consists of two exploration steps. The first one to enter D_0^ρ and the second one to find a common neighbor, which exists due to Lemma 6.

3.2.1 Search Space in the First Phase

To bound the size of the search space in the outer band, we make use of the network geometry in the following way. For two vertices in the outer band to be adjacent, their angular distance has to be small. Moreover, the number of exploration steps is bounded by the diameter of the graph. Thus, the maximum angular distance between vertices visited in the first phase cannot be too large. Note that following lemma restricts the search to a sublinear portion of the disk, which we later use to show that also the number of explored edges is sublinear.

▶ **Lemma 7.** *With high probability, all vertices a BFS on a hyperbolic random graph explores before finding a vertex with radius at most* $\rho = \frac{1}{\alpha}(\log n - \log\log n)$ *lie within a sector of angular width* $\tilde{\mathcal{O}}(n^{1-1/\alpha})$.

Proof. For an illustration of the proof see Figure 3 (left). Recall from Section 2 that $\theta(r_1, r_2)$ denotes the maximum angular distance between two vertices of radii r_1 and r_2 such that

they are still adjacent. As this angle increases with decreasing radii, $\theta(r_1, r_2) \leq \theta(\rho, \rho)$ holds for all vertices in the outer band D_ρ^R.

Now assume we start a BFS at a vertex $v \in D_\rho^R$ and perform d exploration steps without leaving the outer band D_ρ^R. Then no explored vertex has angular distance more than $d\theta(\rho, \rho)$ from v. Thus, the whole search space lies within a disk sector of angular width $2d\theta(\rho, \rho)$. The number of steps d is at most poly-logarithmic as the diameter of a hyperbolic random graph is poly-logarithmic with high probability [11]. Using Equation (2) for $\theta(\rho, \rho)$, we obtain

$$\begin{aligned}\theta(\rho, \rho) &= 2e^{\frac{R-2\rho}{2}}(1 + \Theta(e^{R-2\rho})) \\ &= 2e^{C/2}n^{1-1/\alpha}\log^{1/\alpha} n(1 + \Theta((\log n/n^{1-\alpha})^{2/\alpha}) \\ &= \mathcal{O}(n^{1-1/\alpha}\log^{1/\alpha} n),\end{aligned}$$

which proves the claimed bound. ◀

Note that the expected number of vertices in a sector S of angular width φ is linear in $n\varphi$ due to the fact that the angular coordinate of each vertex is chosen uniformly at random. Thus, Lemma 7 already shows that the expected number of vertices visited in the first phase of the BFS is $\tilde{\mathcal{O}}(n^{2-1/\alpha})$, which is sublinear in n. It is also not hard to see that this bound holds with high probability (see Corollary 3). To also bound the number of explored edges, we sum the degrees of vertices in S. It is not surprising that this yields the same asymptotic bound in expectation, as the expected average degree in a hyperbolic random graph is constant. However, showing that this value is concentrated around its expectation is more involved. Though we can use techniques similar to those that have been used to show that the average degree of the whole graph is constant with high probability [7, 12], the situation is complicated by the restriction to a sublinear portion of the disk. Nonetheless, we obtain the following theorem.

▶ **Theorem 8.** *Let G be a hyperbolic random graph. The degrees of vertices in every sector of angular width φ sum to $\tilde{\mathcal{O}}(\varphi n + \delta_{\max})$ with high probability if $\varphi = \Omega(n^{1-1/\alpha}\log n)$.*

We note that $\delta_{\max}$ has to be included here, as the theorem states a bound for every sector, and thus in particular for sectors containing the vertex of maximum degree. Recall, that $\delta_{\max} = \tilde{\mathcal{O}}(n^{1/(2\alpha)})$ holds almost surely [12]. Moreover, we note that the condition $\varphi = \Omega(n^{1-1/\alpha}\log n)$ is crucial for our proof, i.e., the angular width of the sector has to be sufficiently large for the concentration bound to hold. Fortunately, this matches the bound found in Lemma 7. As the proof for Theorem 8 is rather technical, we defer it to Section 4. Together with Lemma 7, we obtain the following corollary.

▶ **Corollary 9.** *On a hyperbolic random graph, the first phase of the bidirectional search explores with high probability only $\tilde{\mathcal{O}}(n^{2-1/\alpha} + \delta_{\max})$ many edges.*

3.2.2 Search Space in the Second Phase

The first phase of the BFS is completed when the next vertex to be encountered lies in the inner disk. Thus, the second phase consists of only two exploration steps. One step to encounter the vertex in the inner disk and another step to meet the other search. Thus, to bound the running time of the second phase, we have to bound the number of edges explored in these two exploration steps. To do this, let V_1 be the set of vertices encountered in the first phase. Recall that all these vertices lie within a sector S of angular width $\varphi = \tilde{\mathcal{O}}(n^{1-1/\alpha})$ (Lemma 7). The number of explored edges in the second phase is then bounded by the sum of

degrees of all neighbors $N(V_1)$ of vertices in V_1. To bound this sum, we divide the neighbors of V_1 into two categories: $N(V_1) \cap S$ and $N(V_1) \setminus S$. Note that we already bounded the sum of degrees of vertices in S for the first phase (see Theorem 8), which clearly also bounds this sum for $N(V_1) \cap S$. Thus, it remains to bound the sum of degrees of vertices in $N(V_1) \setminus S$.

To bound this sum, we introduce two *hypothetical vertices* (i.e., vertices with specific positions that are not actually part of the graph) c_1 and c_2 such that every vertex in $N(V_1) \setminus S$ is a neighbor of c_1 or c_2. Then it remains to bound the sum of degrees of neighbors of these two vertices. To define c_1 and c_2, recall that the first phase was not only restricted to the sector S but also to points with radius greater than ρ, i.e., all vertices in V_1 lie within S_ρ^R. The hypothetical vertices c_1 and c_2 are basically positioned at the corners of this region, i.e., they both have radius ρ, and they assume the maximum and minimum angular coordinate within S, respectively. Figure 3 (right) shows these positions. We obtain the following.

▶ **Lemma 10.** *Let G be a hyperbolic random graph and let $v \in S_\rho^R$ for a sector S. Then, every neighbor of v lies in S or is a neighbor of one of the hypothetical vertices c_1 or c_2.*

Proof. Let $v = (r, \varphi) \in S_\rho^R$ and $w \in N(v) \setminus S$. Without loss of generality, assume that c_1 lies between v and w, as is depicted in Figure 3 (right). Now consider the point $v' = (\rho, \varphi)$ obtained by moving v to the same radius as c_1. According to Lemma 1 we have $N(v) \subseteq N(v')$. In particular, it holds that $w \in N(v')$ and therefore $\operatorname{dist}(v', w) \leq R$. Since v' and c_1 have the same radial coordinate and c_1 is between v' and w, it follows that $\operatorname{dist}(c_1, w) \leq R$. ◀

By the above argumentation, it remains to sum the degrees of neighbors of c_1 and c_2. It is not hard to see that the degrees of the neighbors of a node with radius r sum to $\mathcal{O}(ne^{-(\alpha-1/2)r})$ in expectation. For c_1 and c_2, which both have radius ρ, the degrees of their neighbors thus sum to $\tilde{\mathcal{O}}(n^{1/(2\alpha)})$ in expectation. Note that this matches the claimed bound in Theorem 4. However, to actually prove Theorem 4, we need to show that this bound holds with high probability for every possible angular coordinates of c_1 and c_2. Again, showing this concentration bound is rather technical and thus deferred to Section 4. Together with the bounds on the sum of degrees in a sector of width $\varphi = \tilde{\mathcal{O}}(n^{1-1/\alpha})$ (Theorem 8), we obtain the following corollary, which concludes the proof of Theorem 4.

▶ **Corollary 11.** *On a hyperbolic random graph, the second phase of the bidirectional search explores with high probability only $\tilde{\mathcal{O}}(n^{2-1/\alpha} + n^{1/(2\alpha)} + \delta_{\max})$ many edges.*

4 Concentration Bounds for the Sum of Vertex Degrees

Here we prove the concentration bounds that were announced in the previous section. For the first phase, we already know that the search space is contained within a sector S of sublinear width (Lemma 7). Thus, the running time in the first phase is bounded by the sum of vertex degrees in this sector. Moreover, all edges explored in the second phase also lie within the same sector S or are incident to neighbors of the two hypothetical vertices c_1 and c_2 (Lemma 10). Thus, the running time of the second phase is bounded by the sum of vertex degrees in S and in the neighborhood of c_1 and c_2. We start by proving Theorem 8 to bound the sum of degrees in a given sector. Afterwards, we consider the neighborhood of c_1 and c_2. To improve readability, we restate Theorem 8 here.

▶ **Theorem 8.** *Let G be a hyperbolic random graph. The degrees of vertices in every sector of angular width φ sum to $\tilde{\mathcal{O}}(\varphi n + \delta_{\max})$ with high probability if $\varphi = \Omega(n^{1-1/\alpha} \log n)$.*

Due to space constraints, we only sketch the proof by explaining the overall strategy and stating the core lemmas. A full proof can be found in the full version of the paper. The proof of Theorem 8 basically works as follows. For each degree, we want to compute the number of vertices of this degree and multiply it with the degree. As all vertices with a certain degree have roughly the same radius, we can separate the disk into small bands, one for each degree. Then summing over all degrees comes down to summing over all bands and multiplying the number of vertices in this band with the corresponding degree. If we can prove that each of these values is highly concentrated (i.e., probability $1 - O(n^{-2})$), we obtain that the sum is concentrated as well (using the union bound). Unfortunately, this fails in two situations. For large radii the degree is too small to be concentrated around its expected value. Moreover, for small radii, the number of vertices within the corresponding band (i.e., the number of high degree vertices) is too small to be concentrated.

To overcome this issue, we partition the sector S into three parts. An inner part $S_0^{\rho_1}$, containing all points of radius at most ρ_1, an outer part $S_{\rho_2}^R$, containing all points of radius at least ρ_2, and a central part $S_{\rho_1}^{\rho_2}$, containing all points in between. We choose ρ_2 in such a way that the smallest degree in the central part $S_{\rho_1}^{\rho_2}$ is $\Omega(\log n)$, which ensures that all vertex degrees in $S_{\rho_1}^{\rho_2}$ are concentrated. Moreover, we choose ρ_1 such that the number of vertices with maximum degree in $S_{\rho_1}^{\rho_2}$ is $\Omega(\log n)$, which ensures that for each vertex degree, the number of vertices with this degree is concentrated. To achieve this, we set

$$\rho_1 = 2\log n - \frac{\log(\varphi n) - \log\log n}{\alpha} \quad \text{and} \quad \rho_2 = \frac{\log n}{\alpha},$$

and show concentration separately for the three parts.

The Inner Part of a Sector. The inner part $S_0^{\rho_1}$ contains vertices of high degree. It is not hard to see that there are only poly-logarithmically many vertices with radius at most ρ_1. Thus, we obtain the following lemma.

▶ **Lemma 12.** *Let G be a hyperbolic random graph. For every sector S of angular width φ, the degrees of the nodes in $S_0^{\rho_1}$ sum to $\tilde{\mathcal{O}}(\delta_{\max})$ with high probability.*

The Central Part of a Sector. For each possible vertex degree k, we want to compute the number of vertices with this degree in the central part $S_{\rho_1}^{\rho_2}$. First note, that by Equation (4) a vertex with fixed radius has expected degree $\Theta(k)$ if this radius is $2\log(n/k)$. Motivated by this, we define $r_k = 2\log(n/k)$. To bound the sum of degrees in the central part $S_{\rho_1}^{\rho_2}$, we use that vertices with radius significantly larger than r_k also have a smaller degree. More formally, one can show that there exists a constant ε such that all vertices of degree k have radius at most $r_k + \varepsilon$ with high probability. From this, we can derive a bound $g(k)$ for the number of vertices with degree at least k by bounding the number of vertices with radius at most $r_k + \varepsilon$. Then summing the vertex degrees boils down to integrating over $g(k)$, which yields the following lemma.

▶ **Lemma 13.** *Let G be a hyperbolic random graph. For every sector S of angular width φ, the degrees of the nodes in $S_{\rho_1}^{\rho_2}$ sum to $\mathcal{O}(\varphi n^{3-2\alpha-1/(2\alpha)})$ with high probability.*

Note that $3 - 2\alpha - 1/(2\alpha) \leq 1$ for $\alpha \in [0.5, 1]$. Thus, the lemma in particular shows that $S_{\rho_1}^{\rho_2}$ contains at most $\mathcal{O}(\varphi n)$ edges, as claimed in Theorem 8.

The Outer Part of a Sector. The outer part $S_{\rho_2}^R$ contains many vertices, all of which have low expected degree. To bound their sum with high probability, we consider the coordinates of the vertices as random variables and the sum of their degrees as a function in these

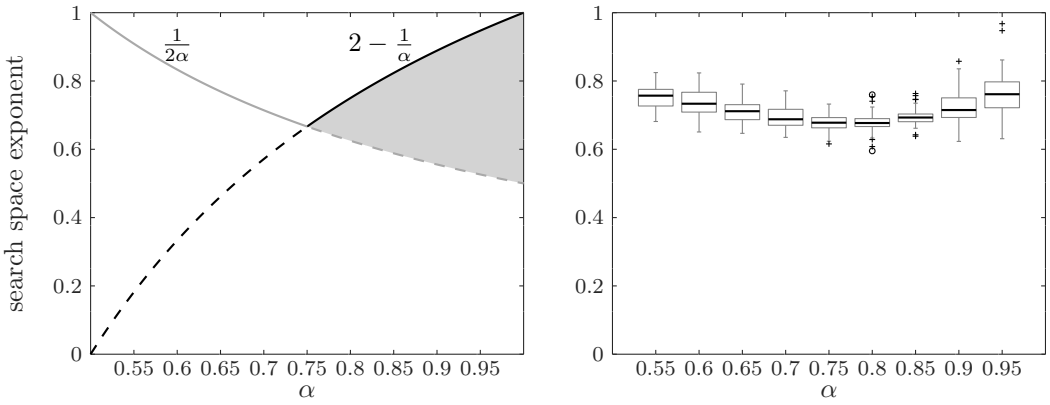


Figure 4 Left: The exponent of our theoretical bound depending on α. Right: The corresponding empirically measured search spaces. The data was obtained by generating 20 hyperbolic random graphs with average degree roughly 8 for each shown α and each $n \in \{100\text{k}, 200\text{k}, 300\text{k}\}$. For each graph we sampled 300k start-destination pairs and report the maximum number of edges explored in one search. The numbers are normalized with the total number of edges m of the graph such that x is plotted for a search space of size m^x.

variables. Then, our plan to show concentration is to apply a method of average bounded differences [10, Theorem 7.2]. It is based on the fact that changing the value of a single random variable (i.e., moving the position of a single vertex) has only little effect on the function (i.e., on the sum of degrees). To make sure that this is actually true, we exclude certain bad events that happen only with low probability: First, the maximum degree in $S^R_{\rho_2}$ should not be too high such that moving a single vertex can increase its degree only slightly. Second, there should not be too many vertices in $S^R_{\rho_2}$ such that the sum of degrees actually changes only for few vertices (as we do not count vertices not in $S^R_{\rho_2}$). Overall we obtain the following lemma.

▶ **Lemma 14.** *Let G be a hyperbolic random graph. For every sector S of angular width φ, the degrees of the nodes in $S^R_{\rho_2}$ sum to $\mathcal{O}(\varphi n)$ with high probability if $\varphi = \Omega(n^{1-1/\alpha} \log n)$.*

The Neigborhood of a Vertex with Radius ρ. For the second phase, we showed in Section 3.2.2 that it remains to bound the sum of degrees in the neighborhood of the corner vertices. Recall that they both have radius $\rho = 1/\alpha(\log n - \log\log n)$. Let v be a vertex with radius ρ and let A be the disk with radius R around v. Note that we already know from Section 3.2 that the maximum angular distance of neighbors of v with radius at least ρ is $\tilde{\mathcal{O}}(n^{1-1/\alpha})$. Thus, A^R_ρ is contained within a sector of this width and we can use Theorem 8 to obtain the desired bound for this part. Moreover, as in Lemma 12, we can bound the number of vertices with small radius. For all radii in between, A contains a sector of angular width $\Omega(n^{1-1/\alpha})$. It is thus not surprising that for each degree occurring in this part, the number of vertices of this degree is concentrated around its expectation. Hence, similar arguments as for Lemma 13, lead to the following lemma.

▶ **Lemma 15.** *Let G be a hyperbolic random graph and let v be a hypothetical vertex with radius $\rho = 1/\alpha(\log n - \log\log n)$ and arbitrary angular coordinate. The degrees of neighbors of v sum to $\tilde{\mathcal{O}}(n^{2-1/\alpha} + n^{1/(2\alpha)} + \delta_{\max})$ with high probability.*

5 Conclusion

To conclude, we discuss why we think that the bound $\tilde{\mathcal{O}}(n^{2-1/\alpha} + n^{1/(2\alpha)} + \delta_{\max})$ is rather tight; see Figure 4 (left) for a plot of the exponents. Clearly, the maximum degree of the graph is a lower bound, i.e., we cannot improve the $\delta_{\max}$. As $\delta_{\max} = \tilde{\Theta}(n^{1/(2\alpha)})$ holds almost surely [12], we also cannot improve below $\tilde{\mathcal{O}}(n^{1/(2\alpha)})$. For the term $n^{2-1/\alpha}$ we do not have a lower bound. Thus, the gray region in Figure 4 (left) is the only part where our bound can potentially be improved. However, by only making a single step from a vertex with radius $\rho = 1/\alpha(\log n - \log\log n)$, we can already reach vertices with angular distance $\Theta(n^{1-1/\alpha})$. Thus, it seems likely, that there exists a start-destination pair such that all vertices within a sector of this angular width are actually explored. As such a sector contains $\Theta(n^{2-1/\alpha})$ vertices, our bound seems rather tight (at least asymptotically and up to poly-logarithmic factors). For a comparison of our theoretical bound with actual search-space sizes in hyperbolic random graphs, see Figure 4.

References

1 Gregorio Alanis-Lobato, Pablo Mier, and Miguel A. Andrade-Navarro. Manifold learning and maximum likelihood estimation for hyperbolic network embedding. *Applied Network Science*, 1(10):1–14, 2016. doi:10.1007/s41109-016-0013-0.

2 Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. Hyperbolic random graphs: Separators and treewidth. In *24th ESA*, volume 57 of *LIPIcs*, pages 15:1–15:16, 2016. doi:10.4230/LIPIcs.ESA.2016.15.

3 Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. Cliques in hyperbolic random graphs. *Algorithmica*, 80:1–21, 2017. doi:10.1007/s00453-017-0323-3.

4 Michel Bode, Nikolaos Fountoulakis, and Tobias Müller. On the giant component of random hyperbolic graphs. In *7th EuroComb*, pages 425–429, 2013. doi:10.1007/978-88-7642-475-5_68.

5 Marián Boguñá, Fragkiskos Papadopoulos, and Dmitri Krioukov. Sustaining the internet with hyperbolic mapping. *Nature Communications*, 1:1–8, 2010. doi:10.1038/ncomms1063.

6 Michele Borassi and Emanuele Natale. KADABRA is an adaptive algorithm for betweenness via random approximation. In *24th ESA*, pages 20:1–20:18, 2016. doi:10.4230/LIPIcs.ESA.2016.20.

7 Karl Bringmann, Ralph Keusch, and Johannes Lengler. Average distance in a general class of scale-free networks with underlying geometry. *CoRR*, abs/1602.05712, 2016. URL: http://arxiv.org/abs/1602.05712.

8 Karl Bringmann, Ralph Keusch, and Johannes Lengler. Sampling geometric inhomogeneous random graphs in linear time. In *25th ESA*, volume 87 of *LIPIcs*, pages 20:1–20:15, 2017. doi:10.4230/LIPIcs.ESA.2017.20.

9 Karl Bringmann, Ralph Keusch, Johannes Lengler, Yannic Maus, and Anisur Rahaman Molla. Greedy routing and the algorithmic small-world phenomenon. In *PODC '17*, pages 371–380, 2017. doi:10.1145/3087801.3087829.

10 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2012.

11 Tobias Friedrich and Anton Krohmer. On the diameter of hyperbolic random graphs. In *42nd ICALP*, pages 614–625, 2015. doi:10.1007/978-3-662-47666-6_49.

12 Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random hyperbolic graphs: Degree sequence and clustering. In *39th ICALP*, pages 573–585, 2012. doi:10.1007/978-3-642-31585-5_51.

13 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Phys. Rev. E*, 82:036106, 2010. doi:10.1103/PhysRevE.82.036106.

14 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, 2014.

15 Ueli Peter. *Random Graph Models for Complex Systems*. PhD thesis, ETH Zürich, 2014.

16 Ira Sheldon Pohl. *Bi-directional and Heuristic Search in Path Problems*. PhD thesis, Stanford University, 1969.

Approximate Convex Hull of Data Streams

Avrim Blum[1]
TTI-Chicago, Chicago, United States
avrim@ttic.edu

Vladimir Braverman[2]
Johns Hopkins University, Baltimore, United States
vova@cs.jhu.edu

Ananya Kumar[3]
Carnegie Mellon University, Pittsburgh, United States
skywalker94@gmail.com

Harry Lang[4]
Johns Hopkins University, Baltimore, United States
hlang8@math.jhu.edu

Lin F. Yang[5]
Princeton University, Princeton, United States
lin.yang@princeton.edu

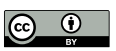
Abstract

Given a finite set of points $P \subseteq \mathbb{R}^d$, we would like to find a small subset $S \subseteq P$ such that the convex hull of S approximately contains P. More formally, every point in P is within distance ϵ from the convex hull of S. Such a subset S is called an ϵ-hull. Computing an ϵ-hull is an important problem in computational geometry, machine learning, and approximation algorithms.

In many applications, the set P is too large to fit in memory. We consider the streaming model where the algorithm receives the points of P sequentially and strives to use a minimal amount of memory. Existing streaming algorithms for computing an ϵ-hull require $O(\epsilon^{(1-d)/2})$ space, which is optimal for a worst-case input. However, this ignores the structure of the data. The minimal size of an ϵ-hull of P, which we denote by OPT, can be much smaller. A natural question is whether a streaming algorithm can compute an ϵ-hull using only $O(\mathsf{OPT})$ space.

We begin with lower bounds that show, under a reasonable streaming model, that it is not possible to have a single-pass streaming algorithm that computes an ϵ-hull with $O(\mathsf{OPT})$ space. We instead propose three relaxations of the problem for which we can compute ϵ-hulls using space near-linear to the optimal size. Our first algorithm for points in $\mathbb{R}^2$ that arrive in random-order uses $O(\log n \cdot \mathsf{OPT})$ space. Our second algorithm for points in $\mathbb{R}^2$ makes $O(\log(\epsilon^{-1}))$ passes before outputting the ϵ-hull and requires $O(\mathsf{OPT})$ space. Our third algorithm, for points in $\mathbb{R}^d$ for any fixed dimension d, outputs, with high probability, an ϵ-hull for all but δ-fraction of directions and requires $O(\mathsf{OPT} \cdot \log \mathsf{OPT})$ space.

[1] This work was supported in part by the National Science Foundation under grant CCF-1525971. Work was done while the author was at Carnegie Mellon University.
[2] This material is based upon work supported in part by the National Science Foundation under Grants No. 1447639, 1650041 and 1652257, Cisco faculty award, and by the ONR Award N00014-18-1-2364.
[3] Now at DeepMind.
[4] This research was supported by the Franco-American Fulbright Commission and supported in part by National Science Foundation under Grant No. 1447639, 1650041 and 1652257. The author thanks INRIA (l'Institut national de recherche en informatique et en automatique) for hosting him during the writing of this paper.
[5] This material is based upon work supported in part by National Science Foundation under Grant No. 1447639, 1650041 and 1652257. Work was done while the author was at Johns Hopkins University.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 21; pp. 21:1–21:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Theory of computation → Computational geometry, Theory of computation → Sketching and sampling, Theory of computation → Streaming models

Keywords and phrases Convex Hulls, Streaming Algorithms, Epsilon Kernels, Sparse Coding

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.21

Related Version A full version of the paper is available at `https://arxiv.org/abs/1712.04564`.

Acknowledgements We want to thank our anonymous reviewers for their useful feedback in preparing this manuscript, and Rachel Holladay, Drew Bagnell, Maya Rau-Murthy, Satya Prateek Tangirala, and Sunny Nahar for discussion of ideas and feedback on paper drafts.

1 Introduction

The question addressed by this paper is: *Can we compute approximate convex hulls of data streams using near-optimal space?* Approximate convex hulls are fundamental in computational geometry, computer vision, data mining, and many more (see e.g. [2]), and computing them in a streaming manner is important in the big data regime.

Our notion of approximate convex hulls is the commonly used ϵ-hull. Let P be a set of n points in $\mathbb{R}^d$. Let $\mathcal{C}(P)$ denote the convex hull of P, the smallest convex set containing P. We want a small subset S of P such that all points in P are inside $\mathcal{C}(S)$ or within distance ϵ from $\mathcal{C}(S)$. Since every point in P can be approximated by a sparse convex combination of points in S, S is also called a generating set [7]. For an example motivation of this particular definition, consider two far-away sensors rapidly collecting data: one of positive examples and the other of negative examples; if it is expected that these should be linearly separable with some margin ϵ, then an appropriate small summary of their data would be an ϵ-hull.

ϵ-hulls and their variants have been studied extensively in the literature. In the multiplicative error variant, ϵ-kernels, one requires that any directional width (the diameter of S in a particular direction) of S is a $(1 \pm \epsilon)$ approximation to that of P. ϵ-hulls and ϵ-kernels are intimately connected: algorithms for ϵ-kernels typically apply a transformation to the data, and then use algorithms for ϵ-hulls. For more details, we refer the reader to [2].

Existing work focuses on worst case bounds, which scale poorly with the dimension d. The worst case lower bound for the size of an ϵ-hull is $\Omega(\epsilon^{-(d-1)/2})$. Recently, it has been shown in [7] that one can do much better than the worst case bound if the size of the smallest ϵ-hull for P (which we denote as OPT) is small. In their paper, they show that one can efficiently obtain S of size nearly linear in OPT and at most linear in the dimension d.

One concern of the algorithms in [7] is that they require storing all points of P in memory. The huge size of real-world datasets limits the applicability of these algorithms. A natural question to ask is whether it is possible to efficiently maintain an ϵ-hull of P when P is presented as a data stream while using a small amount of memory.

We provide both negative and positive results, summarized below.

1.1 Our Contributions

Let OPT be the optimal size of an ϵ-hull. In Section 3, we show, under a reasonable streaming model, that no streaming algorithm can achieve space bounds comparable to OPT. In particular, no streaming algorithm can have space complexity competitive with $f(\mathsf{OPT}, d)$ in 3

dimensions or higher for any $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. This lower bound guides us to consider variants on this problem. Note that the lower bound applies specifically to streaming algorithms; for the batch setting, [7] gives a polynomial-time algorithm that computes an ϵ-hull with space $O(d\mathsf{OPT} \log \mathsf{OPT})$.

We devise and prove the correctness of streaming algorithms for three relaxations of the problem. In Section 4, we show the first relaxation, in which the points are from $\mathbb{R}^2$ and come in a random order. In Section 5, we relax the problem (again in $\mathbb{R}^2$) by allowing the algorithm to make multiple passes over the stream. In Section 6, we show the third relaxation, in which the points come in an arbitrary order and from d-dimensional space, but we only require to approximate the convex hull in a large fraction of all directions.

In the first relaxation, our algorithm maintains an initially empty point set S. When our algorithm sees a new point p, it adds p to S if p is at least distance ϵ away from the convex hull of S. Additionally, our algorithm keeps removing points $p' \in S$ when some p' is contained inside the convex hull of $S \setminus \{p'\}$, that is, removing p' does not change the convex hull of S. Surprisingly, for any point stream P, with high probability this algorithm keeps an ϵ-hull of size $O(\mathsf{OPT} \cdot \log n)$, where n is the size of P.

In the second relaxation, we permit the algorithm to make a small number of passes over the stream. Our algorithm begins the first pass by taking $O(1)$ directions and storing the point with maximal dot product with each direction. In each of $O(\log(\frac{1}{\epsilon}))$ subsequent passes, we refine the solution by adding a new direction in sectors that incurred too much error while potentially deleting old directions that become no longer necessary. The algorithm computes an ϵ-hull of size $O(\mathsf{OPT})$.

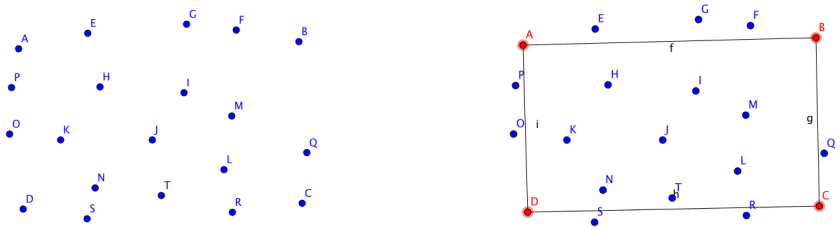
In the third relaxation, we only need to be correct in most directions (all but a δ fraction of directions). Our algorithm picks $O_d(\frac{\mathsf{OPT}}{\delta^2} \log \frac{\mathsf{OPT}}{\delta})$ random unit vectors. For each of these vectors v, we keep the point in the stream that has maximal dot product with v.

To the best of our knowledge, this is the first work that gives lower bounds and streaming algorithms for ϵ-hulls with space complexity comparable to the optimal approximation.

1.2 Related Work

Batch Algorithms. We use the term batch algorithm for an algorithm that stores the entire set of points in memory. In the batch setting, Bentley, Preparata, and Faust [5] give a $O_d(1/\epsilon^{(d-1)})$ space algorithm for computing an ϵ-hull of a set of points (assuming constant dimension d). Agarwal, Har-Peled, and Varadarajan [1] improve the result to give a $O_d(1/\epsilon^{(d-1)/2})$ space algorithm for ϵ-kernels, a multiplicative approximation of convex hulls. The running time bounds were further improved in [8, 10, 12]. Recently, Blum, Har-Peled, and Raichel [7] give the only known batch algorithms for an ϵ-hull that are competitive with the optimal ϵ-hull size of the given point set.

Streaming Algorithms with Worst Case Guarantees. Hershberger and Suri [11] and Agarwal & Yu [3] give 2D one-pass streaming algorithms for ϵ-hulls that uses $O(1/\sqrt{\epsilon})$ space. Agarwal, Har-Peled, and Varadarajan [1] give a one-pass streaming algorithm for ϵ-kernels that uses $O_d((1/\epsilon^{\frac{d-1}{2}}) \log^d n)$ space. Chan [8] removes the dependency on n and gives a streaming algorithm for ϵ-kernels that uses $O_d((1/\epsilon^{d-3/2}) \log^d 1/\epsilon)$ space. This was then improved to $O_d((1/\epsilon^{\frac{d-1}{2}}) \log \frac{1}{\epsilon})$ [13] and the time complexity was further improved by Arya and Chan [4]. Chan [9] also gives a dynamic streaming (allowing deletions in the stream) algorithm based on polynomial methods. All of these space bounds assume a constant dimension d, and focus on worst case guarantees.



(a) Set of points. **(b)** ϵ-hull in red (for suitable ϵ).

Figure 1 ϵ-hull of a set of points.

ϵ-kernels vs ϵ-hulls. Past work focuses on both ϵ-hulls and ϵ-kernels, a multiplicative error variant. ϵ-kernels can be trickier to compute, but are closely related to ϵ-hulls, and often use algorithms for ϵ-hulls as a core subroutine. We focus on ϵ-hulls, but extending this work to ϵ-kernels is an exciting (non-trivial) avenue for future research. In particular, typical reductions from ϵ-kernels to ϵ-hulls (e.g. see [2]) are not compatible with the notion of OPT.

Our Techniques. The proof of our 2D random order algorithm exposes an elegant connection between our 2D result, and a classic 1D result. Our multipass algorithm and (ϵ, δ)-hull algorithm are built on existing methods (e.g. [1, 3]) in that a core subroutine involves preserving the maximal point along certain directions.

2 Preliminaries

▶ **Definition 2.1.** For any bounded set $C \subseteq \mathbb{R}^d$, a point q is ϵ-*close* to C if $\inf_{x \in C} \|q - x\|_2 \leq \epsilon$.

▶ **Definition 2.2.** Given a set of points $P \subseteq \mathbb{R}^n$, $S \subseteq P$ is an ϵ-*hull* of P if for every $p \in P$, p is ϵ-close to $\mathcal{C}(S)$, the convex hull of S.

▶ **Definition 2.3.** Let $\mathsf{OPT}(P, \epsilon)$ denote the number of points in a (not necessarily unique) smallest ϵ-hull of P. We omit P and ϵ if it is clear from the context.

2.1 Streaming Model

Our streaming model, while simple, captures most streaming algorithms for ϵ-hulls in the literature. In our model, a streaming algorithm $\mathcal{A}$ is given ϵ in advance but not the size of the input point stream $P \in \mathbb{R}^d$. P is presented to an algorithm $\mathcal{A}$ sequentially:

$$P = (p_1, p_2, \ldots, p_t, \ldots),$$

where $p_t \in \mathbb{R}^d$ is the point coming at time t. Note that P may have duplicate points. For the ϵ-hull problem, we require Algorithm $\mathcal{A}$ to maintain a subset $S \subseteq P$. For each point $p \in P$, $\mathcal{A}$ can choose to add p to S (remembering p) or ignore p (therefore permanently forgetting p). $\mathcal{A}$ can also choose to delete points in S, in which case these points are permanently lost. After one-pass of the stream, we require S to be an ϵ-hull of the points set P. A trivial streaming algorithm could just keep all points it has seen. However, such an algorithm would not be feasible in the big data regime. Ideally, $\mathcal{A}$ should use space competitive with $\mathsf{OPT}(P, \epsilon)$.

3 Lower Bounds

An (f, r)-optimal algorithm in dimension d uses space competitive with $f(\mathsf{OPT}(P, \epsilon), d)$ and maintains an $(r\epsilon)$-hull where $r > 1$. Note that this definition is rather permissive, since it allows an arbitrary function of OPT and allows slack in ϵ as well.

▶ **Definition 3.1.** For $r \geq 1$, $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, we say a streaming algorithm $\mathcal{A}$ is (f, r)-optimal if given arbitrary $\epsilon > 0$ and point stream $P \subseteq \mathbb{R}^d$, $\mathcal{A}$ keeps an $(r\epsilon)$-hull of P of size at most $f(\mathsf{OPT}(P, \epsilon), d)$.

▶ **Theorem 3.2.** *For all $r \geq 1$, $d \geq 3$, $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, there does not exist an (f, r)-optimal streaming algorithm in $\mathbb{R}^d$.*

Proof. See Theorem A.5, Theorem A.6, and Corollary A.8 in the Appendix of the full version for the proof. Here we give high level intuition for the case $r = 1$, $d = 3$. In our proof, we assume for the sake of contradiction that there exists such a streaming algorithm A and function f. We construct 3 sequences of points P_1, P_2, P_3. Let $P_1 \circ P_2$ denote sequence P_1 followed by sequence P_2 (P_2 appended to P_1). We then show that if A keeps an ϵ-hull of size at most $f(\mathsf{OPT}(P_1 \circ P_2, \epsilon), 3)$ after receiving $P_1 \circ P_2$, then it cannot keep an ϵ-hull of size at most $f(\mathsf{OPT}(P_1 \circ P_2 \circ P_3, \epsilon), 3)$ after receiving $P_1 \circ P_2 \circ P_3$. This is a contradiction.

To do this, we ensure that $|P_2|$ is much larger than $|P_1|$ and that P_1 is an ϵ-hull of $P_1 \circ P_2$. This forces the algorithm to keep only a small proportion of points in $P_1 \circ P_2$. We then ensure that $|P_3|$ is much larger than $|P_2|$ and that P_2 is an ϵ-hull for $P_2 \circ P_3$. However, since the algorithm only kept a small number of points in P_2, it is forced to keep many points in P_3. See the appendix for precise details. The result extends easily for $d > 3$. For $r > 1$, we use a similar construction, but add more sets of points $P_4, P_5, \ldots$ ◀

We can also ask a slightly different question: what if an algorithm is given an additional parameter k in advance, and only needs to maintain an ϵ-hull at time t when OPT of the substream at time t falls below k. The algorithm we give for (ϵ, δ)-hulls in Section 6 is of this form. In the appendix of the full version (Definition A.9 and Theorem A.10), we formulate a lower bound for this case.

Our lower bounds guide future research by showing that we need to think beyond the current streaming models, add reasonable assumptions to the problem, or the space bounds of our algorithms must include some functions of ϵ or $|P|$ (along with OPT and d).

4 2D Random Order Algorithm (ROA)

In many cases, data points are generated i.i.d., for example in mixture models or topic models (e.g. [6]). In this section we assume a more general setup: that the points come in a random order. More precisely, for all sets of points P, every permutation of P must have equal probability density. The case where the data points are generated i.i.d. (*making no assumptions about the distribution*) is a special case. We assume the points are in 2D. To begin, we introduce the following definition.

▶ **Definition 4.1.** A point p is *interior* to P if p is in the convex hull of $P \setminus \{p\}$.

4.1 1D Algorithm

To motivate our 2D algorithm, we begin with a classic result in 1-dimension. Consider the algorithm **ROA-insertion**: Begin by keeping a set $S = \{\}$. For each point $p \in P$ that the algorithm sees, if the distance from p to the convex hull of S is at most ϵ, we discard p. Otherwise, we add p to S.

Algorithm 1 Pseudocode for ROA (2D random order algorithm).

```
S ← {}
when p ∈ P is received do:
    if dist(p, C(S)) ≤ ε then:
        // Discard p
    else:
        S ← S ∪ {p}
        for p′ ∈ S sequentially do:
            If p′ is an interior point of S then S ← S \ {p′}
        end for
end when
```

▶ **Lemma 4.2.** *There exists a constant $c > 0$ such that for any random order input stream P containing at most n points, ROA-insertion maintains a subset $S \subseteq P$ which is an ϵ-hull of P at all times. Moreover, if $P \subseteq \mathbb{R}^1$ then with probability at least $1 - 1/n^3$,*

$$|S| \leq c \cdot \log n \leq c \cdot \mathit{OPT}(P, \epsilon) \cdot \log n,$$

note that for any $\epsilon \geq 0$, if $P \subseteq \mathbb{R}^1$ then $1 \leq \mathit{OPT}(P, \epsilon) \leq 2$.

A natural question is whether the space bound for this algorithm generalizes to higher dimensions. Our experiments suggest that it does not even generalize to 2D. In our experiments, we set $\epsilon = 0$ and gave ROA-insertion n equally spaced points inside a square. OPT is 4, since all the points are contained inside a square. However, experimentally, the number of points kept by ROA-insertion increases much faster than $\log n$.

4.2 2D Algorithm

We extend algorithm ROA-Insertion to get algorithm **ROA**. Let the points kept by ROA at the i^{th} step of the algorithm be S_i. At each step i, we iteratively delete interior points from S_i until S_i has no interior points. We summarize algorithm ROA in Algorithm 1.

The proof of ROA gives an interesting connection between our 2D algorithm, and the 1D classical result in the previous section. We begin with a technical lemma (see Lemma B.1 in the appendix of the full version for the proof), and then proceed to the main theorem.

▶ **Lemma 4.3.** *(Similar Boundaries) Suppose A and B are ϵ-hulls of P. Let $\mathcal{H}$ denote the (two-way) Hausdorff distance and $\partial\mathcal{C}(A)$ denote the boundary of the convex hull of A. Then $\mathcal{H}(\partial\mathcal{C}(A), \partial\mathcal{C}(B)) \leq \epsilon$.*

▶ **Theorem 4.4.** *There exists a constant $c > 0$ such that for any random order input stream P containing at most n points, ROA maintains a subset $S \subseteq P$ which is an ϵ-hull of P at all times. Moreover, if $P \subseteq \mathbb{R}^2$ then with probability at least $1 - 1/n^2$,*

$$|S| \leq c \cdot \mathit{OPT}(P, \epsilon) \cdot \log n.$$

Since the algorithm is deterministic, the probability is over the arrival order of P.

Proof. An inductive argument shows that at each iteration i, S is an ϵ-hull of P. We focus on the proof of the space bound.

Step 1: Consider an optimal ϵ-hull T of P. We show that all points in S are near the boundary $\partial\mathcal{C}(T)$. Note that S does not contain any interior points, so for all $s \in S$, $s \in \partial\mathcal{C}(S)$. Then by Lemma 4.3, for every point $s \in S$, $\text{dist}(s, \partial\mathcal{C}(T)) \leq \epsilon$.

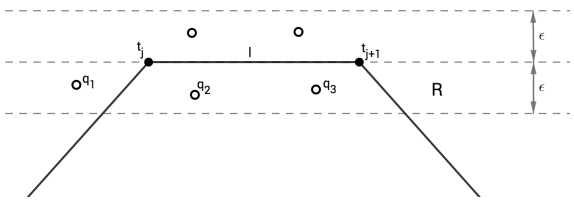


Figure 2 Figure for Theorem 4.4. Consider all points near segment $l = \overline{t_j t_{j+1}}$. Consider points $q_1, q_2, q_3 \in Q$ on one side of l. They are contained in a thin strip R of width ϵ.

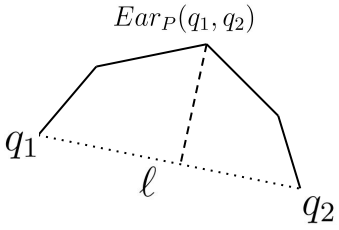


Figure 3 A diagram of $Ear_P(q_1, q_2)$. The dotted line is ℓ, and the length of the dashed line is $\mathsf{Error}_P(q_1, q_2)$.

Step 2: We split T into OPT sections, and show that with high probability our algorithm keeps $O(\log n)$ points for each section. Since T is optimal, it does not contain any interior points. Label the points in T: t_1, ..., t_k, clockwise along the boundary of the convex hull of T. For every $s \in S$, since $\text{dist}(s, \partial\mathcal{C}(T)) \leq \epsilon$, s is within distance ϵ from the line segment connecting some t_i and t_{i+1}. Now, referring to Figure 2, consider the line segment l connecting arbitrary t_j and t_{j+1}, and consider all points in P within distance ϵ from l. We group the points based on which side of the line segment they are on - consider the points Q on one side of the line segment.

The points Q are contained in some narrow strip R with width ϵ. Effectively, because Q is contained in a narrow strip, we can reduce to the 1D case and apply the proof from Lemma 4.2. To see this reduction, consider the projection of the points in Q onto the (infinite) line l' connecting t_j and t_{j+1}. Let $f(q_i)$ denote the projection of q_i onto line l'. If $f(q_k)$ is between $f(q_i)$ and $f(q_j)$, and q_k arrives after q_i and q_j, then our algorithm discards q_k because q_k is within distance ϵ from the line segment connecting q_i and q_j. Applying the 1D proof, we get that with high probability we keep $O(\log n)$ points for each segment.

Step 3: We take a union bound over the OPT sections to get the desired result, where we note that $\mathsf{OPT} \leq n$. ◀

5 2D Multipass Algorithm

In this section, we relax the problem by letting the algorithm pass over the stream P multiple times. Let $\text{diam}(P)$ refer to the diameter of the point-set P which is $\max_{x,y \in P} d(x, y)$. Our algorithm requires $\log(\frac{\text{diam}(P)}{\epsilon})$ passes and $O(\mathsf{OPT})$ memory. For convenience of exposition, we assume $\text{diam}(P) = 1$ and prove a bound of $\log(\frac{1}{\epsilon})$ passes. If $\text{diam}(P) \neq 1$, we can simply scale all the points, and ϵ, by $\frac{1}{\text{diam}(P)}$ and run the algorithm to get the desired bound.

By convention, we define the distance between a point p and a set A to be $d(p, A) = \min_{a \in A} d(p, a)$. In a slight abuse of notation, for a finite set P we define ∂P to be the subset of P that lies on the boundary of the convex hull of P. Formally:

▶ **Definition 5.1.** For a finite set $P \subset \mathbb{R}^2$, we define $\partial P = P \cap \partial\mathcal{C}(P)$. Here $\partial\mathcal{C}(P)$ means the boundary of the convex hull of P.

Given any two points $q_1, q_2 \in \partial P$, define $\ell = \mathcal{C}(\{q_1, q_2\})$ to be the line segment with endpoints q_1 and q_2. Observe that the set $\mathcal{C}(P) \setminus \ell$ has at most two connected components. Define $Ear_P(q_1, q_2)$ to be the component that lies to the left of the vector from q_1 to q_2. We define $\mathsf{Error}_P(q_1, q_2) = \max_{x \in Ear_P(q_1, q_2)} d(x, \ell)$ to be the maximum distance of a point in this

Algorithm 2 Input: a stream of points $P \subset \mathbb{R}^2$ and a value $\epsilon \in (0, 1]$. Output: an ϵ-approximate hull of P

1: $t_1 \leftarrow (1, 0)$, $t_2 \leftarrow (-1, 0)$
2: $T \leftarrow$ an ordered list (t_1, t_2)
3: For $i = \{1, 2\}$, associate $q_i \leftarrow \mathsf{GetMax}_P(t_i)$ with t_i
4: Initialize Flag to down position
5: **for all** $1 \leq i \leq |T|$ (in parallel) **do**
6: Compute $\mathsf{Error}_P(q_i, q_{i+1})$
7: Compute $\mathsf{Error}_P(q_{i-1}, q_{i+1})$
8: $t'_i \leftarrow$ direction bisecting t_i and t_{i+1}
9: $q'_i \leftarrow \mathsf{GetMax}(t'_i)$
10: **for all** $1 \leq i \leq |T|$ (in parallel) **do**
11: **if** $\mathsf{Error}_P(q_{i-1}, q_{i+1}) \leq \epsilon$ and neither t_{i+1} or t_{i-1} have been deleted **then**
12: Remove t_i from T
13: **if** $\mathsf{Error}_P(q_i, q_{i+1}) > \epsilon$ **then**
14: Add t'_i to T and associate q'_i with t'_i
15: Raise Flag
16: Recompute indices of T to preserve clockwise-order
17: Delete any points/vectors except $t_i \in T$ and their associated q_i
18: **if** Flag is up **then**
19: Go to Line 4
20: **else**
21: Output $\{q_1, \ldots, q_{|T|}\}$

component from ℓ. See Figure 3 for an example. Note that we can compute $\mathsf{Error}_P(q_1, q_2)$ in a single pass (see Algorithm 2 and Lemma C.1 in the Appendix of the full version).

Let t be a unit vector. Define $\mathsf{GetMax}_P(t)$ to be $\arg\max_{p \in P} p \cdot t$. It is clear that $\mathsf{GetMax}_P(t)$ can be computed in a single pass. Algorithm 2 is the main multipass algorithm, using Error and GetMax as blackboxes. We always maintain a set of directions T. On Lines 5-9 we run $3|T|$ single-pass algorithms completely in parallel, therefore requiring only a single pass. By the phrase "associating a point with a direction", we mean to keep this point as piece of satellite data.

Our main result for this section is the behavior of Algorithm 2. We define a word as the space required to store a single point in $\mathbb{R}^2$.

We begin with some preliminary statements. We defer the proofs of these lemmas to the Appendix of the full version (see Lemmas C.2, C.3, C.4, and C.6). Throughout this section, we use the convention of incrementing subscripts modulo n (for example $q_{n+1} = q_1$).

▶ **Lemma 5.2.** *If Algorithm 2 terminates, it outputs an ϵ-hull to P.*

▶ **Lemma 5.3.** *Algorithm 2 terminates in $3 + \lceil \log_2(1/\epsilon) \rceil$ passes.*

▶ **Lemma 5.4.** *Let $p, p', q', q \in \partial P$ be in clockwise order along $\partial \mathcal{C}(P)$. Then $\mathsf{Error}_P(p', q') \leq \mathsf{Error}_P(p, q)$.*

▶ **Lemma 5.5.** *There exists an ϵ-hull for P using only points from ∂P of cardinality at most $2\mathsf{OPT}(P, \epsilon)$.*

Note that Lemma 5.5 is not trivial. By definition, there exists an ϵ-hull for P of size $\mathsf{OPT}(P, \epsilon)$ using points from P. It may be that an ϵ-hull of optimal size must use a point from the interior of P. For example, consider a square of side length $r \in (\sqrt{2}\epsilon, 2\epsilon)$, where

∂P consists of the four corners. It is possible, due to interior points, that $\mathsf{OPT}(P,\epsilon) = 2$ and yet an ϵ-hull using only points from ∂P must use all four corners. Note that this example also shows that the bound in Lemma 5.5 is tight.

On Line 8, t_i' is defined as the direction that bisects t_i and t_{i+1}. We define the bisection of unit vectors a and b to be the unit vector obtained by rotating a clockwise through through half of the rotation required to point in the direction of b.

▶ **Theorem 5.6.** *Given a stream of points $P \subset \mathbb{R}^2$ and a value $\epsilon \in (0,1]$, Algorithm 2 terminates within $3 + \lceil \log_2(1/\epsilon) \rceil$ passes, stores at most $24\mathit{OPT}(P,\epsilon) + O(1)$ words, and returns an ϵ-hull of P of cardinality $6\mathit{OPT}(P,\epsilon)$.*

Proof. By Lemma 5.3, Algorithm 2 terminates after $3 + \lceil \log_2(1/\epsilon) \rceil$ passes. By Lemma 5.2, Algorithm 2 outputs an ϵ-hull to P. It only remains to bound the space usage and cardinality of the set returned

Let $W \subset \partial P$ be an ϵ-approximation of P such that $n = |W| \leq 2\mathsf{OPT}(P,\epsilon)$. Lemma 5.5 guarantees that such a W exists. Let $W = \{w_1, \ldots, w_n\}$ be an ordering of W that is clockwise in $\partial\mathcal{C}(P)$. By definition, $\mathsf{Error}_P(w_i, w_{i+1}) \leq \epsilon$ for every $i \in \mathbb{Z}$ (recall the convention of indexing modulo n). Consider the state of the algorithm at the beginning of a pass; for notation let T contain the directions $\{t_i\}_{i=1}^{|T|}$ associated respectively with $\{q_i\}_{i=1}^{|T|}$.

For $s \in \{1,2\}$, suppose that $w_i, q_j, q_{j+s}, w_{i+1}$ are in clockwise order of $\partial\mathcal{C}(P)$. By Lemma 5.4, $\mathsf{Error}_P(q_j, q_{j+s}) \leq \mathsf{Error}_P(w_i, w_{i+1}) \leq \epsilon$. We draw two conclusions. The first conclusion ($s = 1$) is that on Line 13, t_i' will not be added to T. The second conclusion ($s = 2$) is that on Line 11, t_{i+1} is a candidate for deletion (i.e. t_{i+1} will be deleted unless t_i or t_{i+2} have already been deleted).

Using the clockwise ordering of $\partial\mathcal{C}(P)$, we say that a point $q \in \partial P$ is on edge (w_i, w_{i+1}) if it lies between w_i and w_{i+1} in the ordering. Suppose that $\{q_j\}_{j=1}^{|T|}$ contains m points on edge (w_i, w_{i+1}). By the reasoning in the preceding paragraph, it is easy to verify that all but $\lceil \frac{m-1}{2} \rceil + 1$ will be deleted on Line 11. As for points added on Line 13, this can only occur at the boundary (between q_j and q_{j+1} where q_j is the last point on some edge) and therefore adds at most 1 point per edge.

Combining these facts, we see that an edge which enters a pass with m points finishes that pass with at most $\lceil \frac{m-1}{2} \rceil + 2$ points. Inductively we begin with $m = \{0,1,2\}$ for each edge. This implies that $m \leq 3$ after each pass. Therefore $|T| \leq 3n \leq 6\mathsf{OPT}(P,\epsilon)$ at all times.

Finally, observe that the storage of $4|T| + O(1)$ points are used in a pass. To compute Error without precision issues, storing a single point suffices. Therefore for each i we store one point for each of the two Error computations, one point for GetMax, and the original point q_i and vector t_i. The $O(1)$ is just a workspace to carry out the calculations. ◀

6 (ϵ, δ)-Hull

In this section we give an algorithm for a relaxation of ϵ-hulls, which we call (ϵ,δ)-hulls. Our results hold for arbitrary point sets $P \subseteq \mathbb{R}^d$. Intuitively, an (ϵ,δ)-hull of P is within distance ϵ from the boundary of the convex hull of P in at least a $1-\delta$ fraction of directions. We begin by building up the definition of an (ϵ,δ)-hull.

▶ **Definition 6.1.** Given a vector $v \in \mathbb{R}^d$ and a finite point set $P \subseteq \mathbb{R}^d$, we define the **directional extent** as

$$\omega_v(P) = \max_{p \in P} p \cdot v.$$

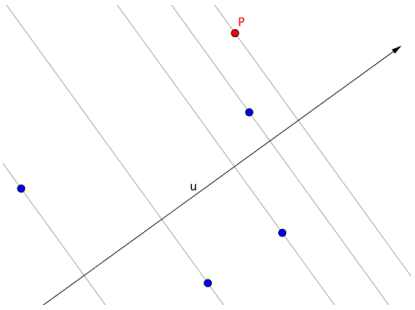


Figure 4 Point p maximizes the set of points in direction u because its projection onto u is the highest.

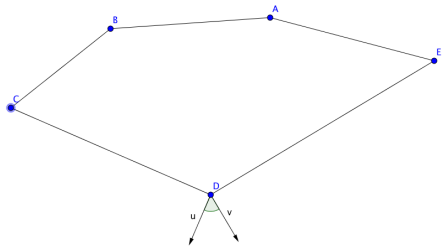


Figure 5 All vectors between u and v with Euclidean norm at most 1 are in E_D^T. See texts for details.

If $p \in \mathbb{R}^d$ is a point we define $\omega_v(p) = p \cdot v = \omega_v(\{p\})$. We say that S **maximizes** P in v if $\omega_v(P) = \omega_v(S)$ (see Figure 4). Note that S can be either a single vector or a set of vectors.

▶ **Definition 6.2.** Let $P \subset \mathbb{R}^d$ be a set of points and $S \subseteq P$. We say S **ϵ-maximizes** P in v if $v = 0$ or, letting $v' = v/\|v\|_2$, we have

$$|\omega_{v'}(P) - \omega_{v'}(S)| \leq \epsilon.$$

Note that as per definition 6.1, S can be either a single vector or a set of vectors.

▶ **Definition 6.3.** Given $P \subseteq \mathbb{R}^d$, an **(ϵ, δ)-hull** is a subset $S \subseteq P$ such that if we pick a vector v uniformly at random from the boundary of the unit sphere, $\mathcal{S}^{d-1}$, S ϵ-maximizes P in direction v with probability at least $1 - \delta$, that is,

$$\Pr_{v \sim \mathcal{S}^{d-1}}(|\omega_v(P) - \omega_v(S)| > \epsilon) \leq \delta.$$

Suppose we fix the dimension d. We give a randomized algorithm that uses m points and with probability at least $1 - \gamma$ gives us an (ϵ, δ)-hull of a point set P, where k is the optimal size for the ϵ-hull of P, and m satisfies:

$$m \in O_d\left(\frac{k}{\delta^2} \cdot \log \frac{k}{\gamma\delta}\right).$$

Note that m does not explicitly depend on ϵ. Our algorithm for d-dimensional space is as follows: Choose m uniformly random vectors in the unit ball B^d (or equivalently on the boundary of the unit ball, $\mathcal{S}^{d-1}$). For each chosen vector v we store a single point $p \in P$ that maximizes P in direction v, that is, $p \cdot v = \omega_v(P)$. This can easily be done in streaming. Note that the given complexity is for a fixed dimension d, the actual space complexity will be multiplied by some (exponential) function of d, but independent of ϵ.

6.1 Proof of (ϵ, δ)-hull Algorithm

Consider an arbitrary point set P, and suppose that our algorithm keeps a subset S of P. The core of our proof, leading up to lemma 6.11, shows that for each point that our algorithm picks, S gets closer to an (ϵ, δ)-hull. In particular, we define the set C of bad vectors in B^d, as vectors v s.t. S does not ϵ-maximize P in v. We want to bound the number of points we need to include in S so that C is small. Crucially, we show that C is a union of a small number of convex sets, and does not contain any vectors we selected (recall that our

algorithm selects uniformly random vectors in B^d, and uses these to select certain points in P). Then, we can approximate C with a union of ellipsoids, which has small VC-dimension. This finally allows us to apply the machinery of ϵ-nets to get the desired result.

We begin with some definitions and lemmas.

▶ **Definition 6.4.** Let B^d denote the unit ball in d dimensions. Let $\mathcal{S}^{d-1}$ denote the unit sphere in d dimensions, which is ∂B^d (the boundary of B^d).

▶ **Definition 6.5.** Let $V^d(S)$ denote the d-dimensional volume (Lebesgue measure) of a measurable set S in d-dimensional space.

▶ **Definition 6.6.** Given $T \subseteq \mathbb{R}^d$ and $t \in \mathbb{R}^d$, we define E_t^T to be the set of all vectors $v \in B^d$ such that t maximizes T in v, that is,

$$E_t^T = \{v \mid v \cdot t = \omega_v(T) \wedge |v|_2 \leq 1\}.$$

Figure 5 shows a set of points T. All vectors between u and v with Euclidean norm at most 1, in the range indicated by the angle, are in E_D^T. Note that u is perpendicular to line segment CD and v is perpendicular to line segment DE. Only points $t \in T$ that lie on the boundary of the convex closure of T have non-empty E_t^T.

▶ **Definition 6.7.** Given a point stream P, and a set S, we say *the set of bad vectors* C (with respect to P, S) is the set of vectors v in B^d such that S does not ϵ-maximize P in v. An equivalent definition of (ϵ, δ)-hulls is that $V^d(C)/V^d(B^d) \leq \delta$.

We are now ready to present the following lemmas about the properties of E_t^T.

▶ **Lemma 6.8** (ϵ-Maximization Lemma). *Suppose $P \subseteq \mathbb{R}^d$ is a finite set of points and $T \subseteq P$ is an ϵ-hull of P, and $t \in T$. Then t ϵ-maximizes P for all vectors $v \in E_t^T$ (see Definition 6.6).*

▶ **Lemma 6.9** (Covering Lemma). *For all finite point sets $T \subseteq \mathbb{R}^d$, $\bigcup_{t \in T} E_t^T = B^d$.*

▶ **Lemma 6.10** (Convex Lemma). *For any point $t \in \mathbb{R}^d$ and finite set $T \subseteq \mathbb{R}^d$, E_t^T is convex and has finite volume.*

We want to show that the set of points S our algorithm chooses is ϵ-maximal in most directions. One way is to show that for each point our algorithm picks, the set of *bad vectors* (vectors that our stored points do not ϵ-maximize) shrinks. We present a crucial lemma that formalizes this notion under some assumptions.

▶ **Lemma 6.11.** *Given a finite point set $P \subseteq \mathbb{R}^d$ and a finite-volume convex set $C \subseteq \mathbb{R}^d$. Assume that there exists some $p \in P$ s.t. for all unit vectors $v \in C$, p ϵ-maximizes P in v. Suppose that we pick arbitrary vectors $v_1, ..., v_k \in C$ and corresponding points $p_1, ..., p_k \in P$ s.t. for all i, p_i maximizes P in v_i. Then there exists a finite-volume convex subset $C' \subseteq C$ s.t.*

1. *For all $i \in [k]$, $v_i \notin C'$.*
2. *For all vectors $v \in C \setminus C'$, $S = \{p_1, ..., p_k\}$ ϵ-maximizes P in v.*

Proof. Consider a vector v_i that we picked, and corresponding point p_i. If $p_i = p$, then $C' = \{\}$ satisfies the required conditions. Otherwise, let $H_i = \{v \mid p_i \cdot v \geq p \cdot v\}$. H_i is a half-space that contains the vector v_i. Furthermore for all vectors $v \in H_i \cap C$, S ϵ-maximizes P in v. So the set of vectors in C that p_i does not maximize are contained in $H_i^c \cap C$, where H_i^c does not contain v_i. Applying this argument for each vector v_i and corresponding

point p_i, we can construct C' to be the intersection of C with the k (open) half-spaces H_i^c corresponding to each of the points p_i we selected. Our constructed C' is convex, because it is the intersection of convex sets, and it is bounded and measurable. ◀

For completeness, we include a standard lemma that is similar to the finite ϵ-net in computational geometry. Before we proceed, for a family of sets $\mathcal{H}$, we denote the simplified version of *VC-dimension* $d' = \widetilde{\text{VC}}(\mathcal{H})$ as the smallest positive integer d' such that for every finite set $A \subseteq \mathbb{R}^d$, $|\{h \cap A : h \in \mathcal{H}\}| \leq |A|^{d'}$ (that is, such that a simple variant of Sauer's Lemma holds). We then have the following lemma, which we prove in the appendix.

▶ **Lemma 6.12.** *Let $\tau, \gamma \in (0,1)$ be two parameters. Let $\mathcal{H}$ be a set of measurable sets in $\mathbb{R}^d$ such that $\widetilde{VC}(\mathcal{H}) \leq d'$ for some integer d'. Given a measuable convex set $C \subseteq \mathbb{R}^d$, let $\mathcal{H}_C = \{c \in \mathcal{H} : c \subseteq C\}$ be the sets of subsets of H contained in C. Suppose we choose $m = \Theta(\frac{d'}{\tau^2} \log \frac{d'}{\tau\gamma})$ points uniformly random in C. Then, except with probability γ, **all** sets $u \in \mathcal{H}_C$ with $V^d(u)/V^d(C) \geq \tau$ contains some selected point, where $V^d(u)$ denotes the volume of u.*

Before we present the main theorem, we note that the set of unions of k ellipsoids is of small VC-dimension. The formal proof is presented in the Appendix.

▶ **Lemma 6.13.** *Let $\mathcal{E}$ be the sets of all ellipsoids in $\mathbb{R}^d$. Let $\mathcal{E}^k = \{e_1 \cup e_2 \cup e_3 \ldots \cup e_k : e_i \in \mathcal{E}\}$. Then $\widetilde{VC}(\mathcal{H}) \leq 4kd^2$.*

Now we are ready to present the main theorem in this section.

▶ **Theorem 6.14.** *Let $\gamma, \delta \in (0,1), k \geq 1$ be parameters. Given a point stream P in $\mathbb{R}^d$ and $\epsilon \geq 0$. Suppose $\mathsf{OPT}(P, \epsilon) \leq k$. Then there exists a one-pass streaming algorithm, given P, γ, δ, k, stores a set $S \subseteq P$ of $m = \Theta_d\big(\frac{k}{\delta^2} \log \frac{k}{\gamma\delta}\big)$ points, such that, except with probability γ, S is an (ϵ, δ)-hull of P.*

Proof. To begin the proof, we recall the algorithm. We first pick uniformly at random $m = \Theta\big(\frac{d^{2d+2}k}{\delta^2} \log \frac{kd}{\gamma\delta}\big)$ directions from B^d, the d-dimensional unit ball. When the stream is coming, we maintain the extreme point from P in each direction. The output S is the set of extreme points in each direction.

Intuitively, S is an (ϵ, δ)-hull iff B^d only contains a small region of bad vectors (with respect to P, S). Let T be an optimal ϵ-hull of P, with $|T| = k$. Fix $t \in T$. Consider the set E_t^T. In our proof we will show that with high probability each set E_t^T only contains a small subset of bad vectors, C_t', such that, for all vectors $v \in E_t^T \setminus C_t'$, S ϵ-maximizes P in v. Then we show that $\sum_{t \in T} V^d(C_t') \leq \delta$, which completes the proof.

Suppose the selected random set of vectors is $A \subseteq B^d$. Fix $t \in T$. By Lemma 6.8, $t \in T$ ϵ-maximizes P for all vectors $v \in E_t^T$. Then by Lemma 6.11, there exists a finite-volume convex subset $C_t' \subseteq E_t^T$ such that $C_t' \cap A = \emptyset$ and for all $v \in E_t^T \setminus C_t'$, S ϵ-maximizes v. Next, for each $t \in T$, we select a large ellipsoid u_t contained in C_t' such that $V^d(C_t') \leq V^d(u_t)d^d$. Note that $\cup_{t \in T} C_t'$ is a member of the family $\mathcal{E}^{|T|} = \{h_1 \cup h_2 \cup \ldots \cup h_{|T|} : \forall i,\ h_i \text{ is an ellipsoid}\}$. By Lemma 6.13, $\widetilde{\text{VC}}(\mathcal{E}^{|T|}) \leq 4kd^2$. By Lemma 6.12, since all u_t do not contain any point from A, with probability at least $1 - \gamma$, it must be the case that $V^d(\cup_{t \in T} u_t)/V^d(B^d) \leq \delta/(d^d)$. Since the u_t are disjoint, this means that $V^d(\cup_{t \in T} C_t')/V^d(B^d) \leq \delta$. Furthermore, by Lemma 6.9, $\bigcup_{t \in T} E_t^T = B^d$. Therefore, with probability at least $1 - \gamma$, S ϵ-maximizes all vectors in B^d except for those in $\cup_{t \in T} C_t'$. Thus, S is an (ϵ, δ)-hull except with probability γ. ◀

References

1 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, 2004. doi:10.1145/1008731.1008736.

2 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. *Geometric approximation via coresets*, pages 1–30. University Press, 2005.

3 Pankaj K Agarwal and Hai Yu. A space-optimal data-stream algorithm for coresets in the plane. In *Proceedings of the twenty-third annual symposium on Computational geometry*, pages 1–10. ACM, 2007.

4 Sunil Arya and Timothy M. Chan. Better epsilon-dependencies for offline approximate nearest neighbor search, euclidean minimum spanning trees, and epsilon-kernels. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SOCG'14, pages 416:416–416:425, New York, NY, USA, 2014. ACM. doi:10.1145/2582112.2582161.

5 Jon Louis Bentley, Franco P. Preparata, and Mark G. Faust. Approximation algorithms for convex hulls. *Commun. ACM*, 25(1):64–68, jan 1982. doi:10.1145/358315.358392.

6 David M Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.

7 Avrim Blum, Sariel Har-Peled, and Benjamin Raichel. Sparse approximation via generating point sets. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 548–557, 2016.

8 Timothy M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Computational Geometry*, 35(1):20–35, 2006. doi:10.1016/j.comgeo.2005.10.002.

9 Timothy M. Chan. Dynamic streaming algorithms for epsilon-kernels. In *Proc. 32nd Annu. Sympos. Comput. Geom. (SoCG)*, 2016.

10 Timothy M. Chan. Applications of chebyshev polynomials to low-dimensional computational geometry. In *Proc. 33rd Annu. Sympos. Comput. Geom. (SoCG)*, 2017.

11 John Hershberger and Subhash Suri. Adaptive sampling for geometric problems over data streams. In *Proceedings of the Twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '04, pages 252–262, New York, NY, USA, 2004. ACM. doi:10.1145/1055558.1055595.

12 David M. Mount Sunil Arya, Guilherme D. da Fonseca. Near-optimal epsilon-kernel construction and related problems. In *Proc. 33rd Annu. Sympos. Comput. Geom. (SoCG)*, 2017.

13 Hamid Zarrabi-Zadeh. An almost space-optimal streaming algorithm for coresets in fixed dimensions. *Algorithmica*, 60(1):46–59, 2011. doi:10.1007/s00453-010-9392-2.

Small Bias Requires Large Formulas

Andrej Bogdanov[1]
Department of Computer Science and Engineering and
Institute of Theoretical Computer Science and Communications,
Chinese University of Hong Kong.
andrejb@cse.cuhk.edu.hk

Abstract

A small-biased function is a randomized function whose distribution of truth-tables is small-biased. We demonstrate that known explicit lower bounds on (1) the size of general Boolean formulas, (2) the size of De Morgan formulas, and (3) correlation against small De Morgan formulas apply to small-biased functions. As a consequence, any strongly explicit small-biased generator is subject to the best-known explicit formula lower bounds in all these models.

On the other hand, we give a construction of a small-biased function that is tight with respect to lower bound (1) for the relevant range of parameters. We interpret this construction as a natural-type barrier against substantially stronger lower bounds for general formulas.

2012 ACM Subject Classification Theory of computation → Circuit complexity, Theory of computation → Pseudorandomness and derandomization

Keywords and phrases formula lower bounds, natural proofs, pseudorandomness

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.22

Acknowledgements Part of this work was done at the Simons Institute for the Theory of Computing at UC Berkeley. I thank Yuval Ishai for raising the question of natural proofs against formula size, Arbel Peled and Alon Rosen for insightful discussions, and anonymous referees (especially ICALP referees!) for useful comments on the manuscript.

1 Introduction

Formula size is one of the most thoroughly studied complexity measures of Boolean functions. A formula is a circuit in which every internal gate has fan-out one. The power of formulas depends on the types of gates allowed. In this work we consider two models: *General formulas* in which any gate of some pre-specified fan-in c is allowed, and *De Morgan formulas* that only use NOT gates and AND/OR gates of fan-in two.

Explicit size lower bounds for general formulas were first proved by Nečiporuk [18], who showed that the selector (addressing) function requires general constant fan-in formula size $\Omega(n^2/\log n \log\log n)$ over inputs of size n. Boppana and Sipser [8] applied a variant of this method to obtain an improved lower bound of $\Omega(n^2/\log n)$ for the element distinctness function by a related but different method.

The case of De Morgan formulas had been studied even earlier. Subbotovskaya [25] proved that computing parity on n bits requires formula size $\Omega(n^{3/2})$. Andreev [3] combined the ideas of Nečiporuk and Subbotovskaya to obtain a $n^{5/2-o(1)}$ De Morgan formula size lower bound for an explicit family of functions from $\{0,1\}^n$ to $\{0,1\}$. Following partial improvements (Impagliazzo and Nisan [11], Paterson and Zwick [19]), Håstad [10] showed

[1] Work supported by HK RGC GRF grant no. CUHK14208215.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 22; pp. 22:1–22:12


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

that Andreev's function requires formula size $n^{3-o(1)}$, which is optimal in the exponent.[2] The same lower bound was reproved by Dinur and Meir [9] using different methods.

More recently, Tal gave two lower-order improvements to Håstad's result. First, in [27] he showed that Andreev's function requires De Morgan formulas of size $\Omega(n^3/(\log n)^2 \log\log n)$, which is optimal for this function up to the doubly logarithmic term. Later, in [28] he showed that another function introduced by Komargodski and Raz [13] requires De Morgan formula size $\Omega(n^3/(\log n)(\log\log n)^2)$.

In a related line of works, Komargodski, Raz, and Tal [13, 14, 27] study correlation lower bounds against small formulas. For every $k \leq n^{1/3}$, they construct two variants of an explicit function that has correlation at most 2^{-k} with any De Morgan formula of size $n^3/(\log n)^{O(1)}k^2$. Their hard functions make use of error-correcting codes with good list-decodable properties and extractors for bit-fixing sources. Weaker correlation bounds for the parity function were proved by Santhanam [23] and, as observed in [14], also follow implicitly from bounds on the approximate degree of De Morgan formulas [6, 22].

Razborov and Rudich [21] observed that all formula size lower bounds (known at the time) are natural, meaning that the formulas to which the bounds apply cannot compute cryptographically pseudorandom functions. On the other hand, the class NC^1 of polynomial-size logarithmic-depth bounded fan-in circuit families, which are equivalent in power to polynomial-size formula families, is believed to contain pseudorandom functions. Naor and Reingold [16,17] and Banerjee, Peikert, and Rosen [5] proposed such candidate families based on the Decisional Diffie Hellman, hardness of factoring, and Learning With Errors hardness assumptions, respectively. These constructions suggest that explicit size n^C lower bounds for formulas is out of reach for current techniques for sufficiently large values of the exponent C. The values of C in these constructions (for the requisite levels of hardness) are apparently rather large, so they are unlikely to explain the perceived barriers of n^2 and n^3 for general and De Morgan formula size, respectively.

Our results

Our main conceptual contribution is the realization that all known formula size lower bound techniques also apply to small-biased functions. A randomized function is (K, ε)-biased if the induced distribution over truth-tables is a (K, ε)-biased distribution (it satisfies (1) below).

From the perspective of natural proofs, the known properties that distinguish small formulas from random functions are local in the sense that they only make a bounded number of non-adaptive queries to the function.[3] It is therefore reasonable to expect that the largeness condition of the relevant natural properties should continue to hold for random functions that only exhibit bounded independence. We show that these properties, in fact, merely require small bias [15], which is closely related to *approximate* bounded independence. As a direct consequence, we show that the best-known explicit formula lower bounds hold against any implicitly specified small-biased generator (the precise definition is given below).

▶ **Theorem 1.** *Any small-biased generator* $SB_{n,2^{-15n}} : \{0,1\}^{O(n)} \to \{0,1\}$

1. *requires fan-in c formulas of size* $\Omega(n^2/2^c \log n)$,
2. *requires De Morgan formulas of size* $\Omega(n^3/\log n\ (\log\log n)^2)$,
3. *has correlation at most* $2^{-\Omega(k)}$ *with De Morgan formulas of size at most* $n^3/(\log n)^{O(1)}k^2$ *for any* k *such that* $\omega(\log n) \leq k \leq n$.

[2] Our discussion of formula lower bounds is based on Chapter 6 of Jukna's book [12].
[3] The correlation lower bounds [13, 14] in fact apply adaptive queries of a restricted type.

Together with the existence of strongly explicit small-biased generators (see definition and discussion below), Theorem 1 reproves the best-known formula lower bounds in a unified manner and even gives a minor improvement in one case. Item 1 matches the explicit formula size lower bound of Nečiporuk. Item 2 matches the lower bound of Tal [28]. Item 3 is a minor improvement over the lower bound of Tal [27]. His proof requires the additional assumption $k \leq n^{1/3}$.

Like previous formula size lower bounds with the exception of [9], the proof of Theorem 1 relies on shrinkage. It is therefore not surprising that it merely matches but fails to improve the state of the art in explicit lower bounds. The value of Theorem 1 is in explaining hardness against formulas by a single natural property, namely small bias. In contrast, shrinkage proofs are tailored to the model in question. The proofs of [10, 18, 28] rely on shrinkage of a random restriction, the one of [8] on simultaneous shrinkage of multiple restrictions, while [13, 14, 27] use high probability shrinkage. While the role of small bias in the shrinkage arguments is more or less self-evident in certain proofs (Propositions 1 and 3), it is less obvious properties of small bias (Lemmas 2 and 3) that enables the others (Propositions 2, 4, and 5).

From a wider perspective, the utility of circuit lower bounds extends far beyond separating complexity classes, which is merely a motivating purpose. It is just as important to identify which natural (both in the common and technical sense) properties of functions make them intractable in specific computational models. In this sense Theorem 1 provides a new criterion for pseudorandomness of a cryptographic function against a restricted class of distinguishers.

On the other hand, in Theorem 6 we construct a (K, ε)-biased function F with fan-in two formula size $O(n(\log K)^2(\log 1/\varepsilon))$. For $\varepsilon = 2^{-2K}$, this is a $(K, 2^{-K})$-wise independent function of formula size $O(nK(\log K)^2)$, which matches our lower bounds for general formulas in Propositions 1 and 2 up to terms polylogarithmic in K.

In the parameter regimes that yield lower bounds 1 and 2 in Theorem 1, the function F has formula size $O((n \log n)^2)$ and De Morgan formula size $O(n^4(\log n)^3)$. We view this as a barrier to proving super-quadratic lower bounds for general formulas, and super-quartic ones for De Morgan formulas.

In the notation of Razborov and Rudich, our barriers are $\oplus$-natural, where $\oplus$ is the class of parity functions. However, they are not quasipolynomial-size-natural since our function F is not cryptographically pseudorandom: In addition to having small formula size, F is computable by polynomial-size, depth 3 circuit families with AND, OR, and PARITY gates (the class $\mathrm{AC}^0[\oplus]$), which is known not to contain cryptographic pseudorandom functions [20, 21, 24]. It remains open whether our bounds can be matched (or even improved in the case of De Morgan formulas) by a different construction that is plausibly secure with respect to all subexponential-size circuits, of which linear tests are a very special case.

Theorems 1 and 6 suggest that small-biased functions should be studied as suitable candidates for formula size lower bounds. In the extreme setting of parameters $K = 2^n$, $\varepsilon = 2^{-\Theta(n)}$, known constructions of small-biased functions have seed lengths linear in n and may be plausible candidates for improved formula size lower bounds. In this regime, the general and De Morgan formula sizes of F in Theorem 6 are as large as $\tilde{\Theta}(n^4)$. Do there exist, say, $(2^n, 2^{-100n})$-biased functions of smaller formula size?

Bounded independence and small bias

We will call a randomized function $F\colon \{0,1\}^n \to \{0,1\}$ (k, ε)-*wise independent* (in qualitative terms, *almost locally independent*) if for any k distinct inputs $x_1, \ldots, x_k$, the distribution

$(F(x_1), \ldots, F(x_k))$ is within statistical distance ε of the uniform distribution over $\{0,1\}^k$. A random function $F\colon \{0,1\}^n \to \{0,1\}$ is (K,ε)-*biased* (*locally small-biased*) if for any nonempty set X of at most K distinct inputs,

$$\left|\mathbb{E}\left[\prod_{x\in X}(-1)^{F(x)}\right]\right| \leq \varepsilon. \tag{1}$$

When $K = 2^n$ the family is called ε-*biased* (*small-biased*). Small bias implies bounded independence by the following claim [15, Corollary 2.1].

▶ **Claim 1.** *Every (K,ε)-biased function is $(K, 2^{K/2}\varepsilon)$-wise independent.*

Small-biased generators

A family of functions $SB_{n,\varepsilon}\colon \{0,1\}^{s(n,\varepsilon)} \times \{0,1\}^n \to \{0,1\}$ is a *small-biased generator* if the random function $F_r(x) = SB_{n,\varepsilon}(r,x)$ (with r uniformly random) is ε-biased for all n and ε. If we view $SB_{n,\varepsilon}$ as a function from $\{0,1\}^{s(n,\varepsilon)}$ to the set of truth-tables of functions $\{0,1\}^n \to \{0,1\}$, we recover the usual representation of a pseudorandom generator as a function of its seed.

The generator is *strongly explicit* if $s(n,\varepsilon) = O(n + \log 1/\varepsilon)$ and $SB_{n,\varepsilon}$ is uniformly polynomial-time computable. Known constructions of small-biased sets [1, 2, 7, 15, 26] are strongly explicit.

2 Small bias requires large formulas

We are aware of two techniques for proving general formula size lower bounds, the one of Nečiporuk [18] and the variant due to Boppana and Sipser [8]. We show that both imply lower bounds on the formula size of almost locally independent functions. While the second technique yields a stronger lower bound, we find the first one instructive as the role of almost-independence is more transparent.

In the case of De Morgan formulas, we study three proof techniques. The first one, based on average-case shrinkage, underlies the lower bound of Andreev including improvements by Impagliazzo and Nisan, Paterson and Zwick, Håstad, and Tal. We show that this method also bounds the formula size of almost independent functions.

The second method for De Morgan formula lower bounds is due to Tal, who applies a correlation-to-computation reduction in addition to bounds on average-case shrinkage. The third method, due to Komargodski and Raz and improvements by these authors and Tal, applies a high-probability shrinkage lemma to derive strong correlation lower bounds. We show that these two methods give lower bounds on the size of small-biased functions.

Arbitrary formulas

A *restriction* $f|_\rho$ of a function f under a partial assignment ρ of its inputs is the function on the unassigned inputs obtained by fixing all the assigned variables to their values. A random $\overline{k}$-restriction of f is the distribution of restrictions of f under a uniform random assignment that leaves exactly k inputs unassigned.

The *size* of a formula is the number of leaves in the formula tree, namely the number of variables occurring in the formula. The following shrinkage property of formulas follows immediately from linearity of expectation:

▶ **Claim 2.** *Assume $f\colon \{0,1\}^n \to \{0,1\}$ has formula size s. Then the expected formula size of a random $\overline{k}$-restriction of f is at most $(k/n)\cdot s$.*

We say a random function F has formula size at most s if every function in the support of F can be computed by a formula of size at most s.

▶ **Proposition 1.** *Assuming $c \leq \log\log k$, any $(2^k, 1/4)$-wise independent function $F\colon \{0,1\}^n \to \{0,1\}$ requires fan-in c formulas of size $\Omega(n \cdot 2^k/k \log k)$.*

Proof. Suppose F has formula size s. By Claim 2 and averaging, there exists a partial assignment ρ with k unassigned variables under which the expected formula size of $F|_\rho$ is at most ks/n. By Markov's inequality,

$$\Pr_F[\text{size}(F|_\rho) \leq 2ks/n] \geq \frac{1}{2} \tag{2}$$

for any distribution of functions F, where size denotes formula size.

A formula of size $\tilde{s}$ can be specified by listing its at most $2\tilde{s}$ gates in depth-first order. For a formula of fan-in c on k inputs, there are 2^{2^c} possible internal gates and k possible input gates, so the number of such formulas is at most $(2^{2^c} + k)^{2\tilde{s}} \leq (2k)^{2\tilde{s}}$. Therefore, setting $\tilde{s} = 2ks/n$, for a uniformly random function R it holds that

$$\Pr_R[\text{size}(R|_\rho) \leq 2ks/n] \leq \frac{(2k)^{4ks/n}}{2^{2^k}}. \tag{3}$$

The event "$\text{size}(F|_\rho) \leq 2ks/n$" depends on at most 2^k values of F, so if F is $(2^k, 1/4)$-wise independent, then

$$\Pr_F[\text{size}(F|_\rho) \leq 2ks/n] \leq \Pr_R[\text{size}(R|_\rho) \leq 2ks/n] + \frac{1}{4}. \tag{4}$$

Combining (2), (3), and (4), we obtain that $(2k)^{4ks/n}/2^{2^k} \geq 1/4$, from where the desired lower bound on s follows. ◀

An improved lower bound

We now discuss the other proof of Nečiporuk, which gives a slightly stronger lower bound in the regime of $k < \log n$ and for exponentially small error.

▶ **Proposition 2.** *For $k \leq \log n - 1$, any $(2 \cdot 2^k, 2^{-2^k})$-wise independent function $F\colon \{0,1\}^n \to \{0,1\}$ requires fan-in c formulas of size $\Omega(n \cdot 2^{k-c}/k)$.*

The proposition is proved by showing that the number of possible restrictions of a small formula that leave the least frequently occurring inputs unrestricted is small. On the other hand, the following lemma shows that the number of distinct restrictions of an almost locally independent function is large, even when the set of unrestricted variables is fixed. A $\overline{U}$-restriction is a restriction under any assignment in which U is the set of free variables.

▶ **Lemma 2.** *Assume F is $(2 \cdot 2^k, 2^{-2^k})$-wise independent. For any set U of k variables, the number of distinct $\overline{U}$-restrictions of F is at least $\min\{2^{n-k-2}, 2^{2^k-3}\}$ with probability more than half.*

In particular, when $k \leq \log n - 1$, a $(2 \cdot 2^k, 2^{-2^k})$-wise independent function family has at least $\frac{1}{8} \cdot 2^{2^k}$ distinct $\overline{U}$-restrictions with probability more than half.

Proof. Let ρ, ρ' be independent random partial assignments to the variables in $\overline{U}$. Then

$$\Pr_{F,\rho,\rho'}[F|_\rho = F|_{\rho'}] \leq \Pr[\rho = \rho'] + \Pr[F|_\rho = F|_{\rho'} \mid \rho \neq \rho']. \tag{5}$$

The first term equals 2^{-n+k}. To bound the second term, fix an arbitrary pair of distinct ρ, ρ'. The event that the restricted functions $F|_\rho$ and $F|_{\rho'}$ are identical depends on at most $2 \cdot 2^k$ values of F. By the almost local independence of F,

$$\Pr_F[F|_\rho = F|_{\rho'} \mid \rho \neq \rho'] \leq \Pr[R = R'] + 2^{-2^k},$$

where $R, R' \colon \{0,1\}^k \to \{0,1\}$ are independent uniformly random functions. Such functions are equal with probability at most 2^{-2^k}, and so the second term in (5) at most 2^{-2^k+1}. Therefore

$$\Pr_{F,\rho,\rho'}[F|_\rho = F|_{\rho'}] \leq 2^{-n+k} + 2^{-2^k+1}.$$

Now assume the support size of $F|_\rho$ over random ρ is less than S for at least half the functions F. Then the collision probability $\Pr_{\rho,\rho'}[F|_\rho = F|_{\rho'}]$ is at least $1/S$ for at least half the functions F and so

$$2^{-n+k} + 2^{-2^k+1} \geq \frac{1}{2S},$$

from where it follows that the larger of 2^{-n+k} and 2^{-2^k+1} is at least $1/4S$. It follows that $S \geq \min\{2^{n-k-2}, 2^{2^k-3}\}$. ◀

Proof of Proposition 2. Let s be the size of F. By Claim 2 and averaging, there is a set U of size k so that on average, F has at most $(k/n) \cdot s$ occurrences of variables from U. By Markov's inequality, at least half of the formulas in F have no more than $\tilde{s} = 2ks/n$ occurrences of variables from U.

We now upper bound the number of $\overline{U}$-restrictions of ϕ (for fixed ϕ and U). Under each partial assignment ρ to this inputs in $\overline{U}$, ϕ reduces to a formula $\phi|_\rho$ of size at most $\tilde{s}$. This formula can be simplified by propagating the restricted inputs and subsuming all gates of fan-in one into their parents or children in some canonical way. The simplified formula can then be described by specifying, say in depth first order, the truth-tables of its gates (of fan-in at least two). As there are at most $\tilde{s}$ such gates and each can compute one of at most 2^{2^c} possible functions, the desired number of restrictions can be at most $2^{2^c\tilde{s}}$.

By Lemma 2, there must then exist a formula in the support of F whose number of $\overline{U}$-restrictions is at most $2^{2^c\tilde{s}} = 2^{2^{c+1}ks/n}$ and at least $\frac{1}{8} \cdot 2^{2^k}$. It follows that $s = \Omega(n \cdot 2^{k-c}/k)$. ◀

Computation by De Morgan formulas

In this section we show that known proofs for De Morgan formula size also apply to small-biased functions. The following proof relies on expected shrinkage of De Morgan formulas under random restrictions [3, 10, 11, 19].

▶ **Proposition 3.** *Assuming $k \leq n/2$, any $(2^k, 1/4)$-wise independent function $F \colon \{0,1\}^n \to \{0,1\}$ requires De Morgan formula size $\Omega(n^2 \cdot 2^k/k^2 \log k)$.*

Proof. In a $\overline{p}$-random restriction, the unrestricted variables are sampled from the binomial distribution with parameter p. Tal [27] showed that if f has a De Morgan formula of size s then the expected formula size of a $\overline{p}$-random restriction of f is $\tilde{s} = O(p^2 s + \sqrt{p^2 s})$. Set $p = 2k/n$. By deviation bounds, for every f in the support of F, the event that ρ has fewer than $k = \frac{1}{2}pn$ unassigned inputs or $f|_\rho$ has formula size more than $4\tilde{s}$ has probability at most $\frac{1}{2}$.

By averaging, there exists a partial assignment ρ with k unassigned inputs under which $F|_\rho$ has formula size at most $4\tilde{s}$ for at least half the functions F. Since F is $(2^k, 1/4)$-wise

independent, the same is true for at least a quarter of truly random functions R. The number of size $4\tilde{s}$ De Morgan formulas on k inputs is at most $(2k)^{8\tilde{s}}$, and these must compute at least $\frac{1}{4} \cdot 2^{2^k}$ distinct functions. It follows that $\tilde{s} = \Omega(2^k/\log k)$. As $\tilde{s} = O(p^2 s + \sqrt{p^2 s})$ it follows that $p^2 s = \Omega(2^k/\log k)$. Using the constraint $k \geq \frac{1}{2}pn$ we obtain the desired bound. ◀

Tal [27] recently obtained a slight improvement to the aforementioned bounds. His method also applies to small-biased functions as demonstrated in the following proposition:

▶ **Proposition 4.** *Assume that $k \leq n/2$ and $2^{-\frac{7}{16}k2^{k/8}} < \varepsilon \leq 2^{-2k}$. Then every $(2^k, \varepsilon)$-biased F requires De Morgan formula size $\Omega(n^2 \log(1/\varepsilon)/k(\log k)^2)$.*

The proof relies on the large deviation bound for small-bias distributions of Naor and Naor [15, Section 5]. We rework it here in more convenient notation. We say a random variable X over $\{-1,1\}^K$ is ε-biased if $|\mathbb{E}[\prod_{i \in S} X_i]| \leq \varepsilon$ for every subset S of indices.

▶ **Lemma 3.** *Let t be even and X be a (t, ε)-biased random variable over $\{-1,1\}^K$. The probability that $|\sum X_i|$ exceeds δK is at most $\delta^{-t} \cdot (2(t/K)^{t/2} + \varepsilon)$.*

Proof. We apply a t-th moment calculation. By Markov's inequality,

$$\begin{aligned}\Pr\Big[\Big|\sum\nolimits_{i=1}^{K} X_i\Big| \geq \delta K\Big] &\leq \frac{1}{(\delta K)^t}\,\mathbb{E}\Big[\Big(\sum\nolimits_{i=1}^{K} X_i\Big)^t\Big] \\ &= \frac{1}{(\delta K)^t}\Big(\sum\nolimits_{S \in \mathcal{E}} \mathbb{E}\Big[\prod\nolimits_{i \in S} X_i\Big] + \sum\nolimits_{S \in \overline{\mathcal{E}}} \mathbb{E}\Big[\prod\nolimits_{i \in S} X_i\Big]\Big),\end{aligned}$$

where $\mathcal{E}$ is the set of ordered terms of size t in which every index appears an even number of times. The first expectation is upper bounded by the number of such terms, which is at most $K^{t/2} \cdot t!/(t/2)! \leq 2 \cdot (tK)^{t/2}$. The second expectation is upper bounded by the number of terms times the maximum bias of each term, namely $K^t \cdot \varepsilon$. The desired bound follows. ◀

The following consequence of the lemma is far from tight but will be of use in the proof of Proposition 4. The *correlation* of two functions $f, \phi\colon \{0,1\}^k \to \{0,1\}$ is $\langle f, \phi \rangle = \mathbb{E}_x[(-1)^{f(x)} \cdot (-1)^{g(x)}]$, where x is uniform in $\{0,1\}^n$.

▶ **Corollary 4.** *Assuming $2^{-\frac{7}{16}k2^{k/8}} < \varepsilon \leq 4 \cdot 2^{-2k}$ and $F\colon \{0,1\}^k \to \{0,1\}$ is $(2^k, \varepsilon)$-biased, for every $\phi\colon \{0,1\}^k \to \{0,1\}$, the probability that $|\langle F, \phi \rangle|$ is greater than $2^{-k/4}$ is at most $3 \cdot \varepsilon^{1/4}$.*

Proof. Assuming $4 \leq t \leq 2^{k/8}$, the expression $(t \cdot 2^{-k})^{t/2}$ is non-increasing as a function of t. By our assumption on ε, there must exist even value $4 \leq t < 2^{k/8}$ for which

$$\big((t+2)2^{-k}\big)^{(t+2)/2} < \varepsilon \leq (t2^{-k})^{t/2}. \tag{6}$$

Applying Lemma 3 with parameters $K = 2^k$, $\delta = 2^{-k/4}$ to the truth-table X of the (K, ε)-biased function $(-1)^{F(x) \oplus \phi(x)}$, we obtain that the desired probability is at most

$$(2^{-k/4})^t \cdot \big(2(t2^{-k})^{t/2} + \varepsilon\big) = 3 \cdot t^{t/2} \cdot 2^{-kt/4}.$$

To derive the corollary, it remains to show that $t^{t/2} \cdot 2^{-kt/4} \leq ((t+2)2^{-k})^{(t+2)/8}$. This follows from $k \geq (4t \log t)/(t-2)$, which is true in the regime $4 \leq t < 2^{k/8}$. ◀

Proof of Proposition 4. Initially we proceed as in the proof of Proposition 3 to obtain a partial assignment ρ with k unassigned inputs under which $F|_\rho$ has formula size $\tilde{s} = O((k/n)^2 s + \sqrt{(k/n)^2 s})$ for at least half the functions F. Let $\mathcal{S}$ (for shrinkage) denote this event so that $\Pr[\mathcal{S}] \geq \frac{1}{2}$.

Tal [28] showed that every formula of size $\tilde{s}$ has correlation at least $\delta = 2^{-k/4}$ (7) with some formula of size $\tilde{s}' = O(\sqrt{\tilde{s}} + \tilde{s}\log k/k)$. Let Φ be the set of all such formulas over inputs in U. Then we have

$$\mathbb{E}[|\langle F|_\rho, \Phi\rangle|] \geq \mathbb{E}[|\langle F|_\rho, \Phi\rangle| \mid \mathcal{S}] \cdot \Pr[\mathcal{S}] \geq \frac{\delta}{2},$$

where $|\langle f, \Phi\rangle|$ denotes the maximum value of $|\langle f, \phi\rangle|$ over all $\phi \in \Phi$. By Markov's inequality,

$$\Pr[|\langle F|_\rho, \Phi\rangle| \geq \delta/4] \geq \frac{\delta}{4}.$$

On the other hand, by a union bound and Corollary 4,

$$\Pr[|\langle F|_\rho, \Phi\rangle| \geq \delta/4] \leq 3|\Phi|\varepsilon^{1/4}.$$

From these two inequalities we obtain that

$$|\Phi| \geq \frac{1}{12} \cdot \varepsilon^{-1/4} \cdot \delta \geq \frac{\varepsilon^{-1/8}}{12}$$

by (7) and the assumption $\varepsilon \leq 2^{-2k}$. Since $|\Phi| \leq (9k)^{\tilde{s}'}$, it follows that $\tilde{s}' = \Omega(\log(1/\varepsilon)/\log k)$. A calculation shows that $s = \Omega(n^2 \log(1/\varepsilon)/k(\log k)^2)$ as desired. ◀

Correlation with De Morgan formulas

Komargodski, Raz, and Tal [13, 14, 27] proved a correlation lower bound for small De Morgan formulas. Their main technical ingredient is the following high-probability shrinkage lemma for De Morgan formulas [14, 27].

Lemma 5 and the proof of Proposition 5 use the following notation. Given a depth-$(n-k)$ decision tree Δ in n variables and a seed $\pi \in \{0,1\}^{n-k}$, the partial assignment $\Delta(\pi)$ is the one obtained by assigning values to the variables in Δ from root to leaf according to the sequence π (the first bit π_1 is assigned to the root variable, the second bit π_2 is assigned to its child, and so on).

▶ **Lemma 5** (High-probablity shrinkage). *For every constant $c > 0$ there exists a constant $c' > 0$ such that for every $c' \log n \leq k \leq n$ the following holds. For every formula f on n variables of size $s \leq n^c$ there exists a decision tree Δ over its variables of depth at most $n-k$ so that $f|_{\Delta(\pi)}$ has formula size $\tilde{s} = (\log n)^{O(1)} \cdot (k/n)^2 \cdot s$ except with probability $\delta = 2^{-\Omega(k)}$ over the choice of π.*

Without loss of generality we will assume that Δ is a complete decision tree of depth exactly $n-k$ so that $\Delta(\pi)$ has exactly k unrestricted variables for every π.

▶ **Proposition 5.** *Assuming $2^{-\frac{7}{16}k2^{k/8}} \leq \varepsilon \leq 3^{-8n} \cdot 2^{-2k}$ and $\omega(\log n) \leq k \leq n$, for every $(2^k, \varepsilon)$-biased F, at most a $2^{-\Omega(k)}$-fraction of F has correlation more than $2^{-\Omega(k)}$ with formulas of size at most $n^2 \log(1/\varepsilon)/(\log n)^{O(1)}k^2$.*

Proof of Proposition 5. Let $\mathcal{C}$ (for correlating) be the event that F has correlation at least 2δ in absolute value with some formula $\hat{F}$ of size at most s (which may depend on F) so that

$$\mathbb{E}_F[|\langle F, \hat{F}\rangle| \mid \mathcal{C}] \geq 2\delta.$$

We set $\delta = 2^{-\Omega(k)}$ and assume that $\delta \geq 2^{-k/8}$ (8). For every complete decision tree Δ (which may depend on $\hat{F}$) of depth $n-k$,

$$\mathbb{E}_{F,\pi}[|\langle F|_{\Delta(\pi)}, \hat{F}|_{\Delta(\pi)}\rangle| \mid \mathcal{C}] \geq \mathbb{E}_F\left[|\mathbb{E}_\rho[\langle F|_{\Delta(\pi)}, \hat{F}|_{\Delta(\pi)}\rangle]| \;\middle|\; \mathcal{C}\right] = \mathbb{E}_F[|\langle F, \hat{F}\rangle| \mid \mathcal{C}] \geq 2\delta, \quad (9)$$

where $\pi \sim \{0,1\}^{n-k}$ is a random seed. Let $\mathcal{S}$ be the event that $\hat{F}|_{\Delta(\pi)}$ has formula size at most $\tilde{s} = (\log n)^C \cdot (k/n)^2 \cdot s$ (10). By Lemma 5,

$$\Pr_{F,\pi}[\overline{\mathcal{S}} \mid \mathcal{C}] \leq \delta. \tag{11}$$

By the formula for conditional expectations,

$$\begin{aligned}\mathbb{E}[|\langle F|_{\Delta(\pi)}, \hat{F}|_{\Delta(\pi)}\rangle| \mid \mathcal{C}] &= \mathbb{E}[|\langle F|_{\Delta(\pi)}, \hat{F}|_{\Delta(\pi)}\rangle| \mid \mathcal{C}\mathcal{S}] \cdot \Pr[\mathcal{S} \mid \mathcal{C}] \\ &\quad + \mathbb{E}[|\langle F|_{\Delta(\pi)}, \hat{F}|_{\Delta(\pi)}\rangle| \mid \mathcal{C}\overline{\mathcal{S}}] \cdot \Pr[\overline{\mathcal{S}} \mid \mathcal{C}] \\ &\leq \mathbb{E}[|\langle F|_{\Delta(\pi)}, \hat{F}|_{\Delta(\pi)}\rangle| \mid \mathcal{C}\mathcal{S}] + \Pr[\overline{\mathcal{S}} \mid \mathcal{C}],\end{aligned}$$

so (9) and (11) imply that

$$\mathbb{E}_{F,\pi}[|\langle F|_{\Delta(\pi)}, \hat{F}|_{\Delta(\pi)}\rangle| \mid \mathcal{C}\mathcal{S}] \geq \delta.$$

Let Φ be the set of all size-$\tilde{s}$ formulas over k variables. Then $|\Phi| \leq (9k)^{\tilde{s}}$ (12). Since conditioned on $\mathcal{S}$ all formulas $\hat{F}|_{\Delta(\pi)}$ are in Φ, it must be the case that

$$\mathbb{E}_{F,\pi}[|\langle F|_{\Delta(\pi)}, \Phi\rangle| \mid \mathcal{C}\mathcal{S}] \geq \delta,$$

where $\langle f, \Phi\rangle$ denotes the maximum of $\langle f, \phi\rangle$ over all $\phi \in \Phi$. By the formula for conditional expectations, $\mathbb{E}_{F,\pi}[|\langle F|_{\Delta(\pi)}, \Phi\rangle|]$ must be at least $\delta \cdot \Pr[\mathcal{C}\mathcal{S}]$. We can then bound $\Pr[\mathcal{C}\mathcal{S}]$ by

$$\Pr[\mathcal{C}\mathcal{S}] \leq \frac{1}{\delta} \cdot \mathbb{E}_{F,\pi}[|\langle F|_{\Delta(\pi)}, \Phi\rangle|] \leq \frac{1}{\delta}\Big(\frac{\delta^2}{4} + \Pr_{F,\pi}[|\langle F|_{\Delta(\pi)}, \Phi\rangle| \geq \delta^2/4]\Big). \tag{13}$$

Let $\mathcal{P}_k^n$ denote the set of partial assignments that leave k inputs unassigned. As each input can take value 0, take value 1, or be unassigned, $\mathcal{P}_k^n$ has size at most 3^n. By Corollary 4, (8), and a union bound,

$$\begin{aligned}\Pr_{F,\pi}[|\langle F|_{\Delta(\pi)}, \Phi\rangle| \geq \delta^2/4] &\leq \Pr_F[|\langle F|_\rho, \phi\rangle| \geq \delta^2/4 \text{ for some } \rho \in \mathcal{P}_k^n \text{ and } \phi \in \Phi] \\ &\leq 3^n \cdot |\Phi| \cdot 3\varepsilon^{1/4}.\end{aligned}$$

Using (8) and the assumption $\varepsilon \leq 3^{-8n} \cdot 2^{-2k}$, the right hand side is at most $(\delta^2/4) \cdot 12|\Phi|\varepsilon^{1/8}$. By (12) and (10), this quantity is at most $\delta^2/4$ as long as $s \leq n^2 \log(1/\varepsilon)/(\log n)^C k^2$. Plugging into (13), we conclude that $\Pr[\mathcal{C}\mathcal{S}]$ is at most $\delta/2$ for formulas of the desired size.

Finally, applying (11) again, we have

$$\Pr[\mathcal{C}] = \frac{\Pr[\mathcal{C}\mathcal{S}]}{1 - \Pr[\overline{\mathcal{S}} \mid \mathcal{C}]} \leq \frac{\delta/2}{1-\delta} \leq \delta. \qquad \blacktriangleleft$$

3 Main results

Proof of Theorem 1

Let F be the random function $F(x) = SB_{n,2^{-15n}}(s,x)$ for uniformly random s. To obtain item 1, we apply Proposition 2 with $k = \log n - 1$ and Claim 1. (Proposition 1 gives the weaker bound $\Omega(n^2/\log n \log\log n)$ for fan-in up to $c = \log\log\log n$.)

For item 2, we apply Proposition 4 with $k = 3\log n$ and $\varepsilon = n^9 e^{-n}$. (Proposition 3 with $k = \log n$ gives the weaker bound $\Omega(n^3/(\log n)^2 \log\log n)$.)

For item 3, we apply Proposition 5 with $\varepsilon = 3^{-8n} \cdot 2^{-2n}$. The conclusion is that at most a $2^{-\Omega(k)}$-fraction of F can have correlation more than $2^{-\Omega(k)}$ with formulas of size s. Therefore the correlation between $SB_{n,2^{-15n}}$ and size s formulas can be at most $2^{-\Omega(k)}$. $\blacktriangleleft$

Moderate formulas for small bias

▶ **Theorem 6.** *For every n, k, and ε, there exists a $(2^k, \varepsilon)$-biased $F\colon \{0,1\}^n \to \{0,1\}$ of fan-in two formula size $O(nk^2 \cdot \log 1/\varepsilon)$.*

Applying Claim 1 and a suitable change of parameters we obtain the following corollary to Theorem 6:

▶ **Corollary 7.** *For every n, K, and ε there exist (K, ε)-wise independent functions with formula size $O(n \cdot (\log K)^2 \cdot (K + \log 1/\varepsilon))$.*

Proof of Theorem 6. Let $H_t\colon \{0,1\}^n \to \{0,1\}$ be the random function

$$H_t(x) = \begin{cases} \text{a random bit}, & \text{if } Ax = b, \\ 0, & \text{if not}, \end{cases}$$

where A and b are a uniformly random $t \times n$ matrix and t-dimensional boolean vector, respectively, and all algebra is over $\mathbb{F}_2$. We let

$$F = F_1 \oplus F_2 \oplus \cdots \oplus F_{k+2},$$

where the F_t are independent XORs of $6 \log 1/\varepsilon$ independent copies of H_t. Since H_t has formula size $O(tn)$, F has formula size $O(nk^2 \cdot \log 1/\varepsilon)$.

We now prove that F is $(2^k, \varepsilon)$-biased. Let X be any nonempty set of at most 2^k distinct inputs. Set $t = \lfloor \log |X| \rfloor + 2$ and let $\mathcal{U}$ (for unique) be the event that exactly one x in X satisfies $Ax = b$ for a random $t \times n$ matrix A and t-dimensional vector b. By the isolation lemma of Valiant and Vazirani [29], $\mathcal{U}$ has probability at least $1/8$ (see for example [4, Lemma 17.19]). By the rule of conditional expectations,

$$\begin{aligned} &\left|\mathbb{E}\Big[\prod\nolimits_{x\in X}(-1)^{H_t(x)}\Big]\right| \\ &\quad\le \left|\mathbb{E}\Big[\prod\nolimits_{x\in X}(-1)^{H_t(x)}\Big|\mathcal{U}\Big]\right| \cdot \Pr[\mathcal{U}] + \left|\mathbb{E}\Big[\prod\nolimits_{x\in X}(-1)^{H_t(x)}\Big|\overline{\mathcal{U}}\Big]\right| \cdot \Pr[\overline{\mathcal{U}}] \\ &\quad\le |\mathbb{E}[(-1)^{H_t(u)} \mid \mathcal{U}]| \cdot \Pr[\mathcal{U}] + 1 \cdot \Pr[\overline{\mathcal{U}}] \\ &\quad= 0 \cdot \Pr[\mathcal{U}] + 1 \cdot \Pr[\overline{\mathcal{U}}] \\ &\quad\le 7/8. \end{aligned}$$

By independence, it follows that

$$\left|\mathbb{E}\Big[\prod\nolimits_{x\in X}(-1)^{F_t(x)}\Big]\right| = \left|\mathbb{E}\Big[\prod\nolimits_{x\in X}(-1)^{H_t(x)}\Big]\right|^{6\log 1/\varepsilon} \le \left(\frac{7}{8}\right)^{6\log 1/\varepsilon} \le \varepsilon,$$

so $|\mathbb{E}[\prod_{x\in X}(-1)^{F(x)}]| = \prod_{t=1}^{k+2} |\mathbb{E}[\prod_{x\in X}(-1)^{F_t(x)}]|$ is also upper bounded by ε. ◀

Our small-biased function can be viewed as a simplified variant of a construction of Naor and Naor [15, Section 3.1.1]. The simplifications can be partly explained by a difference in objectives: Naor and Naor (and other constructions) aim to optimize the seed length, while we are interested in minimizing formula size.

By the standard simulation of fan-in two formulas by De Morgan formulas, F has De Morgan formula size at most $O((nk^2 \log 1/\varepsilon)^2)$. The De Morgan formula size analysis can be slightly improved to $O(n^2k^3(\log 1/\varepsilon)^2)$ by observing that the middle layer of AND gates does not suffer from the quadratic blow-up.

Specifically, in the parameter settings used in the proof of items 1 and 2 in Theorem 1, the function F has fan-in two formula size $O((n \log n)^2)$ and De Morgan formula size $O(n^4(\log n)^3)$.

For item 3, plugging in $\varepsilon = 2^{-k}$ gives the (De Morgan) formula size upper bound $O(n^4k^3)$. This can be improved to $O(n^4(\log(n/k))^3)$: In the proof of Corollary 4 (and Proposition 5) it is sufficient that F be (t, ε)-biased where t is the unique even integer satisfying (6). For our choice of parameters t is on the order of n/k.

References

1. Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Trans. Information Theory*, 38(2):509–516, 1992. doi:10.1109/18.119713.
2. Noga Alon, Oded Goldreich, Johan Hastad, and René Peralta. Simple construction of almost k-wise independent random variables. *Random Struct. Algorithms*, 3(3):289–304, 1992. doi:10.1002/rsa.3240030308.
3. A. E. Andreev. On a method for obtaining more than quadratic effective lower bounds for the complexity of π-schemes. *Moscow Univ. Math. Bull.*, 42(1):63–66, 1987.
4. Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.
5. Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *EUROCRYPT*, pages 719–737, 2012. doi:10.1007/978-3-642-29011-4_42.
6. Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48(4):778–797, jul 2001. doi:10.1145/502090.502097.
7. Avraham Ben-Aroya and Amnon Ta-Shma. Constructing small-bias sets from algebraic-geometric codes. *Theory of Computing*, 9:253–272, 2013. doi:10.4086/toc.2013.v009a005.
8. Ravi B. Boppana and Michael Sipser. The complexity of finite functions. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 757–804. Elsevier and MIT Press, 1990.
9. Irit Dinur and Or Meir. Toward the KRW composition conjecture: Cubic formula lower bounds via communication complexity. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 3:1–3:51, 2016. doi:10.4230/LIPIcs.CCC.2016.3.
10. Johan Håstad. The shrinkage exponent of de Morgan formulas is 2. *SIAM J. Comput.*, 27(1):48–64, 1998. doi:10.1137/S0097539794261556.
11. Russell Impagliazzo and Noam Nisan. The effect of random restrictions on formula size. *Random Struct. Algorithms*, 4(2):121–134, 1993. doi:10.1002/rsa.3240040202.
12. Stasys Jukna. *Extremal Combinatorics: With Applications in Computer Science*. Springer Publishing Company, Incorporated, 1st edition, 2010.
13. Ilan Komargodski and Ran Raz. Average-case lower bounds for formula size. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 171–180, 2013. doi:10.1145/2488608.2488630.
14. Ilan Komargodski, Ran Raz, and Avishay Tal. Improved average-case lower bounds for de Morgan formula size. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 588–597, 2013. doi:10.1109/FOCS.2013.69.
15. Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput*, 22:838–856, 1993.

16 Moni Naor and Omer Reingold. On the construction of pseudorandom permutations: Luby-Rackoff revisited. *J. Cryptology*, 12(1):29–66, 1999. doi:10.1007/PL00003817.

17 Moni Naor, Omer Reingold, and Alon Rosen. Pseudorandom functions and factoring. *SIAM J. Comput.*, 31(5):1383–1404, 2002. doi:10.1137/S0097539701389257.

18 E. I. Nechiporuk. On a Boolean function. *Soviet Math. Dokl.*, 7(4):999–1000, 1966.

19 Mike Paterson and Uri Zwick. Shrinkage of de Morgan formulae under restriction. *Random Struct. Algorithms*, 4(2):135–150, 1993. doi:10.1002/rsa.3240040203.

20 Alexander A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Matematicheskie Zametki*, 41(4):598–607, 1987. doi:10.1007-BF01137685.

21 Alexander A. Razborov and Steven Rudich. Natural proofs. In *STOC*, pages 204–213, 1994. doi:10.1145/195058.195134.

22 Ben W. Reichardt. Reflections for quantum query algorithms. In *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 560–569, Philadelphia, PA, USA, 2011. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=2133036.2133080.

23 Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 183–192, 2010. doi:10.1109/FOCS.2010.25.

24 Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *STOC*, pages 77–82, 1987. doi:10.1145/28395.28404.

25 B. A. Subbotovskaya. Realizations of linear functions by formulas using $+$, $\cdot$, $-$. *Soviet Math. Dokl.*, 2:110–112, 1961.

26 Amnon Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:41, 2017. URL: https://eccc.weizmann.ac.il/report/2017/041.

27 Avishay Tal. Shrinkage of de Morgan formulae by spectral techniques. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 551–560, 2014. doi:10.1109/FOCS.2014.65.

28 Avishay Tal. Computing requires larger formulas than approximating. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:179, 2016. URL: http://eccc.hpi-web.de/report/2016/179.

29 Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986. doi:10.1016/0304-3975(86)90135-0.

Geodesic Obstacle Representation of Graphs

Prosenjit Bose
School of Computer Science, Carleton University, Ottawa, Canada.
jit@scs.carleton.ca

Paz Carmi
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel.
carmip@cs.bgu.ac.il

Vida Dujmovic
School of Computer Science and Electrical Engineering, University of Ottawa, Ottawa, Canada.
vida.dujmovic@uottawa.ca

Saeed Mehrabi
School of Computer Science, Carleton University, Ottawa, Canada.
saeed.mehrabi@carleton.ca

Fabrizio Montecchiani
Department of Engineering, University of Perugia, Perugia, Italy.
fabrizio.montecchiani@unipg.it

Pat Morin
School of Computer Science, Carleton University, Ottawa, Canada.
morin@scs.carleton.ca

Luis Fernando Schultz Xavier da Silveira
School of Computer Science and Electrical Engineering, University of Ottawa, Ottawa, Canada.
schultz@ime.usp.br

Abstract

An *obstacle representation* of a graph is a mapping of the vertices onto points in the plane and a set of connected regions of the plane (called *obstacles*) such that the straight-line segment connecting the points corresponding to two vertices does not intersect any obstacles if and only if the vertices are adjacent in the graph. The obstacle representation and its *plane* variant (in which the resulting representation is a plane straight-line embedding of the graph) have been extensively studied with the main objective of minimizing the number of obstacles. Recently, Biedl and Mehrabi [5] studied *non-blocking grid obstacle representations* of graphs in which the vertices of the graph are mapped onto points in the plane while the straight-line segments representing the adjacency between the vertices is replaced by the L_1 (Manhattan) shortest paths in the plane that avoid obstacles.

In this paper, we introduce the notion of *geodesic obstacle representations* of graphs with the main goal of providing a generalized model, which comes naturally when viewing line segments as shortest paths in the Euclidean plane. To this end, we extend the definition of obstacle representation by allowing *some* obstacles-avoiding shortest path between the corresponding points in the underlying metric space whenever the vertices are adjacent in the graph. We consider both *general* and *plane* variants of geodesic obstacle representations (in a similar sense to obstacle representations) under any polyhedral distance function in $\mathbb{R}^d$ as well as shortest path distances in graphs. Our results generalize and unify the notions of obstacle representations, plane obstacle representations and grid obstacle representations, leading to a number of questions on such representations.

2012 ACM Subject Classification Theory of computation → Algorithm design techniques, Theory of computation → Computational geometry

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 23; pp. 23:1–23:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Keywords and phrases Obstacle representation, Grid obstacle representation, Geodesic obstacle representation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.23

Related Version A full version of the paper is available at [7], `https://arxiv.org/abs/1803.03705`.

Funding Research of Prosenjit Bose, Vida Dujmovic, and Pat Morin is supported by Natural Sciences and Engineering Research Council of Canada (NSERC). Saeed Mehrabi is supported by a Carleton-Fields postdoctoral fellowship.

Acknowledgements We thank J. Mark Keil for useful discussions on this problem.

1 Introduction

An obstacle representation of an (undirected simple) graph G is pair (φ, S) where $\varphi : V(G) \to \mathbb{R}^2$ maps vertices of G to distinct points in $\mathbb{R}^2$ and S is a set of connected subsets of $\mathbb{R}^2$ with the property that, for every $u, w \in V(G)$, $uw \in E(G)$ if and only if the line segment with endpoints $\varphi(u)$ and $\varphi(w)$ is disjoint from $\cup S$. The elements of S are called *obstacles*. It is easy to see that every graph G has an obstacle representation: obtain a straight-line drawing of G by taking any φ that does not map three vertices of G onto a single line, and let S be the set of the open faces in the resulting arrangement of line segments.

Since every graph has an obstacle representation, this defines a natural graph parameter called the *obstacle number*, $\text{obs}(G) = \min\{|S| : (\varphi, S) \text{ is an obstacle representation of } G\}$. Since their introduction by Alpert et al. [2], obstacle numbers have been studied extensively with the main goal of bounding the obstacle numbers of various classes of graphs (see e.g. [3, 8, 11, 12, 14, 16] and the references therein).

For planar graphs, there is also a natural notion of a *plane obstacle representation* (φ, S) which is an obstacle representation in which φ defines a plane straight-line embedding of G. This leads to *plane obstacle number*: $\text{plane-obs}(G) = \min\{|S| : (\varphi, S) \text{ is a plane obstacle representation of } G\}$. Using Euler's formula, it is not hard to see that the plane obstacle number of any n-vertex planar graph is $O(n)$: let φ define any plane drawing of G with no three vertices collinear and take S to be the set of open faces in this drawing. Since an n-vertex planar graph has at most $2n - 4$ faces, this implies $\text{plane-obs}(G) \leq 2n - 4$.

Recently, Biedl and Mehrabi [5] studied *non-blocking grid obstacle representations* of graphs, consisting of the pair (φ, S) as before in which φ maps the vertices of the graph to points in the plane and S is a set of obstacles, but the adjacency in the graph is represented by replacing straight-line segments with L_1 shortest paths in the plane. That is, for every $u, w \in V(G)$, $uw \in E(G)$ if and only if some L_1 shortest path from $\varphi(u)$ to $\varphi(w)$ is disjoint from $\cup S$; see Figure 1 for an illustration of these obstacle representations.

Geodesic obstacle representation. In this paper, we generalize the notions of obstacle representations [2], plane obstacle representations, and grid obstacle representations [5] by introducing *geodesic obstacle representations* of graphs. This natural generalization of obstacle representations comes from viewing line segments as shortest paths in the Euclidean plane. An obstacle representation (φ, S) has the property that $uw \in E(G)$ if and only if the shortest path from $\varphi(u)$ to $\varphi(w)$ does not intersect $\cup S$. The Euclidean distance is a very special case because the shortest path between any two points p and q is unique. To

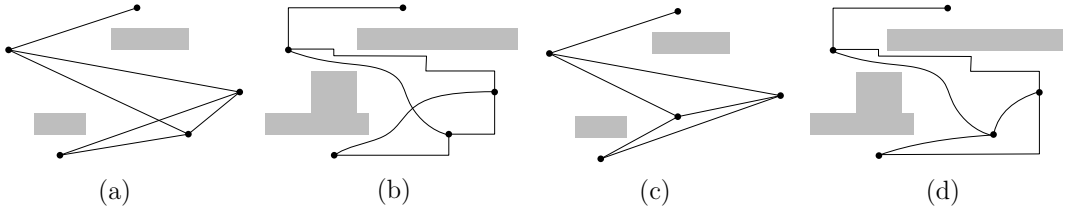


Figure 1 Four different obstacle representations of the same graph G: (a) an obstacle representation, (b) a geodesic obstacle representation under L_1 distance, (c) a plane obstacle representation, and (d) a non-crossing geodesic obstacle representation under L_1 distance.

accommodate other distance measures, we extend the definition of obstacle representation by saying that $uw \in E(G)$ if and only if *some* shortest path from $\varphi(u)$ to $\varphi(w)$ does not intersect $\cup S$. In this way, we can obtain many generalizations of obstacle representations by changing the underlying distance measure. For example, with the L_1 distance measure, every xy-monotone path is a shortest path. Therefore, if (φ, S) is an obstacle representation under L_1, then $uw \in E(G)$ if and only if there is some xy-monotone path from u to w that avoids $\cup S$. Analogous to plane obstacle representations, we can define *non-crossing* geodesic obstacle representations in which φ defines a plane embedding of graph G. Under the L_1 metric, this non-crossing version is equivalent to non-blocking grid obstacle representations as defined by Biedl and Mehrabi [5].

Considering the L_1 metric in the plane, one can view a geodesic obstacle representation of G as a partition of the neighbours of each vertex $u \in V(G)$ into four sets based on which of the four quadrants relative to u the neighbours of u are in the representation. Consequently, if $uv, vw \in E(G)$ in such a way that v is in the same quadrant of u as w is in the quadrant of v in a representation, then we must have $uw \in E(G)$ since there is an xy-monotone path from u to w in the representation. Notice that it is now not clear whether every graph has a geodesic obstacle representation. In fact, Pach [15] showed that there exists a bipartite graph that does not admit a grid obstacle representation. Indeed, the focus of this paper is to determine, for a class $\mathcal{G}$ of graphs, whether or not every member of $\mathcal{G}$ has a geodesic obstacle representation (under some metric space). Clearly, the existence of such representations is more likely if one extends the definition of monotonicity by considering $2k$ equal-angled *cones* around each vertex (instead of $2k = 4$ quadrants), where $k > 2$ is an integer. This leads us to the general question of, informally speaking, what is the minimum integer $k > 0$ for which every member of $\mathcal{G}$ has a geodesic obstacle representation when shortest paths are defined by monotone paths relative to such $2k$ equal-angled cones around each vertex. In this paper, with this "parameter k", we study geodesic obstacle representations and its non-crossing version under polyhedral distance functions in $\mathbb{R}^d$ as well as shortest path distances in graphs. See Section 2 for a formal definition of this generalized notion of obstacle representations.

Related work. It is known that every n-vertex graph has obstacle number $O(n \log n)$ [3] and some n-vertex graphs have obstacle number $\Omega(n/(\log \log n)^2)$ [8]. For planar graphs, there exist planar graphs with obstacle number 2 (the icosahedron is an example [4]), but the best upper bound on the obstacle number of an n-vertex planar graph is $O(n)$. Recall the $O(n)$ upper bound on the plane obstacle number of any n-vertex planar graph by Euler's formula. A lower bound of $\Omega(n)$ is also not difficult: any plane drawing of the $\sqrt{n} \times \sqrt{n}$ grid $G_{\sqrt{n}\times\sqrt{n}}$ has at least $n - 2\sqrt{n}$ bounded faces. Each of these faces has at least four vertices and therefore requires at least one obstacle, so plane-obs$(G_{\sqrt{n}\times\sqrt{n}}) \geq n - 2\sqrt{n}$. Gimbel et al. [11] have nailed the leading constant by showing that every planar graph has plane

obstacle number at most $n-3$, the maximum being attained by planar bipartite graphs. See [2, 3, 8, 11] and the references therein for more details of results on obstacle number and its plane version.

While the obstacle numbers have been extensively studied under the Euclidean distance as shortest path, not much is known about obstacle representations under other shortest path metrics. In fact, we are only aware of the works of Bishnu et al. [6], and Biedl and Mehrabi [5] both of which considered only a restricted version of obstacle representations. Bishnu et al. [6] showed that any n-vertex planar graph G has an obstacle representation on an $O(n^4) \times O(n^4)$ grid in the plane under L_1 metric, with the additional restriction that, for any $uw \in E(G)$, the shortest path from $\varphi(u)$ to $\varphi(w)$ also avoids $\varphi(v)$ for all $v \in V(G) \setminus \{u, w\}$ (in addition to avoiding $\cup S$). Biedl and Mehrabi [5] relaxed this "vertex blocking" constraint and were able to show that every n-vertex planar bipartite graph has a non-blocking grid obstacle representation on an $O(n) \times O(n)$ grid. They left open the problem of finding other classes of graphs for which such non-blocking grid obstacle representations exist and, in particular, whether every planar graph has such a representation.

Our results. In this paper, we prove the following results:

- For any integer $k > 1$, there is a graph with $O(k^2)$ vertices that does not have a geodesic obstacle representation with parameter k. On the other hand, every n-vertex graph has a geodesic obstacle representation with every $k \geq n$.
- For any integer $d > 1$ and any integer $k > 1$, there exists a graph that does not have a geodesic obstacle representation in $\mathbb{R}^d$ with parameter k. On the other hand, every n-vertex graph has a geodesic obstacle representation in $\mathbb{R}^3$ with $k = \lceil (1/2) \log_2 n + 2 \rceil$.
- Every planar graph of treewidth at most 2 (and hence every outerplanar graph) has a non-crossing geodesic obstacle representation with $k = 2$; i.e., a non-blocking obstacle representation.
- Not every planar 3-tree has a non-crossing geodesic obstacle representation with $k = 2$, answering the question asked by Biedl and Mehrabi [5] negatively. Moreover, not every planar 4-connected triangulation has a non-crossing geodesic obstacle representation with $k = 2$.
- Every planar 3-tree has a non-crossing geodesic obstacle representation with $k = 3$. Furthermore, every 3-connected cubic planar graph has a non-crossing geodesic obstacle representation with $k = 7$.
- Every n-vertex graph admits a non-crossing geodesic obstacle representation when taking the D-cube graph as the underlying distance metric, where $D = C \log n$ for some constant $C > 0$.

Organization. We first give some definitions and notation in Section 2. Then, we show our results for (general) geodesic obstacle representations in Section 3 and for its non-crossing version in Section 4. Finally, we give our result for graph metrics in Section 5, and conclude the paper with a discussion on open problems in Section 6.

Throughout this paper, the proofs of lemmas and theorems marked with $(*)$ are given in the full version of the paper [7] due to space constraints.

2 Notation and Preliminaries

Let (X, δ) be a metric space. A *curve* over X is a function $f : [0, 1] \to X$. We call $f(0)$ and $f(1)$ the *endpoints* of the curve f and define the *image* of f as $I(f) = \{f(t) : 0 \leq t \leq 1\}$. A curve f is a *geodesic* if, for every $0 \leq t \leq 1$, $\delta(f(0), f(t)) + \delta(f(t), f(1)) = \delta(f(0), f(1))$. A

path space is a triple $(X, \delta, \mathcal{C})$, where (X, δ) is a metric space and $\mathcal{C}$ is a set of curves over X that has the following closure property: if the curve f is in $\mathcal{C}$ then, for every $0 \leq t \leq 1$, $\mathcal{C}$ also contains the curves $g(x) = f(x \cdot t)$ and $h(x) = f(t + x \cdot (1 - t))$. A path space $(X, \delta, \mathcal{C})$ is *connected* if, for every distinct pair $u, w \in X$, there is some path in $\mathcal{C}$ with endpoints u and w. For a path space $P = (X, \delta, \mathcal{C})$ and a subset $R \subset X$, we denote the subspace induced by R as $P[R] = (R, \delta, \{f \in \mathcal{C} : I(f) \subseteq R\})$. The subspace that *avoids* R is defined as $P \setminus R = P[X \setminus R]$. Moreover, any curve in $P \setminus R$ is called an *R-avoiding curve*. With these definitions in hand, we are ready to define a generalization of obstacle representations.

▶ **Definition 1.** An *$(X, \delta, \mathcal{C})$-obstacle representation* of a graph G is a pair (φ, S) where $\varphi : V(G) \to X$ is a one-to-one mapping and S is a set of connected subspaces of $(X, \delta, \mathcal{C})$ with the property that, for every $u, w \in V(G)$, $uw \in E(G)$ if and only if $\mathcal{C}$ contains a $\cup S$-avoiding geodesic with endpoints $\varphi(u)$ and $\varphi(w)$.

Notice that it is now not clear whether every graph has an $(X, \delta, \mathcal{C})$-obstacle representation. Indeed, the focus of this paper is to determine, for a class $\mathcal{G}$ of graphs and a particular path space $(X, \delta, \mathcal{C})$, whether or not every member of $\mathcal{G}$ has an $(X, \delta, \mathcal{C})$-obstacle representation. This is closely related to certain types of embeddings of G into X. An *embedding* (φ, c) of a graph G into $(X, \delta, \mathcal{C})$ consists of a one-to-one mapping $\varphi : V(G) \to X$ and a function $c : E(G) \to \mathcal{C}$ such that, for each $uw \in E(G)$, the endpoints of $c(uw)$ correspond to $\varphi(u)$ and $\varphi(w)$. The embedding is *geodesic* if $c(uw)$ is a geodesic for every $uw \in E(G)$. Moreover, the embedding (φ, c) is *non-crossing* if $c(uw)$ is disjoint from $c(xz)$, for every $uw, xz \in E(G)$ with $\{u, w\} \cap \{x, z\} = \emptyset$. Observe that given an $(X, \delta, \mathcal{C})$-obstacle representation (φ, S) of G, for each $uw \in E(G)$, we can choose some $\cup S$-avoiding geodesic $c(uw) \in \mathcal{C}$ with endpoints $\varphi(u)$ and $\varphi(w)$. Then, the pair (φ, c) gives a geodesic embedding of G into X. If we can choose c such that (φ, c) is also non-crossing, then we say that the representation (φ, S) is *non-crossing*.

Distance functions. In this paper, we focus on the $(X, \delta, \mathcal{C})$-obstacle representation using polyhedral distance functions in $\mathbb{R}^d$. For a set $N = \{v_0, \ldots, v_{t-1}\}$ of vectors in $\mathbb{R}^d$, we define the *polyhedral distance function*

$$\delta_N(p, q) = \min \left\{ \sum_{i=0}^{t-1} |a_i| : q - p = \sum_{i=0}^{t-1} a_i v_i \right\}.$$

Every such distance function defines a centrally symmetric polyhedron $P_N = \{x \in \mathbb{R}^d : \delta_N(\mathbf{0}, x) \leq 1\}$. The facets of P_N determine the geodesics. For a (closed) facet F of P_N, we denote the *cone* C_F as the union of all rays originating at the origin and containing a point on F (this is the conical hull of F). For a point $x \in \mathbb{R}^d$, the *F-sector* of x is $Q_F^N(x) = C_F + x$. For a facet of F of P_N, we say that a curve f is *δ_N-monotone in direction F* if, for all $0 \leq a \leq b \leq 1$, $f(b) \in Q_F^N(f(a))$. We say that a curve is *δ_N-monotone* if it is δ_N-monotone in direction F for some facet F of P_N. Observe that a curve f is a geodesic for δ_N if and only if f is δ_N-monotone.

▶ **Observation 2.** *If uw and xz are curves that are each δ_N-monotone in direction F and $uw \cap xz$ contains at least one point p, then $\delta_k(u, z) = \delta_k(u, p) + \delta_k(p, z)$ and $\delta_k(x, w) = \delta_k(x, p) + \delta_k(p, w)$.*

When $X = \mathbb{R}^d$, we let $\mathcal{C}_d$ denote the set of curves over $\mathbb{R}^d$. For the sake of compactness, when $X = \mathbb{R}^d$, we denote the $(\mathbb{R}^d, \delta_N, \mathcal{C}_d)$-obstacle representation by *δ_N-obstacle representation*. For the plane case $d = 2$, we define, for each integer $k \geq 2 \in \mathbb{N}$, the *regular distance*

function $\delta_k = \delta_{N_k}$, where $N_k = \{(\cos(i\pi/k), \sin(i\pi/k)) : i \in \{0, \ldots, 2k-1\}\}$. In this case, the associated polygon P_N is a regular $2k$-gon. Moreover, we use δ_k*-obstacle representation* as shorthand for $(\mathbb{R}^2, \delta_k, \mathcal{C}_2)$-obstacle representation. Moreover, for a point in $\mathbb{R}^2$, we denote the i-sector of x by $Q_i^k(x)$, for $i \in \{0, \ldots, 2k-1\}$.

In addition to polyhedral distance functions, we consider obstacle representations under *graph distance*. For a graph H, we denote the set of neighbours of a vertex u in H by $N_H(u)$ and the degree of u by $\deg_H(u)$. Moreover, let δ_H denote the graph distance and let $\mathcal{C}_H$ be the set of curves that define paths in H. Then, we call a $(H, \delta_H, \mathcal{C}_H)$-obstacle representation an H-obstacle representation. If we consider the infinite square grid H_4 (resp., the infinite triangular grid H_6), for instance, then it is not difficult to argue that a graph G has a non-crossing δ_2-obstacle representation (resp., non-crossing δ_3-obstacle representation) if and only if G has a non-crossing H_4-obstacle representation (resp., non-crossing H_6-obstacle representation). In general, for any integer $D > 1$, define the D-cube graph Q_D to be the graph with vertex set $V(Q_D) = \{0,1\}^D$ and that contains the edge uw if and only if u and w differ in exactly one coordinate.

3 General Representations

In this section, we show our results for the general representations. We first consider the special case of $\mathbb{R}^2$ and will then discuss our results for higher dimensions. We start by the following result.

▶ **Theorem 3.** *For any $\epsilon > 0$, there exists a graph G with $n = n(\epsilon)$ vertices such that G has no δ_k-obstacle representation for any $k < n^{1-\epsilon}$.*

Proof. For some constant $c > 0$ and all sufficiently large n, there exists a graph G with n vertices and $cn^{2-2/r}$ edges and that contains no $K_{r,r}$ as subgraph [1]. Let (φ, S) be a δ_k-obstacle representation of G and let (φ, c) be an embedding of G obtained by taking, for each $uw \in E(G)$, $c(uw)$ to be some shortest $\cup S$-avoiding path from $\varphi(u)$ to $\varphi(w)$. From this point on we identify the vertices of G with the points they are embedded to and the edges of G with the curves they are embedded to.

By definition each edge $uw \in E(G)$ is k-monotone. Since P_N has at most $2k$ facets and each edge is monotone in at least two of these directions, this means that it has some facet F such that G contains $E(G)/k$ edges that are monotone in direction F. Consider the graph G' consisting of only these edges and the embedding φ of G'. Observe that if two edges uw and xy of G' intersect at some point p, then (after appropriate relabelling), this implies that there is a $\cup S$-avoiding geodesic from u to x as well as from w to y. Therefore, $ux, uw \in E(G')$.

Therefore, if G' contains an r-tuple of pairwise crossing edges, then G' contains a $K_{r,r}$ subgraph. Now, observe that the edges of G' are monotone in some direction and (after an appropriate rotation) we can assume that they are x-monotone. We call this an x*-monotone embedding*. Valtr [17] has shown that for every fixed r, there exists a constant $C = C(r)$ such that any x-monotone embedding of any n-vertex graph with more than $Cn \log n$ edges contains a set of r pairwise crossing edges. In our case, this means that G contains a $K_{r,r}$ subgraph if $(cn^{2-2/r})/k \geq Cn \log n$, which gives a contradiction when $k \leq cn^{1-2/r}/C \log n$. The result then follows by choosing any $r > 2/\epsilon$. ◀

As $k \to \infty$, δ_k becomes the usual Euclidean distance function and δ_k-obstacle representations are just the usual obstacle representations, which we know every graph has. Thus, for every $n \in \mathbb{N}$, there is a threshold value $k(n)$ such that every n-vertex graph has a $\delta_{k(n)}$-obstacle representation. Theorem 3 shows that $k(n) \in \Omega(n^{1-\epsilon})$ and the following theorem shows that $k(n) \in O(n)$.

▶ **Theorem 4** (∗). *Every n-vertex graph G has a δ_k-obstacle representation for $k = \lceil n/2 \rceil$.*

Higher dimensions. The proof of Theorem 3 makes critical use of the fact that obstacle representations live in the plane so that any sufficiently dense (sub)graph has a k-tuple of pairwise crossing edges. An obvious question, then, is whether every graph has a δ_N-obstacle representation in $\mathbb{R}^3$ (i.e., an $(\mathbb{R}^3, \delta_N, \mathcal{C}_3)$-obstacle representation), where δ_N is some polyhedral distance function. The following theorem shows that the answer to this question is no.

▶ **Theorem 5.** *Let δ_N be a polyhedral distance function over $\mathbb{R}^d$ whose corresponding polyhedron P_N has $2k$ facets, for $k \in o(\log n)$. Then, there exists an n-vertex graph G that has no δ_N-obstacle representation.*

Proof. Let G be an n-vertex graph with no clique and no independent set of size larger than $2 \log n$. The existence of such graphs was shown by Erdős and Renyi [10]. Suppose, for the sake of contradiction, that G has some δ_N-obstacle representation (φ, S). Let $\prec$ denote lexicographic order over points in $\mathbb{R}^d$.

We will k-colour the $\binom{n}{2}$ pairs of vertices of G where the colours are facets of P_N. A pair (u, w) with $u \prec w$ is coloured with a facet F of P_N such that $w \in Q_F^N(u)$. If more than one such facet exists, we choose one arbitrarily. For each $i \in \{1, \ldots, k\}$, let $\prec_i$ denote the partial order obtained by restricting the total order $\prec$ to the pairs of vertices in G with colour i. We claim that for at least one i, $\prec_i$ contains a chain $v_1 \prec_i \cdots \prec_i v_r$ of size $r \geq n^{1/k}$. To see why this is so, observe that, by Dilworth's Theorem, if $\prec_k$ does not contain a chain of length $n^{1/k}$, then it contains an antichain A_k of size $n^{1-1/k}$. Now, proceed inductively on $\prec_1, \ldots, \prec_{k-1}$ and A_k, observing that every pair in A_k is coloured with $\{1, \ldots, k-1\}$.

Next, consider the relation $\prec_i'$ over $v_1, \ldots, v_r$ in which $v_a \prec_i v_b$ if and only if $1 \leq a < b \leq r$ and $v_a v_b \in E(G)$. Observe that $\prec_i'$ is a partial order over $\{v_1, \ldots, v_r\}$. Therefore, by Dilworth's Theorem, it contains a chain of size at least $\sqrt{r}$ or it contains an antichain of size at least $\sqrt{r}$. A chain corresponds to a clique in G and an antichain corresponds to an independent set in G. This contradicts our choice of G when $\sqrt{r} > 2 \log n$, which is true for all $k \in o(\log n)$ and all sufficiently large n. ◀

Theorem 5 shows that, for some n-vertex graphs G, any δ_N-obstacle representation of G must use a distance function δ_N with $k = \Omega(\log n)$ facets. Our next result shows that, even in $\mathbb{R}^3$, a polyhedral distance function with $k = O(\log n)$ facets is indeed sufficient.

▶ **Theorem 6** (∗). *Let δ_N be any polyhedral distance function in $\mathbb{R}^d$, where $d \geq 3$, for which the polyhedron P_N has at least $2 \log_2 n$ facets. Then, every n-vertex graph G has a δ_N-obstacle representation.*

If we take t generic unit vectors in $\mathbb{R}^3$, then the polyhedral distance function determined by these vectors defines a polyhedron having $2t$ vertices and $4t-8$ triangular faces. Theorem 6 therefore implies that a polyhedral distance function determined by $t \geq (1/2) \log_2 n + 2$ unit vectors is sufficient to allow a obstacle representation of any n-vertex graph.

In constant dimensions $d > 3$, there exists sets of t vectors in $\mathbb{R}^d$ defining polytopes with $\Theta(t^{\lfloor d/2 \rfloor})$ facets. Therefore, in $\mathbb{R}^d$, every n-vertex graph has a δ_N-obstacle representation with $|N| \in O(\sqrt[\lfloor d/2 \rfloor]{\log n})$ vectors.

4 Non-Crossing Representations

In this section, we consider non-crossing δ_k-obstacle representations. The following lemma shows that these representations are equivalent to plane δ_k-obstacle embeddings.

▶ **Lemma 7** (∗). *A graph G has a non-crossing δ_k-obstacle representation if and only if G has a non-crossing δ_k-obstacle embedding.*

Lemma 7 allows us to focus our effort on studying the existence (or not) of plane δ_k-obstacle embeddings. We begin with non-crossing δ_k-obstacle embeddings of small treewidth graphs.

Treewidth. A *k-tree* is any graph that can be obtained in the following manner: we begin with a clique on $k+1$ vertices and then we repeatedly select a subset of the vertices that form a k-clique K and add a new vertex adjacent to every element in K. The class of k-trees is exactly the set of edge-maximal graphs of treewidth k. A graph G is called a *partial k-tree* if it is a subgraph of some k-tree. The class of partial k-trees is exactly the class of graphs of treewidth at most k. We will make use of the following lemma, due to Dujmović and Wood [9] in some recursive embeddings.

▶ **Lemma 8** (Dujmović and Wood [9]). *Every k-tree is either a clique on $k+1$ vertices or it contains a non-empty independent set S and a vertex $u \notin S$, such that (i) $G \setminus S$ is a k-tree, (ii) $\deg_{G\setminus S}(u) = k$, and (iii) every element in S is adjacent to u and $k-1$ elements of $N_{G\setminus S}(u)$.*

4.1 δ_2-Obstacle Representations

In this section, we focus on plane δ_2-obstacle embeddings. Recall that these are equivalent to the non-blocking planar grid obstacle representation studied by Biedl and Mehrabi [5]. We begin with the positive result that all graphs of treewidth at most 2 (i.e., partial 2-trees) have plane δ_2-obstacle embeddings.

▶ **Theorem 9.** *Every partial 2-tree has a plane straight-line δ_2-obstacle embedding.*

Proof. Let G be a partial 2-tree. We can, without loss of generality, assume that G is connected. If $|V(G)| < 4$, then the result is trivial, so we can assume $|V(G)| \geq 4$. We now proceed by induction on $|V(G)|$.

Let $T = T(G)$ be a 2-tree with vertex set $V(G)$ and that contains G. Apply Lemma 8 to find the vertex set S and the vertex u. Let x and y be the neighbours of u in $T \setminus S$. Now, apply induction to find a plane straight-line δ_2-obstacle embedding of the graph G' whose vertex set is $V(G') = V(G) \setminus S$ and whose edge set is $E(G') = E(G \setminus S) \cup \{ux, uy\}$. Denote by S_x (resp., S_y) the neighbours of x (resp., y) that belong to S.

Now, observe that, since u has degree 2 in G' and the edges ux and uy are in G', this embedding does not contain any monotone path of the form uxw or uyw for any $w \in V(G) \setminus \{u, x, y\}$. Therefore, if we place the vertices in S sufficiently close to u, we will not create any monotone path of the form ayw or axw for any $a \in S$ and any $w \in V(G) \setminus \{u, x, y\}$. What remains is to show how to place the elements of S in order to avoid unwanted monotone paths of the form uay, uax, or aub for any $a, b \in S$. There are three cases to consider:

1. $x \in Q_i^2(u)$ and $y \in Q_{i+2}^2(u)$ for some $i \in \{0, \ldots, 3\}$. W.l.o.g., assume that $Q_{i+3}^2(u)$ does not intersect the segment xy. Then, we can embed the elements of S in $Q_{i+3}^2(u)$ without creating any new monotone paths; see Figure 2(a).
2. $x, y \in Q_i^2(u)$ for some $i \in \{0, \ldots, 3\}$. There are two subcases:
   - **(i)** At least one of ux or uy is in $E(G)$. Suppose $ux \in E(G)$. Then we embed S_x in $Q_i^2(u)$ and embed S_y in $Q_{i+3}^2(u)$; see Figure 2(b). The only monotone paths this creates are of the form uax with $a \in S_x$, which is acceptable since $ux \in E(G)$.

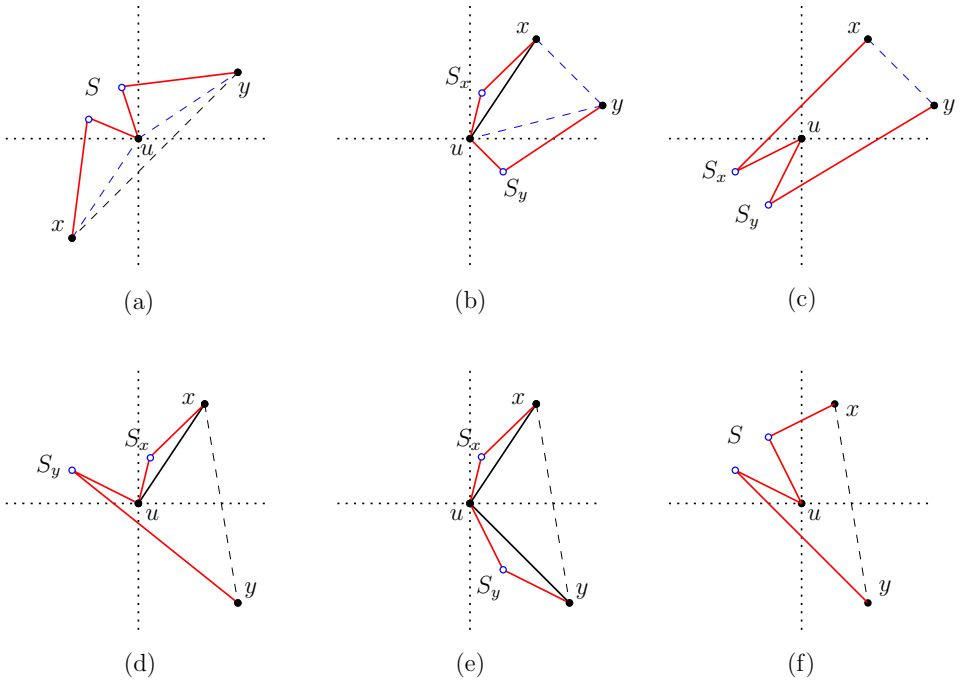


Figure 2 An illustration in supporting the proof of Theorem 9.

(ii) Neither ux nor uy is in $E(G)$. In this case, we embed all of S in $Q^2_{i+2}(u)$ (see Figure 2(c)). This does not create any new monotone paths.

3. $x \in Q^2_i(u)$ and $y \in Q^2_{i+3}(u)$ for some $i \in \{0, \ldots, 3\}$. We have three subcases to consider:

(i) $|\{ux, uy\} \cap E(G)| = 1$. In this case, assume $ux \in E(G)$. Then, we embed the vertices of S_x in $Q^2_i(u)$ and we embed the vertices of S_y in $Q^2_{i+1}(u)$. See Figure 2(d). The only monotone paths this creates are of the form uax with $a \in S_x$, which is acceptable since $ux \in E(G)$.

(ii) $|\{ux, uy\} \cap E(G)| = 2$. In this case, we embed the vertices of S_x in $Q^2_i(u)$ and we embed the vertices of S_y in $Q^2_{i+3}(u)$ (see Figure 2(e)). The only monotone paths this creates are of the form uax with $a \in S_x$ and uby with $b \in S_y$, which is acceptable since $ux, uy \in E(G)$.

(iii) $|\{ux, uy\} \cap E(G)| = 0$. In this case, we embed all of S into Q^2_{i+1} (see Figure 2(f)). This does not create any new monotone paths.

This completes the proof of the theorem. ◀

In the full version of the paper [7], we show that not every planar 3-tree admits a non-crossing δ_2-obstacle embedding.

▶ **Theorem 10** (∗). *There exists a planar 3-tree that does not have a non-crossing δ_2-obstacle embedding.*

We further prove that even 4-connectivity does not help to guarantee the existence of non-crossing δ_2-obstacle embeddings. To this end, we show that a 4-connected triangulation having a plane δ_2-obstacle representation must have a constrained 4-colouring in the sense that, for the neighbours of a vertex, which colours and in what order are they allowed to be assigned to them. The following theorem then follows by finding a 4-connected triangulation that does not admit such a constrained 4-colouring (see the full version of the paper [7]).

▶ **Theorem 11** (∗). *There exists a 4-connected triangulation G with maximum degree 7 that has no plane δ_2-obstacle embedding.*

4.2 Higher-k δ_k-Obstacle Representations

In this section, we consider the non-crossing embeddings for $k > 2$. We show that planar 3-trees have plane δ_3-obstacles embeddings and that all 3-connected cubic planar graphs have plane δ_7-obstacles embeddings. We start by planar 3-trees.

▶ **Theorem 12.** *Every planar 3-tree has a plane δ_3-obstacle embedding.*

Proof Sketch. Here, we sketch the proof; see the full version of the paper [7] for the complete proof. The proof is by induction on $n = |V(G)|$ in which our inductive hypothesis is that every n vertex planar 3-tree has a plane δ_3-obstacle embedding in which the neighbours of each vertex u occupy at least 3 of the sectors $Q_0^3(u), \ldots, Q_5^3(u)$. The key to our proof is the result of Dujmovic and Wood [9] when specialized to planar 3-trees, which says that every planar 3-tree is either K_4 or has a vertex u and an independent set S ($|S| \leq 3$) such that $G \setminus S$ is a 3-tree, u has degree 3 in $G \setminus S$ with neighbours x, y and z, and every vertex r in S forms a clique with exactly one of uxy, uyz or uzx.

By applying this result and recursing on $G \setminus S$ (when $n > 4$), we obtain a plane δ_3-obstacle embedding of $G \setminus S$. By our induction hypothesis, there are two cases depending on the locations of x, y and z with respect to u. In both cases, the elements of S are placed close enough to u that we do not create any new δ_3-monotone paths involving vertices other than those in $\{u, x, y, z\} \cup S$. Since $\{u, x, y, z\}$ form a clique, we only need to worry about (possibly) creating a new δ_3-monotone path involving at least one vertex of S. ◀

We next show that every 3-connected cubic planar graph has a plane δ_7-obstacle embedding. The algorithm contrustructs a δ_7-obstacle embedding by adding one vertex per time according to a canonical ordering of the graph [13], and at each step it maintains a set of geometric invariants which guarantee its correctness. The key ingredients are the fact that each new vertex v to be inserted has exactly two neighbors in the already constructed representation, together with the existence of a set of edges whose removal disconnects the representation in two parts, each containing one of the two neighbors of v. A sufficient stretching of these edges allows for a suitable placement for vertex v. See the full version of the paper for details [7].

▶ **Theorem 13** (∗). *Every 3-connected cubic plane graph has a plane δ_7-obstacle embedding.*

5 Graph Metrics

In this section, we consider the problem under graph distances. Recall the graph D-cube, Q_D whose vertex set is $V(Q_D) = \{0, 1\}^D$ and that contains the edge uw if and only u and w differ in exactly one coordinate. It is not hard to see that every n vertex graph has a Q_n-obstacle representation: Each vertex of G is assigned a coordinate with a single 1 bit. Then, for any two vertices u and w there are exactly two shortest paths in Q_n joining them and they each have length 2. One path goes through the intermediate vertex $\mathbf{0} = (0, \ldots, 0)$ and the other goes through $u + w$. Therefore, by placing an obstacle at $\mathbf{0}$ and at each $u + w$ for which $uw \notin E(G)$, we obtain a Q_D-obstacle representation of G. The following theorem shows we can do this with much fewer coordinates.

▶ **Theorem 14.** *There exists a constant $C > 0$ such that, for $D = C \log n$, every n-vertex graph has a non-crossing Q_D-obstacle representation.*

Proof. Consider the following embedding (φ, c) of G into Q_D: For each $u \in V(G)$, $\varphi(u)$ is a random element of $\{0,1\}^D$. We use the notation u_i to denote the ith coordinate of u. Let $\prec$ denote lexicographic order on D-tuples. For each edge $uw \in E(G)$ with $u \prec w$, we take $c(uw)$ to be the *greedy* path that visits, for $i = 0, \ldots, D$, the vertex $uw_i = (w_1, \ldots, w_i, u_{i+1}, \ldots, u_D)$. Thus $uw_0, \ldots, uw_D$ is a sequence of vertices that—after removing duplicates—is a shortest path, in Q_D, from u to w. Note that there is an asymmetry here that we should be careful of, so for $u \prec w$, we define $wu_i = (w_1, \ldots, w_{D-i}, u_{D-i+1}, \ldots, u_D) = uw_{D-i}$. Here are some observations about the embedding (φ, c):

1. All vertex distances are close to $D/2$: The distance between any two vertices is a binomial$(D, 1/2)$ random variable. Therefore, by Chernoff's bounds, for any constant $\epsilon > 0$ and for any vertex pair $u \neq w$, $\Pr\{|\delta_{Q_D}(u,w) - D/2| > \epsilon(D/2)\} \leq n^{-\Omega(C)}$. By the union bound, the probability that there exists any pair of vertices $u \neq w$ with $|\delta_{Q_D}(u,w) - D/2| > \epsilon(D/2)$ is also $n^{-\Omega(C)}$.
2. The embedding is non-crossing: For any four distinct vertices $u \prec w$ and $x \prec y$, and any $i, j \in \{0, \ldots, D\}$, the vertices uw_i and xy_j are independent random D-bit strings. Therefore, $\Pr\{\delta_{Q_D}(uw_i, xy_j) \leq 1\} = (D+1)/2^D$. By the union bound, the probability that there exists any four vertices u, w, x, y and any pair of indices i, j for which $\delta_{Q_D}(uw_i, xy_j) \leq 1$ is at most $n^4(D+1)^3/2^D = n^{-\Omega(C)}$.
3. No geodesic passes close to a vertex except its endpoints: Let u, w, and x be distinct vertices and $r \in \{0, \ldots, D\}$ be an integer. Then, the probability that there exists any geodesic with endpoints u and w that contains a vertex z with $\delta_{Q_D}(z, x) \leq r$ is at most $n^{-\Omega(C)}$. To see why this is so, suppose that such a geodesic, C, contains a vertex z such that $\delta_{Q_D}(z, x) \leq r$. Then, at least one of the following events occurs:
   (a) $\delta_{Q_D}(u, w) \geq (1+\epsilon)D/2$;
   (b) $\delta_{Q_d}(u, x) \leq (1+\epsilon)D/4 + r$; or
   (c) $\delta_{Q_d}(w, x) \leq (1+\epsilon)D/4 + r$.
   Point 1 above establishes that the probability of the first event is $n^{-\Omega(C)}$ and that, for $r \leq (1-3\epsilon)D/4$, the probability of each of the other two events is $n^{-\Omega(C)}$. Applying the union bound over all 3 events, and over all $\binom{n}{3}$ choices of u, w, and x then shows that the probability that there is any triple u, w, x such that any geodesic from u to w passes within distance $(1-3\epsilon)D/4$ of x is $n^{-\Omega(C)}$.
4. Paths diverge quickly: Let $xu, xw \in E(G)$, be two edges of G with the common endpoint x and let $r \in \{0, \ldots, D\}$. We want to show that the directed paths xu and xw diverge quickly. There are three cases to consider:
   a. $x \prec u$ and $x \prec w$. In this case $xu_r = xw_r$ if and only if $u_1, \ldots, u_r = w_1, \ldots, w_r$, so $\Pr\{xu_r = xw_r\} = 2^{-r}$.
   b. $x \prec u$ and $w \prec x$. In this case, we consider $xu_r = u_1, \ldots, u_r, x_{r+1}, \ldots, x_D$ and $xw_r = wx_{D-r} = x_1, \ldots, x_{D-r}, w_{D-r+1}, \ldots, w_D$. For any choice of i, these two strings have independent bits in at least r locations, so $\Pr\{xu_r = xw_r\} \leq 2^{-r}$.
   c. $u \prec x$ and $w \prec x$. In this case $xu_r = ux_{D-r} = x_1, \ldots, x_{D-r}, u_{D-r+1}, \ldots, u_D$ and $xw_r = wx_{D-r} = x_1, \ldots, x_{D-r}, w_{D-i+1}, \ldots, w_D$. So $\Pr\{xu_r = xw_r\} = 2^{-r}$.
   If we choose $r = \alpha \log n$, then this probability is at most $n^{-\Omega(\alpha)}$. Again, the union bound shows that the probability that there is any u, w, or x such that $xu_r = xw_r$ is at most $n^{-\Omega(\alpha)}$.

In the following, we choose C sufficiently large and $\alpha < (1/4 - \epsilon)C$ also sufficiently large so that with probability greater than 0, we obtain an embedding for which all four of preceding properties hold. Therefore, there exists some embedding (φ, c) such that 1. for all $u, w \in V(G)$, $|\delta_{Q_D}(u,w) - D/2| \leq \epsilon D/2$; 2. for all $uw, xy \in E(G)$ with $\{u, w\} \cap \{x, y\} = \emptyset$,

$\delta_{Q_D}(c(uw), c(xy)) > 1$; 3. for all $uw \in E(G)$ and $x \in V(G) \setminus \{u, w\}$, $\delta_{Q_D}(c(uw), x) \geq (1-\epsilon)D/4$; and 4. for all $xu, xw \in E(G)$ and all $r \geq \alpha \log n$, $xu_r \neq xw_r$.

To obtain a Q_D-obstacle representation (φ, S) we take S to contain all the vertices not used in any path of the embedding (φ, c). To verify that this is indeed a non-crossing Q_D-obstacle representation, we need only verify that, for any $u, w \in V(G)$ with $uw \notin E(G)$, $\delta_{Q_D \setminus S}(u, w) > \delta_{Q_D}(u, w)$. This is implied by the following inequality, which relates distances in G to those in $Q_D \setminus S$:

$$\delta_{Q_D \setminus S}(u, w) \geq \delta_G(u, w)(1-\epsilon)D/2 - (\delta_G(u, w) - 1)2\alpha \log n \ . \tag{1}$$

Note (1) is sufficient since, if $uw \notin E(G)$, then $\delta_G(u, w) \geq 2$ and (1) implies $\delta_{Q_D \setminus S}(u, w) \geq (1-\epsilon)D - 2\alpha \log n = ((1-\epsilon)C - 2\alpha) \log n > (1+\epsilon)D/2$, which contradicts Property 1. Thus, all that remains is to establish (1). To do this, consider any path P from u to w in $Q_D \setminus S$. Since the only vertices in $Q_D \setminus S$ are those that are used by some embedded edge of G, the path P consists of a sequence of subpaths $P_0, \ldots, P_k$ where each P_i is a subpath of $c(x_i y_i)$ for some edge $x_i y_i \in E(G)$. Note that Property 3 implies that $x_0 = u$ and that $x_k = w$. Furthermore, Properties 2 and 3 imply that $x_i = y_{i-1}$ for each $i \in \{1, \ldots, k\}$. Therefore, $x_0, \ldots, x_k$ is a path in G from u to w, so $k \geq \delta_G(u, w)$. Finally, Property 4 implies that, for each $i \in \{1, \ldots, k-1\}$, the portion of $c(x_i, x_{i+1})$ not used by P_i has length at most $2\alpha \log n$. Thus, the length of P is at least $k(1-\epsilon)D/2 - 2(k-1)\alpha \log n$, as required. ◀

It is worth noting that Theorem 14 is closely related to Theorem 6. Indeed, before perturbing it, the point set X used in the proof of Theorem 6 is a projection of the vertices of Q_D with $D = \lceil \log_2 n \rceil$ onto $\mathbb{R}^3$. In Theorem 6 we then perturb X to obtain a non-crossing embedding. In the proof of Theorem 14 we have to be more careful to avoid crossings.

6 Conclusion

In this paper, we introduced the geodesic obstacle representation of graphs, providing a unified generalization of obstacle representations and grid obstacle representations. Our work leaves several problems open. As perhaps the main question, does every planar graph admit a non-crossing δ_k-obstacle representation for some constant k? It would be also interesting to extend the classes of graphs for which non-crossing δ_k-obstacle representations exist for small values of k. For graph metrics, given two graphs G and H, is it NP-hard to decide if G has an H-obstacle representation?

References

1 Noga Alon, Michael Krivelevich, and Benny Sudakov. Turán numbers of bipartite graphs and related Ramsey-type questions. *Combinatorics, Probability & Computing*, 12(5-6):477–494, 2003.

2 Hannah Alpert, Christina Koch, and Joshua D. Laison. Obstacle numbers of graphs. *Discrete & Computational Geometry*, 44(1):223–244, 2010.

3 Martin Balko, Josef Cibulka, and Pavel Valtr. Drawing graphs using a small number of obstacles. *Discrete & Computational Geometry*, 59(1):143–164, 2018.

4 Leah Wrenn Berman, Glenn G. Chappell, Jill R. Faudree, John Gimbel, Chris Hartman, and Gordon I. Williams. Graphs with obstacle number greater than one. *J. Graph Algorithms Appl.*, 21(6):1107–1119, 2017.

5 Therese C. Biedl and Saeed Mehrabi. Grid-obstacle representations with connections to staircase guarding. In *proceedings of the 25th International Symposium on Graph Drawing and Network Visualization (GD 2017), Boston, MA, USA*, 2017.

6 Arijit Bishnu, Arijit Ghosh, Rogers Mathew, Gopinath Mishra, and Subhabrata Paul. Grid obstacle representations of graphs, 2017. arXiv version available at: arxiv.org/abs/1708.01765.

7 Prosenjit Bose, Paz Carmi, Vida Dujmovic, Saeed Mehrabi, Fabrizio Montecchiani, Pat Morin, and Luís Fernando Schultz Xavier da Silveira. Geodesic obstacle representation of graphs. *CoRR*, abs/1803.03705, 2018.

8 Vida Dujmovic and Pat Morin. On obstacle numbers. *Electr. J. Comb.*, 22(3):P3.1, 2015.

9 Vida Dujmovic and David R. Wood. Graph treewidth and geometric thickness parameters. *Discrete & Computational Geometry*, 37(4):641–670, 2007.

10 Paul Erdős and Alfred Renyi. On random graphs. *Publicationes Mathematicae*, 6:290–297, 1959.

11 John Gimbel, Patrice Ossona de Mendez, and Pavel Valtr. Obstacle numbers of planar graphs. In *25th International Symposium Graph Drawing and Network Visualization (GD 2017), Boston, MA, USA*, pages 67–80, 2017.

12 Matthew P. Johnson and Deniz Sariöz. Representing a planar straight-line graph using few obstacles. In *proceedings of the 26th Canadian Conference on Computational Geometry, (CCCG 2014), Halifax, NS, Canada*, 2014.

13 Goos Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996.

14 Padmini Mukkamala, János Pach, and Dömötör Pálvölgyi. Lower bounds on the obstacle number of graphs. *Electr. J. Comb.*, 19(2):P32, 2012.

15 János Pach. Graphs with no grid obstacle representation. *Geombinatorics*, 26(2):80–83, 2016.

16 János Pach and Deniz Sariöz. On the structure of graphs with low obstacle number. *Graphs and Combinatorics*, 27(3):465–473, 2011.

17 Pavel Valtr. Graph drawings with no k pairwise crossing edges. In *5th International Symposium on Graph Drawing (GD 1997), Rome, Italy*, pages 205–218, 1997.

The Bottleneck Complexity of Secure Multiparty Computation

Elette Boyle
IDC Herzliya
elette.boyle@idc.ac.il

Abhishek Jain
Johns Hopkins University
abhishek@cs.jhu.edu

Manoj Prabhakaran
Indian Institute of Technology Bombay
mp@cse.iitb.ac.in

Ching-Hua Yu
University of Illinois at Urbana-Champaign
cyu17@illinois.edu

Abstract

In this work, we initiate the study of *bottleneck* complexity as a new communication efficiency measure for secure multiparty computation (MPC). Roughly, the bottleneck complexity of an MPC protocol is defined as the maximum communication complexity required by any party within the protocol execution.

We observe that even without security, bottleneck communication complexity is an interesting measure of communication complexity for (distributed) functions and propose it as a fundamental area to explore. While achieving $O(n)$ bottleneck complexity (where n is the number of parties) is straightforward, we show that: (1) achieving *sublinear* bottleneck complexity is *not* always possible, even when no security is required. (2) On the other hand, several useful classes of functions do have $o(n)$ bottleneck complexity, when no security is required.

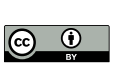
Our main positive result is a compiler that transforms any (possibly insecure) efficient protocol with fixed communication-pattern for computing any functionality into a secure MPC protocol while preserving the bottleneck complexity of the underlying protocol (up to security parameter overhead). Given our compiler, an efficient protocol for any function f with sublinear bottleneck complexity can be transformed into an MPC protocol for f with the same bottleneck complexity.

Along the way, we build cryptographic primitives – incremental fully-homomorphic encryption, succinct non-interactive arguments of knowledge with ID-based simulation-extractability property and verifiable protocol execution – that may be of independent interest.

2012 ACM Subject Classification Theory of computation → Cryptographic protocols, Theory of computation → Communication complexity

Keywords and phrases distributed protocols, secure computation, communication complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.24



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 24; pp. 24:1–24:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Secure multi-party computation (MPC) [32, 20] is a fundamental notion in cryptography, enabling a collection of mutually distrusting parties to jointly evaluate a function on their private inputs while revealing nothing beyond the function output. In the past decades, a great deal of research has been dedicated to the design and optimization of efficient MPC protocols.

In this work, we study one fundamental metric of MPC efficiency: the required *communication* between parties. In particular, we focus on the communication complexity of MPC in large-scale settings, where the number of participants is significant.

In nearly all existing works in MPC literature, the communication complexity goal has been to minimize the *total* communication of the protocol across all n parties. However, for many important applications, such as peer-to-peer computations between lightweight devices,[1] total costs (such as total communication) are only secondarily indicative of the feasibility of the computation, as opposed to the primary issue of *per-party* cost. Indeed, while a total communication bound L implies average per-party communication of the protocol is L/n, the computation may demand a subset of the parties to *each* communicate as much as $\Theta(L)$. When all parties contribute input to the computation, then $L \geq n$, meaning these parties must bear communication proportional to the *total number of parties*. In large-scale distributed settings, or when the protocol participants are lightweight devices, such a requirement could be prohibitive.

New efficiency measure: (MPC) Bottleneck Complexity

To address these concerns, we initiate the study of *bottleneck complexity* of MPC. The bottleneck complexity of a protocol Π is defined as the *maximum communication required by any party* within the protocol execution. One may further specialize this to incoming versus outgoing communication. The MPC bottleneck complexity of a (distributed) function is the minimum possible bottleneck complexity of a secure MPC protocol for the function. In this work, our goal is to explore this notion as a complexity measure for distributed computations, and to develop secure protocols with low bottleneck complexity.

Bottleneck complexity addresses certain (practically important) aspects ignored by standard communication complexity. For instance, if two messages are transmitted in two different parts of a network, say $A \to B$ and $C \to D$, they would be delivered faster than two messages sent to/from the same party, say $A \to B$ and $C \to B$. While both have same total communication, the latter has higher bottleneck communication.

Bottleneck Complexity without Security

Before studying bottleneck communication complexity for secure protocols, we first consider this measure for arbitrary protocols without any security considerations. Indeed, this already forms an interesting measure of complexity for (distributed) functions, and we propose it as a fundamental area to explore. As in the case of total communication complexity (which coincides with bottleneck complexity for the case of 2 parties), there is a trivial upper bound of $O(n)$ bottleneck complexity for any n-party functionality (with boolean inputs), where all parties simply send their input to a central party who computes the functionality. On

[1] For example, optimizing navigation routes based on traffic information contributed by the cell phones of drivers on the road, without revealing the locations of individual users.

the other hand, in many functions, bottleneck complexity brings out structures that total communication complexity overlooks. For instance, in computing say the XOR or AND of n bits, total communication complexity is $\Theta(n)$, but the bottleneck complexity is $O(1)$. These functions naturally allow for *incremental computation* along a chain, in which each party receives and sends a single bit. Indeed, there is a large class of useful functions which have protocols with low bottleneck complexity, as discussed below.

However, a priori it is not clear whether *all* functions can be computed in a similar manner. This brings us to the first question considered in this work:

Can all functions be computed (without security)
with *sublinear* bottleneck complexity?

For concreteness, we may consider n-party functions, with n inputs (one for each party) each k bits long, and a single-bit output. Because of the trivial $O(nk)$ upper bound on total communication complexity for any such function (as discussed above), each party *on average* needs only to communicate $O(k)$ bits. But in this protocol, the communication complexity of the central party—and thus bottleneck complexity of the protocol—is $(n-1)k$ bits. Surprisingly, we show that this is the best one can ask for, for general functions. That is, there exist n-party functionalities with k-bit inputs for which the bottleneck complexity is $\Omega(nk)$.

▶ **Theorem 1.** *(Informal.) There exist n-party functions with k-bit input for each party that have bottleneck complexity close to that in the trivial upperbound, namely $(n-1)k$.*

Our proof is based on a counting argument, and quantifies over possibly inefficient functions too. Interestingly, giving an explicit efficient function f with such a lower bound will require a breakthrough in complexity theory, as it would imply an $\Omega(n^2)$ lower bound on the circuit size of computing f. (We discuss this connection below.)

Functions with Low Bottleneck Complexity

Despite the above lower bound, there is a large class of interesting functions which do have sublinear bottleneck complexity. One simple but widely applicable example is addition in a finite group: the sum of n group elements distributed among n parties can be aggregated bottom-up (and then disseminated top-down) using a constant-degree tree, with every party communicating $O(d)$ group elements, where d is its degree in the tree. [2]

A wider class of functions are obtained from the literature on streaming algorithms [18, 26]. Indeed, any streaming algorithm with a small memory and a small number of passes corresponds to a low bottleneck complexity function. (Here, we refer to the actual function that the streaming algorithm computes, which may in fact be an approximation to some other desired function.) This is because we can design a protocol which passes around the state of the streaming algorithm from one party to the next, in the order in which their inputs are to be presented to the algorithm.

On the other hand, low bottleneck complexity protocols appear to be much more general than streaming algorithms. Indeed, observe that the low bottleneck complexity protocol described above has a very special communication structure of a chain (or multi-pass cycle). We leave it as an open problem to separate these two notions – i.e., find functions which have low bottleneck complexity protocols, but do not have low-memory streaming algorithms.

[2] If the group is not abelian, the tree used should be such that its in-order traversal should result in the parties to be ordered in the same way their inputs are ordered in the sum being computed.

Finally, we note that, any n-input function with a constant fan-in circuit of subquadratic size (i.e., $o(n^2)$ gates) has a sublinear bottleneck complexity protocol. To see this, first we note that such a circuit can be made to have constant fan-out as well, by increasing the circuit size by a constant factor. [3] Then, a sublinear bottleneck complexity protocol can be obtained from the circuit by partitioning all the $o(n^2)$ gates roughly equally among the n parties, and letting the parties evaluate the gates assigned to them, communicating with each other when wires cross party boundaries. The communication incurred by each party is bounded by the number of wires incident on all the gates it is assigned, which is $o(n)$.

MPC Bottleneck complexity

We next turn our attention to achieving low bottleneck complexity for *secure* computation of functionalities. We focus on the general setting where up to $n-1$ out of n parties can be corrupted. As a baseline, we observe that the MPC protocol of Dodis *et al.* [15] based on "additive-spooky encryption" can be easily adapted to obtain generic secure computation with $O(n)$ bottleneck complexity (where $O(n)$ hides factors of the security parameter). Therefore, as in the insecure setting, we focus on constructing MPC protocols with sublinear $o(n)$ bottleneck complexity.

Specifically, we ask the question:

> If a function f can be computed with bottleneck complexity L,
> can it be computed *securely* with the same bottleneck complexity,
> up to a multiplicative overhead of the security parameter?

We note that the goal of sublinear bottleneck complexity is strictly stronger than the recently studied problem of MPC with sublinear communication locality [5]. The locality of a protocol is the maximum number of other parties that any party must communicate with during the course of the protocol. It is easy to see that sublinear bottleneck complexity directly implies sublinear locality (since sending/receiving $o(n)$ bits means that a party can only communicate with $o(n)$ neighbors); however, as locality does not place any requirements on the number of bits communicated by a party, the converse is not true. Indeed, without security requirements, every function has an $O(1)$-local protocol, which is not the case for bottleneck complexity.

We show a general compiler which transforms any (possibly insecure) efficient multi-party protocol Π for computing a function f into a protocol Π' for *securely* computing f, preserving the per-party communication and computation requirements up to $O(\lambda^c)$ factors in the security parameter λ for small constant c. The original protocol Π can have an arbitrary communication pattern; however, we require that this pattern must be fixed a priori (independent of inputs) and known to all parties. Our compiler additionally preserves the topology of communication graph of Π (and in particular, preserves locality).

▶ **Theorem 2** (Informal). *There is a transformation which maps any (possibly insecure) efficient protocol with fixed-communication-pattern for an n-party distributed function f into a secure MPC protocol for f with asymptotically (as a function of n) the same communication and computational requirements per party, and using the same communication graph as the original protocol.*

[3] Given a gate with fan-out $d > 2$, consider the depth-1 tree T rooted that gate with d leaves being the gates to which its outputs are connected. T can be replaced by an equivalent *binary* tree T' with the same root and leaves, and $d-2$ new internal nodes. The new internal nodes of T' can be "charged" to the leaves of T. On doing this for all gates in the circuit, each gate gets charged at most as many times as its fan-in. Since each gate in the original circuit has constant fan-in, this transformation increases the circuit size by at most a constant factor.

The transformation to achieve security against passive corruption is based on a new tool that we develop, namely *Incremental Fully Homomorphic Encryption*, which we show can be instantiated from the Learning With Errors (LWE) assumption, à la [19]. For security against active corruption (possibly for restricted auxiliary information), we build zero-knowledge succinct non-interactive arguments of knowledge (ZK-SNARK [2]) with an"ID-based" simulation-extractability property (see Section 1.1 for details). We rely on a setup that includes a common random string and a (bare) public-key infrastructure, where all the n parties have deposited keys for themselves, and which all the parties can access for free. The setup can be reused for any number of executions.

Our Contributions

To summarize, our main contributions in this work are as follows:

- We introduce a new measure of per-party communication complexity for (distibuted) functions, called bottleneck complexity.
- We demonstrate the existence of n-party functions with k bits of input for each party, that have bottleneck complexity $\Theta(nk)$. Showing an explicit function with $\Omega(n)$ bottleneck complexity will require showing an explicit function with $\Omega(n^2)$ circuit size complexity. On the other hand, we observe that many useful classes of functions do have $o(n)$ bottleneck complexity.
- We show a general transformation from arbitrary efficient protocols to secure MPC protocols (in a model with public setup) that asymptotically (as a function of n) preserves the communication and computational requirements per party, and preserves the same communication graph.
- As part of our transformation, we introduce cryptographic primitives—Incremental FHE, Verifiable Protocol Execution—and give a construction of ZK-SNARKs with an ID-based simulation-extractability property. These may be of independent interest.

We expand on the transformation in the following section.

1.1 Our Techniques

We describe the main ideas underlying our positive result: the bottleneck-complexity-preserving transformation from arbitrary protocols to secure ones.

At a high-level, we follow an intuitive outline for our compiler: (1) We first compile an insecure protocol into a protocol that is secure against semi-honest (or honest-but-curious) adversaries using fully homomorphic encryption (FHE). (2) We then use zero-knowledge succinct arguments of knowledge (ZK-SNARKs) to compile it into a protocol that is (standalone) secure against malicious adversaries. However, we run into several technical challenges along the way, requiring us to develop stronger guarantees for FHE and SNARKs, as well as some other new ideas. We elaborate on these challenges and our solutions below.

Semi-honest Security

A natural starting idea to obtain semi-honest security is to execute an "encrypted" version of the underlying (insecure) protocol by using FHE. Once the parties have the encrypted output, they execute the FHE decryption process to learn the output. The immediate problem with implementing this idea in the multiparty setting is which key must we use for encryption and decryption. If a single party knows the (entire) decryption key, then we cannot guarantee security.

To address this problem, two approaches have been developed in the literature: threshold FHE [1], where the parties jointly generate a public key for an FHE scheme such that each party only knows a share of the decryption key, and multi-key FHE [25], where each party has its own public and secret key pair and FHE evaluation can be performed over ciphertexts computed w.r.t. different public keys.

While these approaches have been shown to suffice for constructing round-efficient MPC protocols, they are not directly applicable to our setting. This is for two reasons:

- Threshold FHE and multi-key FHE systems are defined in the broadcast model of communication where each party gets to see the messages sent by all the other parties. In contrast, our setting is inherently point-to-point, where a party only communicates with its neighbors in the communication graph of the underlying insecure protocol. Indeed, in order to maintain sublinear bottleneck complexity, we cannot afford each party to communicate with all the other parties.
- Further, in all known solutions for threshold FHE [1] and multi-key FHE [25, 9, 27, 6, 29], the size of one or more protocol messages of each party grows at least linearly with the number of parties. This directly violates our sublinear bottleneck complexity requirement.

To address these issues, we define and implement a new notion of *incremental* FHE (IFHE). Roughly, an IFHE scheme is defined similarly to threshold FHE, with the following key strengthened requirements: a "joint" public key can be computed by incrementally combining shares provided by different parties in an arbitrary order. Similarly, a ciphertext w.r.t. the joint public key can be decrypted by incrementally combining partial decryption shares provided by parties in an arbitrary order. Crucially, the intermediate keys and partial decryption values must be *succinct*.

We construct an IFHE scheme with appropriate security guarantees based on the Gentry-Sahai-Waters FHE scheme [19]. Using IFHE, we are able to directly compile an insecure protocol into a semi-honest secure protocol. In fact, this protocol can withstand a slightly stronger adversary – called a *semi-malicious* adversary [1] – which is allowed to maliciously choose its random tape. This will be crucially exploited in the next step, because without it, one will need to enforce honest random-tapes for all the parties (using n-way coin-tossing-in-the-well) which would incur $\Omega(n)$ communication already.

From Semi-Malicious to Malicious Security

A natural approach to achieve security against malicious adversaries is to use the GMW paradigm [20]. Roughly, in the GMW compiler, each party first commits to its input and random tape. Later, whenever a party sends a message of a semi-malicious protocol,[4] it also proves in ZK to *all* the other parties that it is behaving correctly w.r.t. the committed input and random tape.

The GMW commit-and-prove methodology is problematic in our setting since we cannot allow a party to talk to all other parties (directly or indirectly through the other nodes). Yet, in order to achieve security, each honest party must verify not just that its neighbors behave correctly, but that *all* corrupt parties (many of whom may not directly interact with any honest party) behaved honestly. A priori, these may seem to be contradictory goals.

We address all of these challenges by presenting a new generic compiler for *Verifiable Protocol Execution* (VPE), modeled as a functionality $\mathcal{F}_{\mathsf{vpe}}$. Our protocol Π_{vpe} for implementing

[4] The standard GMW compiler is defined for semi-honest protocols and also involves a coin-tossing step. Here, we consider a natural variant that works for semi-malicious protocols.

$\mathcal{F}_{\mathsf{vpe}}$ asymptotically preserves the per-party communication and computational complexity (up to a multiplicative factor polynomial in the security parameter) of the underlying semi-malicious protocol. We construct Π_{vpe} from two main ingredients: (1) a new commitment protocol that allows the parties to compute a succinct "aggregate" commitment over the inputs and randomness of all of the parties. (2) ZK-SNARKs with a strong extraction property as well as simulation-soundness to ensure that adversary cannot prove false statements even upon receiving simulated proofs. We refer the reader to the full version for details on our commitment protocol. Here, we discuss our use of ZK-SNARKs.

ID-Based Simulation-Extractable ZK-SNARKs

We rely on ZK-SNARKs to let parties provide not just proofs of correctly computing their own messages, but also of having verified previous proofs *recursively*. This use of SNARKs for recursive verification resembles prior work on proof-carrying data [8, 3]. The key difference is that proof-carrying data only addresses correctness of computation, whereas in our setting, we are also concerned with privacy. In particular, in order to argue security, we also require these proofs to be simulation-sound with extractability (or simply *simulation-extractable*) [30, 31], which presents a significant additional challenge.

The core challenge in constructing simulation-extractable ZK-SNARKs (SE-ZK-SNARKs) arises from the inherent limitation that extraction from the adversary must be non-black-box (since the size of the extracted witness is larger than the proof itself), but the adversary receives simulated proofs which he cannot directly produce on his own. Indeed, for this reason SE-ZK-SNARKs are impossible to achieve with strong universal composability (UC) security [23]. To reduce the security of an SE-ZK-SNARK construction to an underlying knowledge assumption (such as standard SNARKs), one must thus either (a) start with an assumption that guarantees non-black-box extraction even in the presence of an oracle (which can be problematic [16]), or (b) somehow in the reduction be able to provide *the code* to answer the adversary's simulated proof queries, without voiding the reduction by including the simulation trapdoor itself.

Two recent works have presented constructions of SE-ZK-SNARKs, each adopting a different approach. Groth and Maller [21] embody approach (a), constructing full SE-ZK-SNARKs from a new specific pairing-based knowledge assumption which assumes extraction in the presence of black-box access to an oracle with the trapdoor. Alternatively, Garman *et al.* [17] take approach (b), basing their construction on standard SNARKs; however, their construction is only applicable to a restricted security model where the statements on which the adversarial prover requests simulated proofs are fixed in advance (in which case these proofs can be hardcoded in the reduction). The case where the adversary's queries are chosen *adaptively* as a function of previously simulated proofs (which we need for our transformation) is not currently addressed in this setting.

We provide a new solution for handling adaptive queries, without relying upon oracle-based assumptions as in [21]. We consider an ID-based notion of SE-ZK-SNARKs, where each proof is generated with respect to an identity (chosen from a set of identities that are fixed in advance). In our definition, the adversary must fix a set ID^* of "honest" identities in advance and can then receive simulated proofs on adaptively chosen statements w.r.t. identities from this set. It must then come up with an accepting proof for a new statement w.r.t. an identity $\mathsf{id} \notin \mathsf{ID}^*$.

We show how to transform any SNARK argument system into an ID-based SE-ZK-SNARKs by relying on only standard cryptographic assumptions. Very roughly, in our construction, it is possible to "puncture" the trapdoor trap for the CRS w.r.t. an identity set

ID^*. A punctured trapdoor $\mathsf{trap}_{\mathsf{ID}^*}$ can only be used to simulate the proofs w.r.t. identities $\mathsf{id} \in \mathsf{ID}^*$, but cannot be used to simulate proofs w.r.t. identities $\mathsf{id} \notin \mathsf{ID}^*$. Using such a punctured trapdoor, we are able to successfully implement approach (b) in the adaptive setting. We implement this idea by using identity-based signatures, which can be readily constructed using certificate chains from a standard signature scheme.

Ultimately, we obtain recursively verifiable ID-based SE-ZK-SNARK generically from signatures and (standard) SNARKs with an "additive extraction overhead." While the latter is a relatively strong requirement [22], such primitives have been considered in prior work [10, 3] and appears to be as justified as the standard SNARK assumption.

Auxiliary Inputs

In order to achieve the standard MPC security, we require SE-ZK-SNARKs that allow extraction even w.r.t. adversaries with auxiliary inputs from arbitrary distribution, which in turn requires SNARKs with the same extraction property. However, even assuming SNARKs which are secure only with respect to specific auxiliary information distribution, our construction obtains MPC that is secure with respect to restricted auxiliary information (see the full version for details).

1.2 Related Work

Communication complexity models

The vast majority of study in communication complexity (c.f. [24]) focuses on the setting of only two parties, in which case the total and bottleneck complexities of protocols align (asymptotically). In the multi-party setting, several models are considered regarding how the input to f begins initially distributed among the players. The most common such models are the "number-on-forehead" model, in which parties begin holding all inputs except their own, and the model considered in this work (as is standard in MPC), frequently known as the "number in hand" model, where each party begins with his own input. In all cases, the "communication complexity" within the given model refers to the total communication of all parties.

Communication complexity of MPC

Communication complexity of secure multiparty computation (MPC) has been extensively studied over the years. Communication complexity preserving compilers from insecure to secure protocols were introduced in the 2-party setting by [28]. The setting of MPC with many parties was first predominantly considered in the line of work on scalable MPC [11, 12]. Here the focus was on optimizing the complexity as a function of the circuit size $|C|$, and the resulting n-party protocols have per-party communication $\tilde{O}(|C|/n) + \mathrm{poly}(n)$. Some of these works explicitly achieve load-balancing (e.g., [14, 4]), a goal similar in spirit to bottleneck complexity, where the complexity of the protocol is evenly distributed across the parties. To the best of our knowledge, however, the $\mathrm{poly}(n)$ term in the per-party communication complexity is $\Omega(n)$ in all works aside from [33], which achieves $\tilde{O}(|C|/n)$ *amortized* per-party communication but $\tilde{O}(|C|/n + n)$ *bottleneck* complexity (due to its dependence on [7]).

Communication Locality

A related notion to bottleneck complexity is communication locality [5]. The locality of a party is the number of total other parties it must communicate with throughout the protocol, and the locality of the protocol is the worst case locality of any party. In [5], Boyle et al.

studied locality in secure MPC and showed (based on various computational assumptions) that any efficiently computable function has a polylog(n)-locality secure MPC protocol.

Lower bounds on MPC communication complexity

As discussed, lower bounds on standard multi-party communication complexity cannot directly imply meaningful lower bounds on bottleneck complexity, as no such bound can exceed $\Omega(n)$ (attainable by all parties sending their input to a single party), but this implies only a bound of $\Omega(1)$ bottleneck complexity. For *secure* computation, in [13], Damgård et al. showed that securely evaluating a circuit of m multiplication gates requires $\Omega(n^2 m)$ total communication in the information-theoretic security setting. This implies a super-linear lower bound for bottleneck complexity in their setting. We note, however, that their lower bound does not apply to us, as we consider computational security, and further, their lower bound does not apply to the setting where the number of parties is larger than the security parameter.

2 Preliminaries and Definitions

By $x \in_R \mathbb{Z}_q^n$ we denote that x is uniformly sampled from $\mathbb{Z}_q^n$, and by $x \leftarrow D$ we denote that x is sampled from a distribution D. By $\overset{c}{\approx}$ we denote computational indistinguishability. We denote an N-party additive secret sharing of $x \in \mathbb{Z}_q$ by $[x]_q^N$. That is, each P_i owns $x_i \in \mathbb{Z}_q$ such that $x = \sum_{i \in [N]} x_i$. When it is clear, we write $[x]$ for brevity.

Communication Model

We consider a synchronous network among n parties, $P_1, \cdots, P_n$, that allows secure communication between (some) pairs of parties; the channels are authenticated and leak nothing except the number of bits in each message.

An n-party protocol π in this model is a "next-message function," that takes as input the round number t, two identities $i, j \in [n]$ and the view of P_i, and outputs the message from P_i to P_j in round t. The view of a party consists of its input, random-tape, and all the messages received in prior rounds of the protocol. If the next-message function is invoked with the keyword out instead of a receiver, π generates the output for the "sender" (for simplicity, we shall restrict ourselves to protocols that produce output only on termination).

We shall require that in a protocol, the message on any edge (i, j) at any round t is encoded using a *prefix-free code* that is agreed up on between the sender and the receiver. Adopting this model precludes communication by not sending any bits. [5]

Security for MPC

We use a standard simulation-based security definition for MPC protocols in the *standalone security* model. We consider two notions of corruptions: (1) active adversaries, who may arbitrarily deviate from the protocol strategy, and (2) semi-malicious adversaries, who follow the protocol instructions but may choose their random tapes arbitrarily. Both of these

[5] While one may argue that it is reasonable to allow zero-cost communication in this manner, it can be abused to communicate large amounts of information at the cost of a single bit, by using a large number of rounds. Further, such signalling cannot be used if multiple such protocols are executed concurrently. Also, practically, the prefix-free communication is more amenable to implementing a synchronous model over an asynchronous network, as delays will not be mistaken for shorter (or empty) messages.

adversaries can abort the execution whenever they choose. We shall also consider a notion of MPC security in which the auxiliary information given to the adversary at the beginning of the protocol is restricted to be from a class of distributions. We refer the reader to the full version for further details.

2.1 Bottleneck Complexity

We introduce a new *per-party* communication metric for distributed computations.

▶ **Definition 3** (Bottleneck Complexity of Protocol)**.** The *individual communication complexity* of a party P_i in an n-party protocol π, denoted as $\mathcal{CC}_i(\pi)$, is the expected number of bits sent or received by P_i in an execution of π, with worst-case inputs.

The *bottleneck complexity* ($\mathcal{BC}$) of an n-party protocol π is the worst-case communication complexity of any party. That is, $\mathcal{BC}(\pi) = \max_{i\in[n]} \mathcal{CC}_i(\pi)$.

▶ **Definition 4** (Bottleneck Complexity of Function)**.** The *bottleneck complexity of an n-input function f* is the minimum value of $\mathcal{BC}(\pi)$ when quantified over all n-party distributed protocols π which correctly evaluate f.

Analogously, we define the *MPC bottleneck complexity* of f as the minimum $\mathcal{BC}(\pi)$ quantified over all n-party protocols π which *securely* evaluate f.

Admissible Protocols

We will show techniques that transform general (insecure) protocols to secure ones. Here we define the required minimal assumption of the original protocols, which we refer to as admissibility. Roughly, a protocol is admissible if its next-message function is polynomial-time computable and it has a fixed communication pattern. Below $\mathbb{Z}_+$ denotes non-negative integers.

▶ **Definition 5** (Admissible Protocol)**.** Let f be a polynomial function, k be a security parameter, and let $\pi = \{\pi_1, ..., \pi_n\}$ be a possibly randomized n-party protocol, where π_i is a next message function of P_i. Let $x = \{x_1, ..., x_n\}$ and $r = \{r_1, ..., r_n\}$ be the input set and the random string set respectively. Denote $\left\{m^t_{i,j}(x,r)\right\}^{t\in[T]}_{i,j\in[n]}$ as the set of the messages generated by $\pi(x,r)$, and let $|m^t_{i,j}(x,r)| \in [0, f(k)]$ be the length of message from P_i to P_j at time t.[6] We say π is admissible if it satisfies the following two conditions:

- **Polynomial-Time Computable:** For each i, next-message function π_i is expressed by a circuit of fixed polynomial-size in $|x_i| + |r_i|$, with a universally bounded depth.
- **Fixed Communication Pattern:** A protocol π is said to have a *fixed communication pattern* if, irrespective of the input and random-tapes of the parties, the total number of rounds $t_{\max}$ is fixed and there is a function len : $[t_{\max}] \times [n] \times [n] \to \mathbb{Z}_+$ that maps (t, i, j) to the length of message (possibly 0) from P_i to P_j at round t as determined by π for any view of P_i.

Note that above we allow randomized protocols, as some interesting low bottleneck complexity protocols (e.g., those derived from streaming algorithms) tend to be randomized.

---

[6] Precisely, for each $i \in [n], t \in [T]$, $\{m_{i,j}(x,r)\}^t_{j\in[n]} \leftarrow \pi_i\left(x_i, r_i, \{m_{j,i}(x,r)\}^{t\in[t-1]}_{j\in[n]}\right)$

3 Lowerbound on Bottleneck Complexity of Distributed Functions

In this section we show that for most functions f on n inputs (each input could be as short as 1 bit, and the output a single bit delivered to a single party), for any distributed computation protocol π that implements f, the (incoming) bottleneck complexity $\mathcal{BC}(\pi)$ is at least $n - O(\log n)$ bits. In fact, this holds true even without any security requirement. This is tight in the sense that, even with a security requirement, there is a protocol in which only one party has individual communication complexity $\Omega(n)$, and all others have communication proportional to their inputs and outputs (with a multiplicative overhead independent of the number of parties).

This is somewhat surprising since many interesting functions do have protocols with constant communication complexity. As mentioned before, any (possibly randomized) function which has a streaming algorithm or a sub-quadratic sized circuit (with small fan-in gates) gives rise to low botteleneck complexity protocols.

To show our lower-bound, we need to therefore rely on functions with roughly a quadratic lower-bound on circuit size. Given the current lack of explicit examples of such functions, we present an *existential result*, and leave it as a conjecture that there are n-bit input boolean functions with polynomial sized circuits with bottleneck communication complexity of $\tilde{\Omega}(n)$. For simplicity, we discuss the case of perfectly correct protocols, but as we shall point out, a small constant probability of error does not change the result significantly. This result says that there is a function (in fact, most functions) such that the best bottleneck complexity is almost achieved by the trivial protocol, in which one party receives the inputs of all the other $n - 1$ parties and carries out the computation locally. We prove the following in the full version.

▶ **Theorem 6.** *$\exists f : \{0,1\}^{k \times n} \to \{0,1\}$ such that any n-party, each with k bits input, distributed computation protocol that computes f correctly will have at least one party receiving at least $(n-1)k - O(\log nk)$ bits in the worst-case.*

4 Incremental FHE

We define and implement a new notion of incremental FHE (IFHE), which is used within our main positive result. We start by providing syntax and security definitions for IFHE in Section 4.1. We describe our construction of IFHE in Section 4.2. Our construction builds upon the FHE scheme of Gentry, Sahai and Waters (GSW) [19]; we refer the reader to the full version for discussion on the specific properties of their scheme that we rely upon.

4.1 Definitions

(Leveled) Fully Homomorphic Encryption

A fully homomorphic encryption (FHE) scheme consists of algorithms (KEYGEN, ENCRYPT, EVAL, DECRYPT), where (KEYGEN, ENCRYPT, DECRYPT) constitute a semantically secure public-key encryption scheme, and EVAL refers to the homomorphic evaluation algorithm on ciphertexts. An ℓ-leveled FHE scheme supports homomorphic evaluation of circuits of depth at most ℓ.

Incremental FHE

An IFHE scheme is defined similarly to threshold FHE [1], with the following key modifications: a "joint" public key can be computed by incrementally combining shares provided by different parties in an arbitrary order. Similarly, a ciphertext w.r.t. the joint public key can be

decrypted by incrementally combining partial decryption shares provided by parties in an arbitrary order. Crucially, the intermediate keys and partial decryption values must be *succinct*.

For technical reasons, it is convenient to describe the joint decryption procedure via three sub-algorithms: A procedure PreDec which pre-processes a homomorphically evaluated ciphertext to be safe for joint decryption; PartDec run by each individual party on a ciphertext (with his share of the secret key) to generate his contribution toward the decryption; and CombineDec which combines the outputs of PartDec from each party for a given ciphertext to reconstruct the final decrypted output. In addition to standard semantic security, we also require the output of PartDec to hide information about the secret key share that was used; this is captured by the Simulatability of Partial Decryption property below.

▶ **Definition 7** (Incremental FHE)**.** An *incremental fully homomorphic encryption (IFHE)* scheme is an FHE scheme with an additional algorithm IFHE.CombineKeys and with Decrypt replaced by three algorithms IFHE.PreDec, IFHE.PartDec and IFHE.CombineDec. By PK_S we denote a combined public key of a subset $S \subseteq [n]$ of parties. Particularly, $\mathrm{PK}_{\{i\}} = \mathrm{PK}_i$ is generated by P_i using the algorithm Keygen, and $\mathrm{PK} = \mathrm{PK}_{[n]}$ is the final public key. Similarly, by v_S we denote a combined decryption, and by v_i when $S = \{i\}$. For the completeness of notations, let PK_S and v_S be empty strings when $S = \emptyset$. We describe the syntax of the four algorithms as follows:

- IFHE.CombineKeys$(\mathrm{PK}_S, \mathrm{PK}_T)$: On input 2 combined public keys $\mathrm{PK}_S, \mathrm{PK}_T$, where $S \cap T = \emptyset$, output a combined public key $\mathrm{PK}_{S\cup T}$.
- IFHE.PreDec$(\mathrm{PK}, \mathbf{C})$: On input a final public key PK and a ciphertext $\mathbf{C}$, sample a public random $\mathbf{R}$, and output a re-randomized ciphertext $\mathbf{C}'$ of the same plaintext.
- IFHE.PartDec$(\mathrm{PK}, \mathrm{SK}_i, \mathbf{C})$: On input a final public key PK, ith secret key SK_i, ciphertext $\mathbf{C}$, output a partial decryption v_i.
- IFHE.CombineDec(v_S, v_T): On input 2 partial decryptions v_S, v_T, where $S \cap T = \emptyset$, if $|S \cup T| < n$, output a partial decryption $v_{S\cup T}$; otherwise, output a plaintext y as the final decryption.

Also, we require the following additional properties:

Efficiency: There are polynomials $poly_1(\cdot), poly_2(\cdot)$ such that for any security parameter λ and any $S \subseteq [n], S \neq \emptyset$, $|\mathrm{PK}_S| = poly_1(\lambda)$ and $|v_S| = poly_2(\lambda)$.

Correctness: Given a set of plaintexts and a circuit to evaluate, the correctness of IFHE says that the FHE evaluation of the circuit over the ciphertexts can always be decrypted to the correct value, where the ciphertexts are encryption of plaintexts using a single combined public key.

Furthermore, by "Incremental" FHE, we mean that the final combined public key as well as the final combined decryption can be formed in an arbitrary incremental manner. We defer the formal definition of correctness to the full version.

Semantic security under Combined Keys (against Semi-Malicious Adversary): Given the parameters prepared in the initial setup, the (corrupted) parties $\{P_j\}_{j\neq i}$, instead of using random strings to compute $\{\mathrm{PK}_i, \mathrm{SK}_i\}_{j\neq i}$, can use an arbitrary string to generate $\{\mathrm{PK}_i, \mathrm{SK}_i\}_{j\neq i}$. Then as long as an honest party generates $(\mathrm{PK}_i, \mathrm{SK}_i)$ independently, the encryption using the final combined public key $(\mathrm{PK}_{[n]}, \mathrm{SK}_{[n]})$ is semantically secure. Formally, we say that the IFHE scheme has semantic security under combined keys if the advantage $\Pr[\beta' = \beta] - 1/2$ in the following experiment is negligible for all PPT Adv.

1. $(\mathsf{params}) \leftarrow \mathrm{Setup}(1^\lambda, 1^d)$

2. $\forall j \neq i$, Adv computes $(\text{PK}_j, \text{SK}_j)$ according to $\text{KEYGEN}(\mathsf{params})$ but replaces the randomly sampled string by a chosen one. Then Adv computes a combined key $\text{PK}_{[n]\setminus\{i\}}$ according to COMBINEKEYS, picks $x \in \{0,1\}$ and sends $(\text{PK}_{[n]\setminus\{i\}}, x)$ to the challenger.
3. The challenger computes $(\text{PK}_i, \text{SK}_i) \leftarrow \text{KEYGEN}(\mathsf{params})$, $\text{PK} \leftarrow \text{COMBINEKEYS}(\text{PK}_i, \text{PK}_{[n]\setminus\{i\}})$, and chooses a random bit $\beta \xleftarrow{\$} \{0,1\}$. If $\beta = 0$, it lets $\mathbf{C} := \text{ENCRYPT}(\text{PK}, 0)$, and else $\mathbf{C} := \text{ENCRYPT}(\text{PK}, x)$, and sends $\mathbf{C}$ to Adv.
4. Finally Adv outputs a bit β'.

Simulatability of Partial Decryption: There is a PPT simulator SIM, s.t. for every combined public key PK, ciphertext $\mathbf{C}'$ of a plaintext y, index $i \in [n]$:

$$\left\{\text{PK}, \mathbf{C}', \{\text{SK}_j\}_{j\in[n]\setminus\{i\}}, v_i\right\} \overset{c}{\approx} \left\{\text{PK}, \mathbf{C}', \{\text{SK}_j\}_{j\in[n]\setminus\{i\}}, v_i'\right\},$$

where $v_i \leftarrow \mathsf{IFHE}.\text{PARTDEC}(\text{PK}, \text{SK}_i, \mathbf{C}')$ and $v_i' \leftarrow \text{SIM}(\text{PK}, \{\text{SK}_j\}_{j\in[n]\setminus\{i\}}, y, \mathbf{C}')$.

4.2 Construction of IFHE

We present an IFHE scheme building on the FHE scheme of Gentry et al. [19]. The SETUP, KEYGEN, ENCRYPT, and EVAL parts are the same as those of [19], while COMBINEKEYS, PREDEC, PARTDEC and COMBINEDEC parts are new. The algorithms are described as follows:

An IFHE scheme.

- **Setup:** $(\mathsf{params}) \leftarrow \mathsf{IFHE}.\text{SETUP}(1^\lambda, 1^d)$. Same as GSW, where $\mathsf{params} = (q, n, m, \chi, B_\chi, \mathbf{B})$.
- **Key Generation:** $(\text{PK}_i, \text{SK}_i) \leftarrow \mathsf{IFHE}.\text{KEYGEN}(\mathsf{params})$. Same as GSW, where $\text{PK}_i = (\mathbf{B}, \mathbf{b}_i)$ and $\text{SK}_i = \mathbf{t}_i \equiv (-\mathbf{s}_i, 1)$.
- **Combining Keys:** $\text{PK}_{S\cup T} \leftarrow \mathsf{IFHE}.\text{COMBINEKEYS}(\text{PK}_S, \text{PK}_T)$
  On input $\text{PK}_S = (\mathbf{B}, \mathbf{b}_S)$ and $\text{PK}_S = (\mathbf{B}, \mathbf{b}_T)$, output $\text{PK}_{S\cup T} = (\mathbf{B}, \mathbf{b}_S + \mathbf{b}_T)$. Particularly $\text{PK}_{[n]}$ is abbreviated as PK or a matrix $\mathbf{A}$.
- **Encryption:** $\mathbf{C}_i \leftarrow \mathsf{IFHE}.\text{ENCRYPT}(\text{PK}, x_i)$. Same as GSW.
- **Evaluation:** $\mathbf{C} \leftarrow \mathsf{IFHE}.\text{EVAL}(\mathbf{C}_1, ..., \mathbf{C}_\tau; f)$. Same as GSW.
- **Preparing Decryption:** $\mathbf{C}' \leftarrow \mathsf{IFHE}.\text{PREDEC}(\text{PK}_i, \mathbf{C})$
  On input $\text{PK} = \mathbf{A}$ and $\mathbf{C} \in \mathbb{Z}_q^{n\times m}$, sample a public random matrix $\mathbf{R}$ in $\{0,1\}^{m\times m}$ and output $\mathbf{C}' = \mathbf{C} + \mathbf{AR}$.
- **Partial Decryption:** $v_i \leftarrow \mathsf{IFHE}.\text{PARTDEC}(\text{PK}_i, \text{SK}_i, \mathbf{C}')$
  On input $\text{PK} = \mathbf{A}$, $\text{SK}_i \equiv \mathbf{t}_i \equiv (-\mathbf{s}_i, 1)$, and $\mathbf{C}' \in \mathbb{Z}_q^{n\times m}$, sample $\mathbf{e}_i' \leftarrow \chi^m$, set $\mathbf{t}_i' = t_i$ if $i = 1$ and $\mathbf{t}_i' = (-\mathbf{s}_i, 0)$ if $i > 1$, and output $v_i = (\mathbf{t}_i'\mathbf{C}' - \mathbf{e}_i')\mathbf{G}^{-1}(\mathbf{w}^T)$, where $\mathbf{w} = (0, ..., 0, \lceil q/2 \rceil) \in \mathbb{Z}_q^n$.
- **Combining Decryption:** $\mathsf{IFHE}.\text{COMBINEDEC}(v_S, v_T)$
  On input two partial decryptions v_S, v_T with $S \cap T = \emptyset$, compute a partial decryption $v_{S\cup T} = v_S + v_T$. For $|S \cup T| < n$, output $v_{S\cup T}$; for $|S \cup T| = n$, output a plaintext $y = \left\lfloor \frac{v}{q/2} \right\rceil$.

---

5 Summary of Further Results

Due to space limitations we defer the remaining technical sections to the full version. We first demonstrate how to convert an arbitrary admissible multi-party distributed protocol Π (as per Definition 5) for computing a function f to a protocol Π_{sm} for computing f secure against

semi-malicious adversaries, while preserving *per-party* computation and communication. Note that as per Definition 5 the communication pattern of the starting protocol Π can be arbitrary, but we require that it be fixed (i.e., not data dependent).

▶ **Theorem 8.** *Let* IFHE *be an incremental FHE scheme, and* Π *be an* n*-party protocol for evaluating a function* f *with fixed communication pattern. Then there exists a protocol* Π_{sm} *that securely evaluates* f *against up to* $(n-1)$ semi-malicious *corruptions, preserving the per-party computation and communication requirements of* Π *up to* $\mathrm{poly}(\lambda)$ *multiplicative factors (independent of the number of parties* n*). Moreover, the communication pattern of* Π_{sm} *is identical to that of* Π *plus two additional traversals of a communication spanning tree of* Π*.*

At a very high level, the parties will run a one-pass protocol to (incrementally) construct a joint key for the incremental FHE scheme, then execute the original protocol Π underneath FHE encryption, and finally run one more pass to (incrementally) decrypt.

Next, we give a general compiler to transform the above protocol into a maliciously-secure MPC protocol, while preserving the bottleneck complexity. Our compiler relies upon multiple cryptographic ingredients, most notably, ID-based simulation-extractable succinct non-interactive arguments of knowledge (ZK-SNARKs) – a notion that we define and construct in this work (see the full version).

▶ **Theorem 9.** *Let* Π_{sm} *be an MPC protocol that securely evaluates a functionality* f *against* semi-malicious *corruptions, as in Theorem 8. Then, assuming the existence of an ID-based simulation-extractable succinct non-interactive arguments of knowledge, non-interactive commitment schemes and a family of collision-resistant hash functions, there exists a compiler that transforms* Π_{sm} *into another MPC protocol* Π *in the (bare) public-key and common reference string model such that* Π *computes the same functionality* f *and preserves the per-party computation and communication of* Π_{sm} *up to* $\mathrm{poly}(\lambda)$ *multiplicative factors (independent of the number of parties* n*).*

We remark that in the above theorem, the ID-based SE-ZK-SNARK is w.r.t. arbitrary auxiliary inputs. We stress, however, that in order to achieve a weaker MPC definition where the adversary only receives restricted auxiliary inputs, the requirement on the SE-ZK-SNARK can be similarly relaxed as well.

References

1. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *EUROCRYPT*, pages 483–501, 2012.
2. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 326–349, 2012.
3. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 111–120, 2013.
4. Elette Boyle, Kai-Min Chung, and Rafael Pass. Large-scale secure computation: Multi-party computation for (parallel) RAM programs. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 742–762, 2015.

5 Elette Boyle, Shafi Goldwasser, and Stefano Tessaro. Communication locality in secure multi-party computation: how to run sublinear algorithms in a distributed setting. In *Proceeding TCC'13 Proceedings of the 10th theory of cryptography conference on Theory of Cryptography*, pages 356–376, 2013.

6 Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pages 190–213, 2016.

7 Nicolas Braud-Santoni, Rachid Guerraoui, and Florian Huc. Fast byzantine agreement. In *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 57–64, 2013.

8 Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 310–331, 2010.

9 Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 630–656, 2015.

10 Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, pages 54–74, 2012.

11 Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, pages 501–520, 2006.

12 Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam D. Smith. Scalable multiparty computation with nearly optimal work and resilience. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, pages 241–261, 2008.

13 Ivan Damgård, Jesper Buus Nielsen, Antigoni Polychroniadou, and Michael Raskin. On the communication required for unconditionally secure multiplication. In *Crypto'16*, pages 459–488, 2016.

14 Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Quorums quicken queries: Efficient asynchronous secure multiparty computation. In *Distributed Computing and Networking - 15th International Conference, ICDCN 2014, Coimbatore, India, January 4-7, 2014. Proceedings*, pages 242–256, 2014.

15 Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, pages 93–122, 2016.

16 Dario Fiore and Anca Nitulescu. On the (in)security of snarks in the presence of oracles. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, pages 108–138, 2016.

17 Christina Garman, Matthew Green, and Ian Miers. Accountable privacy for decentralized anonymous payments. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, pages 81–98, 2016.

18 Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi, editors. *Data Stream Management - Processing High-Speed Data Streams*. Data-Centric Systems and Applications. Springer, 2016. doi:10.1007/978-3-540-28608-0.

19 Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Crypto'13*, pages 75–92, 2013.

20 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *STOC*, 1987.

21 Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 581–612, 2017.

22 Divya Gupta and Amit Sahai. On constant-round concurrent zero-knowledge from a knowledge assumption. In *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, pages 71–88, 2014.

23 Ahmed E. Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, T.-H. Hubert Chan, Charalampos Papamanthou, Rafael Pass, Abhi Shelat, and Elaine Shi. How to use snarks in universally composable protocols. *IACR Cryptology ePrint Archive*, 2015:1093, 2015. URL: `http://eprint.iacr.org/2015/1093`.

24 Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.

25 Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1219–1234, 2012.

26 Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43:9–20, 2014.

27 Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, pages 735–763, 2016.

28 Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 590–599, 2001.

29 Chris Peikert and Sina Shiehian. Multi-key FHE from lwe, revisited. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, pages 217–238, 2016.

30 Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 543–553, 1999.

31 Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 566–598, 2001.

32 Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164, 1982.

33 Mahdi Zamani, Mahnush Movahedi, and Jared Saia. Millions of millionaires: Multiparty computation in large networks. *IACR Cryptology ePrint Archive*, 2014:149, 2014.

Revisiting Frequency Moment Estimation in Random Order Streams

Vladimir Braverman[1]
Johns Hopkins University
vova@cs.jhu.edu

Emanuele Viola
Northeastern University
viola@ccs.neu.edu

David P. Woodruff
Carnegie Mellon University
dwoodruf@cs.cmu.edu

Lin F. Yang[2]
Princeton University
lin.yang@princeton.edu

Abstract

We revisit one of the classic problems in the data stream literature, namely, that of estimating the frequency moments F_p for $0 < p < 2$ of an underlying n-dimensional vector presented as a sequence of additive updates in a stream. It is well-known that using p-stable distributions one can approximate any of these moments up to a multiplicative $(1+\epsilon)$-factor using $O(\epsilon^{-2}\log n)$ bits of space, and this space bound is optimal up to a constant factor in the turnstile streaming model. We show that surprisingly, if one instead considers the popular random-order model of insertion-only streams, in which the updates to the underlying vector arrive in a random order, then one can beat this space bound and achieve $\tilde{O}(\epsilon^{-2}+\log n)$ bits of space, where the $\tilde{O}$ hides $\mathrm{poly}(\log(1/\epsilon)+\log\log n)$ factors. If $\epsilon^{-2} \approx \log n$, this represents a roughly quadratic improvement in the space achievable in turnstile streams. Our algorithm is in fact deterministic, and we show our space bound is optimal up to $\mathrm{poly}(\log(1/\epsilon)+\log\log n)$ factors for deterministic algorithms in the random order model. We also obtain a similar improvement in space for $p = 2$ whenever $F_2 \gtrsim \log n \cdot F_1$.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases Data Stream, Frequency Moments, Random Order, Space Complexity, Insertion Only Stream

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.25

Related Version A full version of the paper can be found at `https://arxiv.org/abs/1803.02270`.

[1] This material is based upon work supported in part by the National Science Foundation under Grants No. 1447639, 1650041 and 1652257, Cisco faculty award, and by the ONR Award N00014-18-1-2364.
[2] This material is based upon work supported in part by National Science Foundation under Grant No. 1447639, 1650041 and 1652257. Work was done while the author was at Johns Hopkins University.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 25; pp. 25:1–25:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Analyzing massive datasets has become an increasingly challenging problem. Data sets, such as sensor networks, stock data, web/network traffic, and database transactions, are collected at a tremendous pace. Traditional algorithms that store an entire dataset in memory are impractical. The *streaming model* has emerged as an important model for coping with massive datasets. Streaming algorithms are typically randomized and approximate, making a single pass over the data and using only a small amount of memory.

A well-studied problem in the streaming model is that of estimating the frequency moments of an underlying vector. Formally, in an *insertion-only stream*, the algorithm is presented with a sequence of integers $\langle a_1, a_2, \ldots, a_m \rangle$ from a universe $[n]$. A *turnstile stream* is defined similarly except integers may also be removed from the stream. The p-th frequency moment of the stream is defined to be $F_p = \sum_{i \in [n]} f_i^p$, where f_i is number of times integer i occurs in the stream, i.e., its frequency. The quantity F_p is a basic, yet very important statistic of a dataset (e.g. [1]). For example, interpreting 0^0 as 0, F_0 is equal to the number of distinct items in the stream. F_2 measures the variance and can be used to estimate the size of a self-join in database applications. It also coincides with the (squared) Euclidean norm of a vector and has applications in geometric and linear algebraic problems on streams. For other non-integer $p > 0$, F_p can serve as a measure of the entropy or skewness of a dataset, which can be useful for query optimization.

In their seminal paper, Alon, Matias & Szegedy [2] introduced the study of frequency moments in the streaming model. Nearly two decades of research have been devoted to the space and time complexity of this problem. An incomplete list of papers on frequency moments includes [3, 8, 11, 18–20, 23], and [7]; please also see the references therein. In the turnstile model, $\Theta(\epsilon^{-2} \log(mn))$ bits of space is necessary and sufficient for a randomized one-pass streaming algorithm to obtain a $(1 \pm \epsilon)$-approximation to F_p for $0 < p \leq 2$ [21]. Here, by $(1 \pm \epsilon)$-approximation, we mean that the algorithm outputs a number $\tilde{F}_p$ for which $(1 - \epsilon)F_p \leq \tilde{F}_p \leq (1 + \epsilon)F_p$. For larger values of p, i.e., $p > 2$, the memory required becomes polynomial in n rather than logarithmic [5, 11, 19].

In this paper, we study the frequency moment estimation problem in the *random-order model*, which is a special case of insertion streams in which the elements in the stream occur in a uniformly random order and the algorithm sees the items in one pass over this random order. This model was initially studied by Munro and Paterson [22], which was one of the initial papers on data streams in the theory community. Random-order streams occur in many real-world applications and are studied in, e.g., [10, 13, 15] and the references therein. It has been shown that there is a considerable difference between random order streams and general arbitrary order insertion streams for problems such as quantile estimation [10, 16]. Notice that [12] studies the frequency moment problem for stochastic streams, in which data points are generated from some distribution. That model is different from ours, but when conditioning on the realizations of the values of each point, the stream is exactly in random order. Therefore, our upper bounds are applicable to that model as well.

However, there is a gap in our understanding for the important problem of F_p-estimation, $0 < p \leq 2$, in the random order model. On the one hand there is an $\Omega(\epsilon^{-2})$ bits of space lower bound [9]. On the other hand, the best upper bound we have is the same as the best upper bound in the turnstile model, namely $O(\epsilon^{-2} \log n)$ bits [2, 20, 21]. In practice it would be desirable to obtain $O(\epsilon^{-2} + \log n)$ bits rather than $O(\epsilon^{-2} \log n)$ bits, since if ϵ is very small this can lead to considerable savings. For example, if $\epsilon^{-2} \approx \log n$, it would represent a roughly quadratic improvement. The goal of this work is to close this gap in our understanding of the memory required for F_p-estimation, $0 < p \leq 2$, in the random order model. Note that in all our bounds, we apply the convention that $m = \text{poly}(n)$.

1.1 Our Contribution

In this paper, we make considerable progress on understanding the space complexity of F_p estimation with $0 < p \leq 2$, in random order streams. Specifically,

- for F_2, we show that there exists a simple, and in fact deterministic, one-pass algorithm using $\mathcal{O}(\epsilon^{-2}+\log n)$ bits of space to output a $(1\pm\epsilon)$ approximation, provided $F_2 \geq m\cdot\log n$,
- for F_p with $p \in (0,2)\backslash\{1\}$, we obtain a one-pass deterministic algorithm to obtain a $(1 \pm \epsilon)$ approximation using $\tilde{O}(\epsilon^{-2} + \log n)$ bits of space[3]. We also show that this space complexity is optimal for deterministic algorithms in the random order model, up to $\text{poly}(\log(1/\epsilon)) + \text{poly}(\log\log n)$ factors. Note that for the case $p = 1$, F_p is the length of the stream, which can be computed using $O(\log n)$ bits of memory.

1.2 Our Techniques

For F_2, we partition the stream updates into a sequence of small blocks for which each block can be stored using $\mathcal{O}(\epsilon^{-2})$ bits. We then construct an unbiased estimator by counting the number of "pairs" of updates that belong to the same universe item. The counting can be done exactly and with only $\mathcal{O}(\log n)$ bits of space in each small block. We further show that if $F_2 \geq \log n \cdot F_1$, we can obtain the desired concentration by averaging the counts over all blocks. The analysis of the concentration is by constructing a Doob martingale over the pairs and applying Bernstein's inequality.

For F_p with $0 < p < 2$ our algorithm is considerably more involved. We first develop a new reduction from estimating F_p to finding ℓ_p heavy hitters (an ℓ_p heavy hitter is an item with frequency comparable to $F_p^{1/p}$ of the stream) and then we establish the heavy hitter algorithm. Both our new reduction and the heavy hitter-finding algorithm are redesigned over the existing ones to allow us to obtain better space complexity. Our heavy hitter finding algorithm is similar to that of [6]. However, we need to be more careful so that the final space bound of the heavy hitter algorithm can be controlled (e.g., we cannot afford to store $\Omega(1/\epsilon^2)$ many coordinate IDs, which would cost $\Omega((\log n)/\epsilon^2)$ bits of space. We have to store only $O(1/\epsilon^2)$ many approximate values). The major contribution of our algorithm is the careful reduction from F_p estimation to ℓ_p heavy hitter-finding. Many reductions are known in the literature but are suffering from a $\text{poly}(\log n)$ space blow up. Our careful reduction allows us to pay only a $\text{poly}(\log\log n)$ space blow up.

In what follows, we illustrate the high level ideas of the heavy hitter reduction. Let $v = (f_1, f_2, \ldots, f_n)$ be the frequency vector of the items. We first apply a random scaling X_i to each f_i, where each X_i is pairwise independently drawn from some distribution. We argue that finding the leading ℓ_p heavy hitter of the scaled frequency vector (denoted as $X_{i^*} f_{i^*}$) gives a good estimation of the $F_p^{1/p}$ of the original stream. The distribution of X_i is a so-called p-inverse distribution (see Definition 2 for details). This distribution has a similar tail as that of the max-stable distributions used in [3] or the precision sampling distribution in [4]. However, it has different properties better suited to our setting, e.g., we can use pairwise independent variables to do the scaling. In contrast, for max-stable distributions, we have to use fully random hash functions and apply pseudo-random generators, which is problematic in our very low space regime (a related technique called tabulation-based hashing has similar problems). Note that [4] does not require pseudo-random generators, but their precision sampling technique aims to solve a more general family of problems and their

[3] We use $\tilde{O}$ to hide $\text{poly}(\log\log n + \log\epsilon^{-1})$ factors.

distribution has a slightly different form, e.g., the random variable is drawn from a continuous distribution. The p-inverse distribution is particularly suited for p-norm estimation, and allows for a concise algorithm and improved space complexity in our setting.

After choosing the random scalings, we group the coordinates of v into $\Theta(\log n)$ levels by their scalings, i.e., if $2^{-w}F_p < X_i^p \leq 2^{-w+1}F_p$, then i is in level w for some integer w. Since we do not know F_p before the stream arrives, the exact level ID is not known to the algorithm. But since each X_i is obtained pseudo-randomly, the algorithm is able to know whether two coordinates are in a same level or not (i.e., the relative level ID) before looking at the stream. Let $Z_w \subset [n]$ be the set of universe items in level w. We observe that if for some coordinate $i^* \in Z_w$, it satisfies $X_{i*}^p f_{i^*}^p \approx F_p$, then $f_{i^*}^p = \Omega(2^w)$. Fortunately, we can also show that in expectation, i^* is an ℓ_p-heavy hitter of the substream induced by Z_w (i.e., in expectation the F_p of the stream restricted to Z_w is approximately 2^w). Our algorithm simply *looks for* i^* *from* Z_w *for every* $w \in [\log n]$. One may notice that if we run the search instance in parallel, then there will be a $(\log n)$-factor blowup in the space of a heavy hitter algorithm. However, we can show that for random order streams, one can choose a $w_0 = \Theta(\log \log n)$ such that

1. for all $w > w_0$: the search for the heavy hitter i^* can be done in one pass and in sequence for each w. This is because f_{i^*} is large (i.e., $\Omega(2^w)$) and a small portion of the random order stream contains sufficient information about i^*.
2. 2. for all $w \leq w_0$: with high probability, $|Z_w| = \text{poly}(\log n)$. We thus do a brute force search for each level w below w_0 in parallel. Each search instance uses a small amount of memory because of the small universe size.

The final space overhead becomes a $\text{poly}(w_0)$ factor rather than a $\Theta(\log n)$ factor. This observation is critical to reduce space usage for approximating frequency moments in random order streams and is, to the best of our knowledge, new. For $p \geq 2$, the above claim is no longer true. We leave the exact space complexity for $p \geq 2$ as an open problem.

1.3 Roadmap

In Section 2, we introduce some definitions and the p-inverse distribution. In Section 3, we present our algorithm for F_2. In Section 4, we present our generic framework for approximating F_p as well as our main result. In Section A of the full version of this paper, we present the detailed construction of each subroutine used in Section 4, and the details of the main algorithm. In Section 6, we introduce our deterministic algorithm, which uses our randomized algorithm as a subroutine. We also show its optimality in the same section.

2 Preliminaries

▶ **Definition 1** (Aggregate Streaming). An *insertion-only stream* (or simply *stream*) $\mathcal{S} = \langle a_1, a_2, \ldots, a_m \rangle$ is a sequence of integers, where each $a_i \in [n]$. A *weighted stream* $\mathcal{S}' = \langle (a_1, w_1), (a_2, w_2), \ldots, (a_m, w_m) \rangle$ is a sequence of pairs, where each $a_i \in [n]$ and each $w_m \in \mathbb{R}$. The insertion-only stream $\mathcal{S}$ is a special case of a weighted stream with all the weights being 1. The frequency $f_i(\mathcal{S}')$ of a weighted stream is defined as the sum of all weights of the item i. Formally,

$$f_i(\mathcal{S}') := \sum_{j=1}^{m} \mathbb{I}(a_j = i) w_j.$$

The frequency vector $V(\mathcal{S}') \in \mathbb{R}^n$ is the vector with i-th coordinate equal to $f_i(\mathcal{S}')$, for each $i \in [n]$. The p-th frequency moment, for $p \geq 0$, is defined as

$$F_p(\mathcal{S}') := \|V(\mathcal{S}')\|_p^p := \sum_{i=1}^{n} f_i^p(\mathcal{S}').$$

For time $0 < t_1 \leq t_2 \leq m$, we let

$$\mathcal{S}'^{:t_1} := \langle (a_1, w_1), (a_2, w_2), \ldots, (a_{t_1}, w_{t_1}) \rangle \quad \text{and}$$
$$\mathcal{S}'^{t_1:t_2} := \langle (a_{t_1}, w_{t_1}), (a_{t_1+1}, w_{t_1+1}), \ldots, (a_{t_2}, w_{t_2}) \rangle$$

be sub-streams of $\mathcal{S}'$ from 1 to t_1 or from t_1 to t_2.

We introduce a discretized version of the α-Fréchet distribution: the α-Inverse distribution.

▶ **Definition 2.** Fixing an $\alpha > 0$, we say a random variable X on $\mathbb{N}$ is drawn from an α-*Inverse* distribution if

$$\mathbb{P}[X < x] = 1 - \tfrac{1}{x^\alpha} \quad \text{for} \quad x \in \mathbb{N}_+.$$

▶ **Definition 3** (α-Scaling Transformation)**.** Given an $\alpha > 0$, an *α-scaling transformation (ST)* $\mathcal{T}_{k,\alpha} : [n]^m \to ([k] \times [n] \times \mathbb{R})^m$ is a function acting on a stream of length m on the universe $[n]$. On input stream $\mathcal{S}$, it outputs a weighted stream $\mathcal{S}'$ of length km on universe $[k] \times [n]$ via the following operation: let $X_{i,j}$ be identically distributed and independent (or, actually, limited independence suffices) α-Inverse random variables, where $i = 1, 2, \ldots, k$ and $j = 1, 2, \ldots, n$. For each $a \in \mathcal{S}$, the transformation outputs

$$\mathcal{T}_{k,\alpha}(a) \to ((a, 1), X_{1,a}), ((a, 2), X_{2,a}), \ldots, ((a, k), X_{k,a}).$$

The next lemma shows that the $\frac{k}{2}$-th largest element of the transformed frequency vector gives a good approximation to the α-norm of the vector.

▶ **Lemma 4.** *Let $\mathcal{S}$ be a stream of items from the universe $[n]$. Let $\mathcal{T}_{k,\alpha}$ be a pairwise independent α-ST with $k \geq \frac{160}{\alpha^2\epsilon^2}$ being an even integer and $\alpha \geq 0$, where $\epsilon \in (0, \frac{1}{2\alpha})$. Let $\mathcal{S}' = T_{k,\alpha}(\mathcal{S})$. Define the two sets,*

$$U_+ := \{(a, r) \in [n] \times [k] : f_{(a,r)}(\mathcal{S}') \geq 2^{1/\alpha}(1 - \epsilon)\|V(\mathcal{S})\|_\alpha\}$$

and

$$U_- := \{(a, r) \in [n] \times [k] : f_{(a,r)}(\mathcal{S}') < 2^{1/\alpha}(1 + \epsilon)\|V(\mathcal{S})\|_\alpha\}.$$

Then with probability at least 0.9*,*

$$|U_+| \geq \tfrac{k}{2}, \quad |\overline{U_-}| < \tfrac{k}{2} \quad \textit{and} \quad |U_+ \cap U_-| > 0,$$

where $\overline{U}$ is the complement of the set U.

The proof of this lemma is provided in Section B of the full version.

3 A Simple F_2 Algorithm For Long Streams

We start with a very simple algorithm for approximating F_2 in a random order stream. We denote the algorithm by `RANDF2`. The algorithm has a parameter $b > 0$. It treats the stream updates as a series of length b blocks, i.e.,

$$\mathcal{S} = (B_1, B_2, \ldots, B_{m/b}).$$

Initialize a register $K = 0$. At any time, suppose the current block is B_i. The algorithm simply stores everything it sees until the end of B_i. After storing B_i entirely, the algorithm computes

$$K_i = \sum_{j=1} \binom{f_j(B_i)}{2}.$$

This can be done using $b \log n$ bits of space. Then, the counter K is updated as

$$K \leftarrow K + K_i.$$

At the end of the stream, the algorithm computes

$$Y = \frac{2K(m^2 - m)}{(b^2 - b)T} + m,$$

where T is the ID of the last complete block, and m is the length of the stream. The algorithm uses $O(b \log n)$ bits. By setting

$$b = \Theta[\max((\epsilon^2 \log n)^{-1}, 2) \cdot \log \tfrac{1}{\delta}],$$

we obtain the following theorem.

▶ **Theorem 5.** *Let $\mathcal{S}$ be a random order stream satisfying $F_2(\mathcal{S}) \geq m \cdot \log n$. After one pass over the stream $\mathcal{S}$, Y is a $(1 \pm \epsilon)$ approximation to $F_2(\mathcal{S})$ with probability at least $1 - \delta$. Moreover, to compute Y, `RANDF2` uses $\mathcal{O}(\epsilon^{-2} \log \delta^{-1} + \log n)$ bits of memory.*

Proof. For each $j \in [n]$, we let its stream updates be $u_1^{(j)}, u_2^{(j)}, \ldots, u_{f_j(\mathcal{S})}^{(j)}$. Then K_i is the number of pairs of the form $(u_{\ell_1}^{(j)}, u_{\ell_2}^{(j)})$ appearing in B_i. We denote $F_2 = F_2(\mathcal{S})$ and $F_1 = F_1(\mathcal{S})$ for simplicity. Thus,

$$\forall i \in [T] : \mathbb{E}(K_i) = \frac{\sum_{j=1}^{n} \binom{f_j(\mathcal{S})}{2}}{\binom{F_1}{2}} \binom{b}{2} = \frac{F_2 - m}{m^2 - m} \frac{(b^2 - b)}{2}.$$

Thus, by linearity of expectation,

$$\mathbb{E}[Y] = F_2.$$

We now prove that Y is concentrated around its mean. Notice that the algorithm can be viewed as sampling a number of "pairs". A pair is formed by two updates to the same universe element. There are $q = (F_2 - F_1)/2$ many pairs. Let $P = [q]$ denote the set of pairs. For each pair $z \in P$, we let X_z denote the indicator that X_z is sampled by some bucket. Let $K = \sum_{z \in P} X_z$. Note that this K is the same as the one denoted in the algorithm. Let $Q_z = \mathbb{E}[K | X_1, X_2, \ldots, X_z]$. The Q_z for $z = 1, 2, \ldots$ form a Doob martingale. Also notice that $|Q_z - Q_{z-1}| \leq 1$. Next, we proceed to bound the variance of

$$Q_z - Q_{z-1} = X_z | X_1, X_2, \ldots X_{z-1}.$$

For a pair $z \in P$, let a, b be the two nodes. Consider a fixed assignment of $X_1, X_2, \ldots X_{z-1}$. Also note that, knowing X_i, the two nodes of the i-th pair are assigned to some block.

Now, if from the $X_1, X_2, \ldots, X_{z-1}$, a, b are both assigned and to the same block, then $X_z = 1$ and otherwise $X_z = 0$. For both cases $\mathrm{Var}(Q_z - Q_{z-1}) = 0$. If a, b are assigned, but it cannot be determined if they are in the same block, then $\mathbb{P}[X_z = 1] \leq b/m$ and thus $\mathrm{Var}(X_z) \leq b/m$. If only one of a, b is assigned, then $\mathbb{P}[X_z = 1] \leq b/m$, and thus

$\mathrm{Var}(X_z) \le b/m$. Lastly, if both a, b are not assigned, then $\mathbb{P}[X_z = 1] \le b/m$. Thus $\mathrm{Var}(X_z) \le b/m$. Overall, we have that $v_z^2 := \mathrm{Var}(Q_z - Q_{z-1}) \le b/m$ for all possible $X_1, X_2, \ldots, X_{z-1}$. Let

$$V = \sum_z v_z^2 \le \tfrac{b(F_2-m)}{2m}.$$

Next, by Bernstein's inequality [14], we have that,

$$\mathbb{P}[|K - \mathbb{E}(K)| \ge t] \le 2\exp\left(-\tfrac{t^2}{2V(1+t/3V)}\right).$$

Since we need to have a $(1 \pm \epsilon)$ approximation to F_2, we can set

$$\frac{2t(m^2-m)b}{(b^2-b)m} \le \epsilon F_2 \quad \text{and} \quad t = \epsilon\frac{F_2 b}{2m}.$$

Since $F_2 \ge m \cdot \log n \cdot \log \frac{1}{\delta}$, and ϵ is sufficiently small, we can bound the error probability by:

$$\begin{aligned}\mathbb{P}[|K - \mathbb{E}(K)| \ge t] &\le 2\exp\left(-\frac{t^2}{2V(1+t/3V)}\right)\\ &\le 2\exp\left(-\frac{\epsilon^2 F_2^2 b}{8m(F_2-m)}\right) \le \delta, \end{aligned} \tag{1}$$

for $b = \Omega(\frac{\log \frac{1}{\delta}}{\epsilon^2 \cdot \log n})$. Finally, since $K \in \mathbb{E}(K) \pm \epsilon F_2 b/(2m)$, we have

$$Y \in F_2 \pm \epsilon\tfrac{F_2 b}{2m} \cdot \tfrac{2(m^2-m)b}{(b^2-b)m} \subset (1 \pm \epsilon)F_2$$

as desired. ◀

4 A Generic Framework for F_p Estimation

In this section, we first construct a generic framework for F_p estimation, and then we construct all the components in subsequent sections. For a random order stream $\mathcal{S}$, we will need the following three components to construct an F_p estimation algorithm.

- A counter that stores the time for the current update;
- An algorithm that gives a constant approximation to the current $F_p(\mathcal{S}^{:t})$;
- An algorithm that computes an accurate $(1 \pm \epsilon)$ approximation to $F_p(\mathcal{S})$ given $\mathrm{poly}(\log n, \epsilon^{-1})$ approximations to m and $F_p(\mathcal{S})$.

To begin, we denote by `C2Fp` a data structure that, once initialized with a $\mathrm{poly}(\log n, \epsilon^{-1})$-approximation of both the length and p-th frequency moments, $F_p(\mathcal{S})$, supports two operations: update and query. At the end of the stream, `C2Fp.query()` returns either `Fail` or a $(1 \pm \epsilon)$-approximation to F_p of the input stream $\mathcal{S}$. Component (1) will be used to guess the length of the stream, and component (2) will be used to guess an approximation to $F_p(\mathcal{S})$. We denote component (2) by `ConstFp`, which is a data structure that supports both update and query operations. `ConstFp.query()` returns a 2-approximation to $F_p(\mathcal{S}^{:t})$ at some fixed t. The full framework is described in Algorithm 1.

Our full algorithm is denoted by `RndF`$_p$, which uses `C2Fp` as a subroutine. From a high level, the algorithm constantly guesses the length of the stream. If at some point in time the algorithm finds that the current guess of the length is at least a factor $C = \mathrm{poly}(\epsilon^{-1}, \log n)$ smaller than the true value of the stream, then the algorithm initializes a new instance of `C2Fp` to estimate the F_p value of the stream. At the end, it is guaranteed that a stored

Algorithm 1: Full algorithm for F_p in random order: $\texttt{RndF}_p$.

Data:
$\mathcal{S} = \langle a_1, a_2, \ldots, a_m \rangle$ is a random order stream of length m from a universe $[n]$ (known in advance);
$p \in (0, 2]$ is a real number, which is a constant;

Initialize (p, n, ϵ, δ)**:**
  $m_0 \leftarrow 1$, $m_1 \leftarrow 1$ $G_0 \leftarrow 1$. Here m_0 is the approximate length, G_0 is a guess of F_p, ϵ is the target precision, and δ is the failure probability;
  $A_1 \leftarrow$new $\texttt{C2Fp}(p, \epsilon/3, n, m_0, G_0, \delta/3)$, $A_2 \leftarrow$new $\texttt{C2Fp}(p, \epsilon/3, n, m_0, G_0, \delta/3)$;
  $A_3 \leftarrow$new $\texttt{ConstFp}(p, n, \delta/3)$;
  $C \leftarrow \text{poly}(\frac{1}{\epsilon}, \log \frac{n}{\delta})$;

Update a**:**
  A_1.update(a), A_2.update(a), A_3.update(a);
  $m_1 \leftarrow m_1 + 1$;
  if $m_1 \geq C m_0$ **then**
    $A_1 \leftarrow A_2$;
    $G_0 \leftarrow A_3$.query();
    $m_0 \leftarrow m_1$;
    $A_2 \leftarrow$new $\texttt{C2Fp}(p, \epsilon/3, n, m_0, G_0, \delta/3)$;

Query():
  return A_1.query();

instance of C2Fp uses at least a $(1 - \text{poly}(\epsilon, 1/\log n))$ portion of the stream to approximate the frequency moments. It can be verified that an accurate estimation of F_p of this portion of the stream will serve as a good estimator for the overall stream. Therefore, if C2Fp is able to output the correct answer with high probability, then the algorithm RndF is guaranteed to be correct with high probability.

▶ **Theorem 6** (Main Theorem). *For fixed $p \in [0, 2)$, $\epsilon \in (0, 1)$,$\delta \in (\frac{1}{\text{poly}(n)}, \frac{1}{2})$, and $n \in \mathbb{N}_+$, algorithm $\texttt{RndF}_p$, makes a single pass over a random order stream $\mathcal{S}$ on universe $[n]$, and outputs a number $\hat{F}$ such that, with probability at least $1 - \delta$,*

$$(1 - \epsilon)F_p(\mathcal{S}) \leq \hat{F} \leq (1 + \epsilon)F_p(\mathcal{S}),$$

where the probability is over both the randomness of the algorithm and the randomness of the data stream. The algorithm uses

$$\mathcal{O}[(\frac{1}{\epsilon^2}(\log\log n + \log\frac{1}{\epsilon})^4 + \log n)\log\frac{1}{\delta}]$$

bits of memory in the worst case.

Proof. Without loss of generality, we assume $F_p(\mathcal{S}) = \Omega(\text{poly}\frac{1}{\epsilon}\text{poly}\log\frac{n}{\delta})$, since otherwise we can use a turnstile F_p algorithm with memory $\mathcal{O}\big(\frac{1}{\epsilon^2}\log\log\frac{n}{\delta} + \frac{1}{\epsilon^2}\log\frac{1}{\epsilon}\big)$ bits to solve the F_p estimation problem. Initialize $m_0 = 1$. Let $\mathcal{S}' = \mathcal{S}^{m_0:m}$. By definition of the algorithm, A_1 is an instance of C2Fp that runs on $\mathcal{S}'$. Let $\mathcal{S}'' = \mathcal{S}^{0:m_0}$ and $C = \text{poly}\epsilon^{-1}\text{polylog}\frac{n}{\delta}$. By definition of the algorithm, we always update m_0 so that

$$\frac{m}{C^2} \leq m_0 \leq \frac{m}{C},$$

at the end of the stream. By Lemma 24 and Lemma 25 of the full version of this paper, we have, with probability at least $1 - \delta/3$,

$$\frac{F_p(\mathcal{S})}{5^p C^{2p}} \leq F_p(\mathcal{S}'') \leq (17\log\frac{20n}{\delta})^p(C^{-1}F_p(\mathcal{S}) + 4\log\frac{20n}{\delta}) \leq \frac{(18\log\frac{20n}{\delta})^p}{C}F_p(\mathcal{S}),$$

where the last inequality holds for sufficiently large $F_p(\mathcal{S})$. Conditioned on this event, we obtain that

$$\|V(\mathcal{S}'')\|_p \leq \frac{(18 \log \frac{20n}{\delta})}{C^{1/p}} \|V(\mathcal{S})\|_p \leq \frac{\epsilon}{3p} \|V(\mathcal{S})\|_p,$$

for sufficiently large C. By the triangle inequality, we obtain

$$(1 - \epsilon/(3p))\|V(\mathcal{S})\|_p \leq \|V(\mathcal{S})\|_p - \|V(\mathcal{S}'')\|_p \leq \|V(\mathcal{S}')\|_p \leq \|V(\mathcal{S})\|_p.$$

Thus $F_p(\mathcal{S}')$ is a $(1 \pm \epsilon/3)$ approximation to $F_p(\mathcal{S})$.

In the algorithm, A_3 is an instance of `ConstFp`, i.e., by Theorem 7. Let G_0 be the output of A_3.query(). Since with probability at least $1 - \delta/3$, A_3 outputs a c_0 approximation to $F_p(\mathcal{S}'')$ for some constant c_0, we obtain that G_0 is a $5^p c_0 C^{2p} (\log \frac{20n}{\delta})^{p-1}$ approximation to $F_p(\mathcal{S}')$.

In the algorithm, A_1 is an instance of `C2Fp`, which runs on the stream $\mathcal{S}'$ at any time t with the required parameters, i.e., G_0. By Theorem 8, with probability at least $1 - \delta/3$, A_1.query() outputs a $(1 \pm \epsilon/3)$ approximation to $F_p(\mathcal{S}')$, and thus a $(1 \pm \epsilon)$ approximation to $F_p(\mathcal{S})$. By a union bound, the overall algorithm is correct with probability at least $1 - \delta$.

By Theorem 16 and Theorem 17, of the full version, the space needed for A_1 and A_2 is

$$\mathcal{O}[(\frac{1}{\epsilon^2}(\log \log n + \log \frac{1}{\epsilon})^4 + \log n) \log \frac{1}{\delta}].$$

The space needed for A_3 is $\mathcal{O}(\log n \log \frac{1}{\delta})$ (Theorem 7). Thus the total space is dominated by the total space used by A_1 and A_2 as desired. ◀

The following is a theorem required in the above proof.

▶ **Theorem 7** (Const. F_p approx., [21]).
For a fixed n, there exists a turnstile streaming algorithm, which on input a stream $\mathcal{S}$ of length m, outputs a number $F \in (1 \pm \epsilon)F_p(\mathcal{S})$ with probability at least $1 - \delta$. The algorithm uses $\mathcal{O}(\epsilon^{-2} \log m + \log \log(n)) \log \delta^{-1})$ bits of space in the worst case.

In subsequent sections, we will construct the `C2Fp` Algorithm.

5 A $(1 \pm \epsilon)$ Approximation to F_p With a Prior

In this section, we construct the algorithm `C2Fp`. We assume that the input is a random order stream and that the algorithm is given two parameters, $\hat{m}$ and G, which are $\text{poly}(\epsilon^{-1}, \log n)$ approximations to the length and the p-th frequency moments of the stream $\mathcal{S}$, respectively.

5.1 High Level Idea

Although the high level idea is introduced in the introduction, we repeat it here with more details for better understanding of the algorithm. To illustrate the intuition of the algorithm, we first consider a constant factor approximation algorithm. Estimating the frequency moments can be reduced to finding the *heavy hitters* of a scaled vector, as shown in Lemma 4. Suppose the frequency vector in a stream is $v = (f_1(\mathcal{S}), f_2(\mathcal{S}), \ldots, f_n(\mathcal{S}))$, and the scaling applied to it is $X = (X_1, X_2, \ldots, X_n)$, where the X_i are pairwise independent *p-Inverse* (see Definition 2) random variables. Let i^* be the maximum of the scaled vector. By Lemma 4, we expect $X_{i^*}^p f_{i^*}^p(\mathcal{S}) \approx F_p$. We group the coordinates of v into $\Theta(\log n)$ levels by their scalings, i.e., if $2^{-w} F_p < X_i^p \leq 2^{-w+1} F_p$, then i is in level w. Let $Z_w \subset [n]$ be the universe items in

level w. We observe that if $i^* \in Z_w$, then $f_{i^*}^p(\mathcal{S}) = \Omega(2^w)$. Luckily, we can also show that, in expectation, i^* is an F_p-heavy hitter of the substream induced by Z_w. Our algorithm is simply *looking for i^* from Z_w for every $w \in [\log n]$*. One may notice that if we run the search instances in parallel, then there will be a $\log n$ factor blowup in the space. However, we can show that in a random order stream, one can choose a $w_0 = \Theta(\log \log n)$ such that

1. for all $w > w_0$: the search for i^* can be done in one pass and in series for each w.
2. for all $w \le w_0$: with high probability, $|Z_w| = \operatorname{poly} \log n$. We thus do a brute force search for each level w below w_0 in parallel.

The final space overhead is a $\operatorname{poly}(w_0)$ factor rather than $\Theta(\log n)$.

To reduce the error from constant to $(1 \pm \epsilon)$, we repeat the above process $\Theta\left(\frac{1}{\epsilon^2}\right)$ times conceptually. Namely, we apply a p-ST $\mathcal{T}_{k,p}$ transformation to the stream, where $k = \Theta\left(\frac{1}{\epsilon^2}\right)$. For $r = 1, 2, \ldots, [k]$, $i = 1, 2, \ldots, n$, we denote the scaling p-Inverse random variable by $X_i^{(r)}$. We wish to find the heavy hitter for each r using the same procedure described above. By Lemma 4, the $k/2$-th largest of all the outputs serves as a good approximation to $F_p(\mathcal{S})$.

5.2 The Algorithm

The algorithm needs three components, `SmallApprox`, `SmallCont` and `LargeCont`. All these algorithms support "update" and "query" operations. `SmallApprox` returns fail if $F_p(\mathcal{S})$ is much larger than $\operatorname{poly}(\epsilon^{-1}, \log n)$, otherwise returns an approximation to $F_p(\mathcal{S})$. `SmallApprox` is a turnstile streaming F_p algorithm [21] but with restricted memory. Once the memory exceeds the memory quota, the algorithm simply returns `Fail`. `SmallCont` estimates the contribution from the small-valued frequencies and `LargeCont` estimates the contribution from the large-valued frequencies. The correctness of these algorithms is presented in Theorem 16 and 17 of the full version of this paper. The full algorithm is presented in Algorithm 2. The following theorem guarantees its correctness.

▶ **Theorem 8.** *Fix $p \in [0, 2]$, $\epsilon \in (0, 1/2)$ and $\delta = \Omega(1/\operatorname{poly}(n))$. Let $\mathcal{S}$ be a random order stream on universe $[n]$ and with length m. Given that $C_0^{-1} F_p(\mathcal{S}) \le G_0 \le F_p(\mathcal{S})$ and $C_0^{-1} m \le m_0 \le m$ for some $C_0 = \operatorname{poly}(\epsilon^{-1}, \log n)$, there exists an algorithm $\mathcal{A}$, which makes a single pass over $\mathcal{S}$ and outputs a number F such that $F \in (1 \pm \epsilon) F_p(\mathcal{S})$ with probability at least $1 - \delta$, where the probability is over both the randomness of the algorithm and of the order of the stream. The algorithm uses*

$$\mathcal{O}[(\tfrac{1}{\epsilon^2}(\log \log n + \log \tfrac{1}{\epsilon})^4 + \log n) \log \tfrac{1}{\delta}]$$

bits of memory in the worst case.

We postpone the full proof and detailed algorithmic constructions to the appendix.

6 Deterministic Algorithm for F_p Approximation

In this section we introduce our deterministic algorithm for F_p approximation, which follows from our randomized algorithm with an initial space-efficient randomness extraction procedure applied to a prefix of the stream.

6.1 Upper Bound

▶ **Theorem 9.** *Fix $p \ge 0, \epsilon \in (0, 1)$. There exists a deterministic algorithm that makes a single pass over a random order stream $\mathcal{S}$ on the universe $[n]$, and outputs a number*

Algorithm 2: F_p-Algorithm with Approximation: `C2Fp`$(p, \epsilon, n, m_0, G_0)$.

Data:
$p \in [0, 2]$, a real number;
$L \in [\frac{F_p(\mathcal{S})}{C_0}, F_p(\mathcal{S})]$ for some $C_0 = \Theta[\text{poly}(\epsilon^{-1}, \log n)]$;
$m_0 \in [\frac{m}{C_0}, m]$;
$\mathcal{S} = \langle a_1, a_2, \ldots, a_m \rangle$ is random order stream of length m;
Result: $F \in (1 \pm \epsilon)F_p(\mathcal{S})$;

Initialize(p, n, ϵ, δ)**:**
  $k \leftarrow \Theta\big(\frac{1}{\epsilon^2}\big)$;
  $X_i^{(r)} \sim p$-Inverse distribution for $i \in [n]$ and $r \in [k]$, pairwise independent;
  $w_0 \leftarrow d_0\big(\log\log n + \log\frac{1}{\epsilon}\big)$ for some large constant d_0;
  Let $K \in \mathbb{R}^{n \times k}$ have entries $K_{i,r} = X_i^{(r)} v_i$ (only for notational purposes);
  $B_1 \leftarrow$ new `SmallApprox`(p, n, ϵ);
  $B_2 \leftarrow$ new `SmallCont`$(p, n, k, \epsilon, w_0, L, \{X_i^r\})$;
  $B_3 \leftarrow$ new `LargeCont`$(p, n, k, \epsilon, w_0, L, \{X_i^r\})$;
Update(a)**:**
  B_1.update(a); B_2.update(a); B_3.update(a);
Query:
  if B_1.*query*$() \neq$`Fail` **then**
    return B_1.query()
  else if B_2.*query*$() =$`Fail` *or* B_3.*query*$() =$`Fail` **then**
    return `Fail`;
  else
    $R \leftarrow$ the $(k/2)$-th largest element of B_2.query() $\circ$ B_3.query();
    return $(R)^p/2$

$F \in (1 \pm \epsilon)F_p(\mathcal{S})$ with probability at least $1 - \delta$, where the randomness is over the order of the stream updates. The algorithm uses

$$\mathcal{O}\left[\frac{1}{\epsilon^2}\left(\log\log n + \log\frac{1}{\epsilon}\right)^4 \log\frac{1}{\delta} + \log n \cdot \left(\log\log n + \log\frac{1}{\delta}\right) \cdot \frac{1}{\delta} + \log n \log\frac{1}{\epsilon}\right]$$

bits of memory, provided $\delta \geq 1/\text{poly}(n)$.

Proof. W.l.o.g., we assume $\epsilon \geq 1/\sqrt{n}$, since otherwise we can simply store an approximate counter for each item in the stream. It is sufficient to show that we are able to derandomize the randomized algorithm using the random updates from the stream using a near-logarithmic number of bits of space. First we pick $s = \mathcal{O}(\log\log n + \log\delta^{-1})$ and store all the universe items, their frequencies and their first arrival times until we obtain s distinct items in the stream. Let z_1 denote when the this happens. We assume the stream is long enough that this step can be done, since otherwise we obtain an exact estimate of F_p. We show in Section 6.1.1 how to obtain a nearly uniformly random seed of s bits.

We thus obtain a nearly uniform sample from all the prime numbers with $\mathcal{O}(\log\log n + \log\frac{1}{\delta})$ bits (note that there are $\text{poly}(\log n/\delta)$ many such prime numbers). Let this sampled prime number be q. For the next $\mathcal{O}(\log n/\delta)$ distinct universe items, denoted by R, we argue that with probability at least $1 - \delta$, all of them are distinct modulo q. Indeed, consider any $r_1, r_2 \in R$ with $r_1 \neq r_2$. Then $r_1 - r_2$ can have at most $\log n$ prime factors ([17], p.355). For

all $\binom{|R|}{2}$ pairs, their differences can have at most $\mathcal{O}(\log^3 n/\delta^3)$ distinct prime factors in total. Thus with probability at least $1-\delta$, q does not divide any of the differences. Therefore the set R is mapped to distinct numbers modulo q. The value q can be stored using at $\mathcal{O}(\log\log n + \log\frac{1}{\delta})$ bits.

Next we approximately store the frequencies of each item in R using the random order stream. To do this, we first fix the following numbers $g_1 = 1, g_2 = 2, g_3 = 4, \ldots, g_i = 2^{i-1}$. For each $r \in R$, we store the largest number i such that $f_r(\mathcal{S}^{z_1:g_i}) = \text{poly}(\log n, \epsilon^{-1})$ and $f_r(\mathcal{S}^{z_1:g_i})$ as well. Therefore, such an operation only costs $\mathcal{O}(\frac{\log n}{\delta}(\log\log n + \log\frac{1}{\delta} + \log\epsilon^{-1}))$ bits. By Lemma 22 of the full version, the frequency of each item is preserved up to a $(1\pm\epsilon)$ factor with probability at least $1 - 1/\text{poly}(n)$. Note that if the stream ends before we observe all of R, we obtain a good approximation to $F_p(\mathcal{S})$ immediately. We also store the first occurrence in the stream of each item in R. We also store the parity of the first appearance of each item.

Repeating the extraction argument in Section 6.1.1 for the set R, we can now extract $\mathcal{O}(\log n)$ bits that is $(1\pm\delta)$ close to uniform. Given these bits, it now suffices to run our earlier randomized algorithm on the remaining part of the stream.

There is one last problem remaining, however. Namely, it may be the case that the stream used for extracting random bits contributes too much to $F_p(\mathcal{S})$, causing the estimation of the randomized algorithm to have too much error (since the prefix and the suffix of the stream share the same items, we need to take the p-th power of the sum of their frequencies). This problem can be solved as follows – we can continue the frequency estimation in parallel with the randomized algorithm until the F_p value becomes at least a $1/\epsilon$ factor larger than the time when we initialized our randomized algorithm. Therefore, if the stream ends before this happens, then we use the frequency estimates for calculating $F_p(\mathcal{S})$ from our deterministic algorithm. Otherwise we use the value of the randomized algorithm (which is seeded with the seed found by our deterministic algorithm). In either case, the overall error is at most ϵF_p. ◀

6.1.1 Derandomization

Let $s > 0$ be a parameter. Suppose we store $t = \mathcal{O}(s/\delta)$ distinct universe items with their approximate frequencies as well as the IDs and the parities of their first appearances in the stream. Denote the set of these items by H and the overall length of the stream as m'. First, we sort the items by their approximate counts and take the smallest $\delta t/100$ items as set L. We additionally sort the items in L by their IDs, and obtain a bit string b of length $|L|$, where each bit b_i is the parity of the first appearance of the i-th item in L. Since L contains the smallest $\delta t/100$ items of H, we have for each $w \in L$, $f_w(\mathcal{S}^{0:m'}) \le m'/t/(1-\delta/100)$. Thus for each bit b_i,

$$\mathbb{P}[b_i = 0], \mathbb{P}[b_i = 1] \in \frac{1}{2} \pm \frac{2}{m'}.$$

and

$$\forall x \in \{0,1\}^{|L|} : \mathbb{P}[b = x] \in (\frac{1}{2} \pm \frac{2}{m'})^{|L|} \subset \frac{1}{2^{|L|}}(1 \pm \frac{5|L|}{m'}) \subset \frac{1}{2^{|L|}}(1 \pm \frac{\delta}{20}).$$

As such, we obtain a bit stream of length $\Omega(s)$, that is close to uniform bits up to a $(1\pm\delta)$ factor.

References

1. Noga Alon, Phillip B Gibbons, Yossi Matias, and Mario Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 10–20. ACM, 1999.
2. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29. ACM, 1996.
3. Alexandr Andoni. High frequency moment via max stability. *Unpublished manuscript*, 2012.
4. Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms via precision sampling. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 363–372. IEEE, 2011.
5. Alexandr Andoni, Andrew McGregor, Krzysztof Onak, and Rina Panigrahy. Better bounds for frequency moments in random-order streams. *arXiv preprint arXiv:0808.2222*, 2008.
6. Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, and David P Woodruff. Beating countsketch for heavy hitters in insertion streams. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 740–753. ACM, 2016.
7. Vladimir Braverman, Jonathan Katzman, Charles Seidell, and Gregory Vorsanger. An optimal algorithm for large frequency moments using o (nˆ(1-2/k)) bits. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 28. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.
8. Vladimir Braverman and Rafail Ostrovsky. Recursive sketching for frequency moments. *arXiv preprint arXiv:1011.2571*, 2010.
9. Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 641–650. ACM, 2008.
10. Amit Chakrabarti, TS Jayram, and Mihai Patrascu. Tight lower bounds for selection in randomly ordered streams. In *SODA*, pages 720–729, 2008.
11. Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *Computational Complexity, 2003. Proceedings. 18th IEEE Annual Conference on*, pages 107–117. IEEE, 2003.
12. Michael Crouch, Andrew McGregor, Gregory Valiant, and David P Woodruff. Stochastic streams: Sample complexity vs. space complexity. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 57. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
13. Erik Demaine, Alejandro López-Ortiz, and J Munro. Frequency estimation of internet packet streams with limited space. *Algorithms—ESA 2002*, pages 11–20, 2002.
14. Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.
15. Sudipto Guha and Zhiyi Huang. Revisiting the direct sum theorem and space lower bounds in random order streams. *Automata, Languages and Programming*, pages 513–524, 2009.
16. Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM Journal on Computing*, 38(5):2044–2059, 2009.
17. Godfrey Harold Hardy and Edward Maitland Wright. *An introduction to the theory of numbers*. Oxford University Press, 1979.
18. Piotr Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 189–197. IEEE, 2000.
19. Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 202–208. ACM, 2005.

20 Daniel M Kane, Jelani Nelson, Ely Porat, and David P Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 745–754. ACM, 2011.

21 Daniel M Kane, Jelani Nelson, and David P Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1161–1178. SIAM, 2010.

22 J Ian Munro and Mike S Paterson. Selection and sorting with limited storage. *Theoretical computer science*, 12(3):315–323, 1980.

23 David Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 167–175. Society for Industrial and Applied Mathematics, 2004.

Proportional Approval Voting, Harmonic k-median, and Negative Association

Jarosław Byrka
University of Wrocław, Wrocław, Poland
jby@cs.uni.wroc.pl

Piotr Skowron
University of Warsaw, Warsaw, Poland
p.skowron@mimuw.edu.pl

Krzysztof Sornat
University of Wrocław, Wrocław, Poland
krzysztof.sornat@cs.uni.wroc.pl

Abstract

We study a generic framework that provides a unified view on two important classes of problems: (i) extensions of the k-median problem where clients are interested in having multiple facilities in their vicinity (e.g., due to the fact that, with some small probability, the closest facility might be malfunctioning and so might not be available for using), and (ii) finding winners according to some appealing multiwinner election rules, i.e., election system aimed for choosing representatives bodies, such as parliaments, based on preferences of a population of voters over individual candidates. Each problem in our framework is associated with a vector of weights: we show that the approximability of the problem depends on structural properties of these vectors. We specifically focus on the harmonic sequence of weights, since it results in particularly appealing properties of the considered problem. In particular, the objective function interpreted in a multiwinner election setup reflects to the well-known Proportional Approval Voting (PAV) rule.

Our main result is that, due to the specific (harmonic) structure of weights, the problem allows constant factor approximation. This is surprising since the problem can be interpreted as a variant of the k-median problem where we do not assume that the connection costs satisfy the triangle inequality. To the best of our knowledge this is the first constant factor approximation algorithm for a variant of k-median that does not require this assumption. The algorithm we propose is based on dependent rounding [Srinivasan, FOCS'01] applied to the solution of a natural LP-relaxation of the problem. The rounding process is well known to produce distributions over integral solutions satisfying *Negative Correlation* (NC), which is usually sufficient for the analysis of approximation guarantees offered by rounding procedures. In our analysis, however, we need to use the fact that the carefully implemented rounding process satisfies a stronger property, called *Negative Association* (NA), which allows us to apply standard concentration bounds for *conditional* random variables.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases approximation algorithms, computational social choice, k-median, dependent rounding, negative association

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.26

Related Version A full version of the paper is available at `https://arxiv.org/abs/1704.02183`.

Funding Jarosław Byrka was supported by the National Science Centre, Poland, grant number 2015/18/E/ST6/00456. Piotr Skowron was supported by a Humboldt Research Fellowship

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 26; pp. 26:1–26:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for Postdoctoral Researchers. Krzysztof Sornat was supported by the National Science Centre, Poland, grant number 2015/17/N/ST6/03684.

Acknowledgements We thank Ola Svensson and Aravind Srinivasan for their helpful and insightful comments.

1 Introduction

This paper considers a general unified framework for two classes of problems: (i) extensions of the k-median problem where clients care about having multiple facilities in their vicinity, and (ii) finding winning committees according to a number of well-known, but hard-to-compute multiwinner election systems[1]. Let us first formalize our framework; we will discuss motivation and explain the relation to k-median and to multiwinner elections later on.

For a natural number $t \in \mathbb{N}$, by $[t]$ we denote the set $\{1, \ldots, t\}$. Let $\mathcal{F} = \{F_1, \ldots, F_m\}$ be the set of m facilities and let $\mathcal{D} = \{D_1, \ldots, D_n\}$ be the set of n clients (demands). The goal is to pick a set of k facilities that altogether are most satisfying for the clients. Different clients can have different preferences over individual facilities – by $c_{i,j}$ we denote the *cost* that client D_j suffers when using facility F_i (this can be, e.g., the communication cost of client D_j to facility F_i, or a value quantifying the level of personal dissatisfaction of D_j from F_i). Following Yager [34], we use ordered weighted average (OWA) operators to define the cost of a client for a bundle of k facilities C. Formally, let $w = \big(w_1, \ldots, w_k\big)$ be a non-increasing vector of k weights. We define the w-cost of a client D_j for a size-k set of facilities C as $w(C, j) = \sum_{i=1}^{k} w_i c_i^{\rightarrow}(C, j)$, where $c^{\rightarrow}(C, j) = (c_1^{\rightarrow}(C, j), \ldots, c_k^{\rightarrow}(C, j)) = \mathrm{sort}_{\mathrm{ASC}}\big(\{c_{i,j} \colon F_i \in C\}\big)$ is a non-decreasing permutation of the costs of client D_j for the facilities from C. Informally speaking, the highest weight is applied to the lowest cost, the second highest weight to the second lowest cost, etc. In this paper we study the following computational problem.

▶ **Definition 1** (OWA k-MEDIAN)**.** In OWA k-MEDIAN we are given a set $\mathcal{D} = \{D_1, \ldots, D_n\}$ of clients, a set $\mathcal{F} = \{F_1, \ldots, F_m\}$ of facilities, a collection of clients' costs $\big(c_{i,j}\big)_{i \in [m], j \in [n]}$, a positive integer k ($k \leq m$), and a vector of k non-increasing weights $w = \big(w_1, \ldots, w_k\big)$. The task is to compute a subset C of $\mathcal{F}$ that minimizes the value

$$w(C) = \sum_{j=1}^{n} w(C, j) = \sum_{j=1}^{n} \sum_{i=1}^{k} w_i c_i^{\rightarrow}(C, j).$$

Note that OWA k-MEDIAN with weights $(1, 0, 0, \ldots, 0)$ is the k-MEDIAN problem. Sometimes the costs represent distances between clients and facilities. Formally, this means that there exists a metric space $\mathcal{M}$ with a distance function $d \colon \mathcal{M} \times \mathcal{M} \to \mathbb{R}_{\geq 0}$, where each client and each facility can be associated with a point in $\mathcal{M}$ so that for each $F_i \in \mathcal{F}$ and each $D_j \in \mathcal{D}$ we have $d(i, j) = c_{i,j}$. When this is the case, we say that the *costs satisfy the triangle inequality*, and use the terms "costs" and "distance" interchangeably. Then, we use the prefix METRIC for the names of our problems. E.g., by METRIC OWA k-MEDIAN we denote the variant of OWA k-MEDIAN where the costs satisfy the triangle inequality.

We are specifically interested in the following two sequences of weights:

[1] We note that multiwinner election rules have many applications beyond the political domain – such applications include finding a set of results a search engine should display [12], recommending a set of products a company should offer to its customers [25, 26], allocating shared resources among agents [29, 28], solving variants of segmentation problems [23], or even improving genetic algorithms [15].

(1) **harmonic:** $w_{\text{har}} = \left(1, 1/2, 1/3, \ldots, 1/k\right)$. By Harmonic k-median we denote the OWA k-median problem with the harmonic vector of weights.
(2) ***p*-geometric:** $w_{\text{geom}} = \left(1, p, p^2, \ldots, p^{k-1}\right)$, for some $p < 1$.

The two aforementioned sequences of weights, w_{har} and w_{geom}, have their natural interpretations, which we discuss later on (for instance, see Examples 3 and 4).

1.1 Motivation

In this subsection we discuss the applicability of the studied model in two settings.

Multiwinner Elections

Different variants of the OWA k-median problem are very closely related to the preference aggregation methods and multiwinner election rules studied in the computational social choice, in particular, and in AI, in general – we summarize this relation in Table 1 and in Figure 1. In particular, one can observe that each "median" problem is associated with a corresponding "winner" problem. Specifically, the k-median problem is known in computational social choice as the Chamberlin–Courant rule. Let us now explain the differences between the winner ("election") and the median ("facility location") problems:

1. The election problems are usually formulated as maximization problems, where instead of (negative) costs we have (positive) utilities. The two variants, the minimization (with costs) and the maximization (with utilities) have the same optimal solutions. Yet, there is a substantial difference in their approximability.
   Approximating the minimization variant is usually much harder. For instance, consider the Chamberlin–Courant (CC) rule which is defined by using the sequence of weights $(1, 0, 0, \ldots, 0)$. In the maximization variant standard arguments can be used to prove that a greedy procedure yields the approximation ratio of $(1 - 1/e)$. This stands in a sharp contrast to the case when the same rule is expressed as the minimization one; in such a case we cannot hope for virtually any approximation [30] (we extend this result in Theorem 21 in [6]). Approximating the minimization variant is also more desired. E.g., a $1/2$-approximation algorithm for (maximization) CC can effectively ignore half of the population of clients, whereas it was argued [30] that a 2-approximation algorithm for the minimization (if existed) would be more powerful. In this paper we study the harder minimization variant, and give the first constant-factor approximation algorithm for the minimization OWA-Winner with the harmonic weights.
2. In facility location problems it is usually assumed that the costs satisfy the triangle inequality. This relates to the previous point: since the problem cannot be well approximated in the general setting, one needs to make additional assumptions. One of our main results is showing that there is a k-median problem (OWA k-median with harmonic weights) that admits a constant-factor approximation without assuming that the costs satisfy the triangle inequality; this is the first known result of this kind.

The special case of Harmonic k-median where each cost belongs to the binary set $\{0, 1\}$ is equivalent to finding winners according to Proportional Approval Voting. The harmonic sequence $w_{\text{har}} = (1, 1/2, 1/3, \ldots, 1/k)$ is in a way exceptional: indeed, PAV can be viewed as an extension of the well known D'Hondt method of apportionment (used for electing parliaments in many contemporary democracies) to the case where the voters can vote for individual candidates rather than for political parties [4]. Further, PAV satisfies several other appealing properties, such as extended justified representation [3]. This is one of the reasons why we are specifically interested in the harmonic weights. For more discussion on PAV and other approval-based rules, we refer the reader to the survey of Kilgour [22].

Table 1 The relation between the k-MEDIAN problems and the corresponding problems studied in AI, in particular in the computational social choice community.

k-median problem	election rule	comment
OWA k-MEDIAN	OWA-Winner [29]	Finding winners according to OWA-Winner rules is the maximization variant of OWA k-MEDIAN (utilities instead of costs).
	Thiele methods [33]	Thiele methods are OWA-Winner rules for 0/1 costs.
HARMONIC k-MEDIAN	PAV [33]	In PAV we assume the 0/1 costs. So far, only the maximization variant was considered in the literature.
k-MEDIAN	Chamberlin–Courant [9]	In CC, usually some specific form of utilities is assumed – different utilities have been considered, but always in the maximization variant (utilities instead of costs).

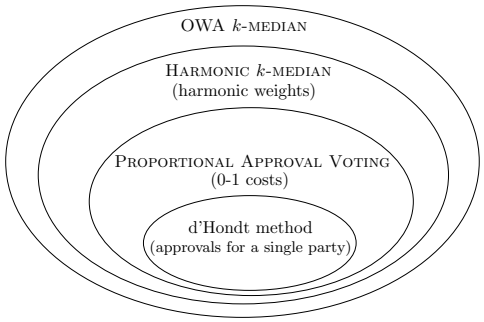


Figure 1 The relation between the considered models. OWA k-MEDIAN is the most general model. PROPORTIONAL APPROVAL VOTING and HARMONIC k-MEDIAN due to the use of harmonic weights can be viewed as natural extensions of the well known and commonly used D'Hondt method of apportionment [4].

OWA k-median as an Extension of k-median

Intuitively, our general formulation extends k-MEDIAN to scenarios where the clients not only use their most preferred facilities, but when there exists a more complex relation of "using the facilities" by the clients. Similar intuition is captured by the FAULT TOLERANT version of the k-MEDIAN problem introduced by Swamy and Shmoys [32] and recently studied by Hajiaghayi et al. [17]. There, the idea is that the facilities can be malfunctioning, and to increase the resilience to their failures each client needs to be connected to several of them.

▶ **Definition 2** (FAULT TOLERANT k-MEDIAN)**.** In FAULT TOLERANT k-MEDIAN problem we are given the same input as in k-MEDIAN, and additionally, for each client D_j we are given a natural number $r_j \geq 1$, called *the connectivity requirement*. The cost of a client D_j is the sum of its costs for the r_j closest open facilities. Similarly as in k-MEDIAN, we aim at choosing at most k facilities so that the sum of the costs is minimized.

When the values $\left(r_j\right)_{j\in[n]}$ are all the same, i.e., if $r_j = r$ for all j, then FAULT TOLERANT k-MEDIAN is called r-FAULT TOLERANT k-MEDIAN and it can be expressed as OWA k-MEDIAN for the weight vector w with r ones followed by $k - r$ zeros. Yet, in the typical

setting of k-MEDIAN problems one additionally assumes that the costs between clients and facilities behave like distances, i.e., that they satisfy the triangle inequality. Indeed, the $(2.675+\epsilon)$-approximation algorithm for k-MEDIAN [5], the 93-approximation algorithm for FAULT TOLERANT k-MEDIAN [17], the 2-approximation algorithm for k-CENTER [18], and the 6.357-approximation algorithm for k-MEANS [1], they all use triangle inequalities. Moreover it can be shown by straightforward reductions from the SET COVER problem that there are no constant factor approximation algorithms for all these settings with general (non-metric) connection costs unless $\mathsf{P}=\mathsf{NP}$.

Using harmonic or geometric OWA weights is also well-justified in case of facility location problems, as illustrated by the following examples.

▶ **Example 3** (Harmonic weights: proportionality)**.** Assume there are $\ell \leq k$ cities, and for $i \in [\ell]$ let N_i denote the set of clients who live in the i-th city. For the sake of simplicity, let us assume that $k \cdot |N_i|$ is divisible by n. Further, assume that the cost of traveling between any two points within a single city is negligible (equal to zero), and that the cost of traveling between different cities is equal to one. Our goal is to decide in which cities the k facilities should be opened; naturally, we set the cost of a client for a facility opened in the same city to zero, and – in another city – to one. Let us consider OWA k-MEDIAN with the harmonic sequence of weights w_{har}. Let n_i denote the number of facilities opened in the i-th city in the optimal solution. We will show that for each i we have $n_i = \frac{k|N_i|}{n}$, i.e., that the number of facilities opened in each city is proportional to its population. Towards a contradiction assume there are two cities, i and j, with $n_i \geq \frac{k|N_i|}{n} + 1$ and $n_j \leq \frac{k|N_j|}{n} - 1$. By closing one facility in the i-th city and opening one in the j-th city, we decrease the total cost by at least:

$$|N_j| \cdot w_{n_j+1} - |N_i| \cdot w_{n_i} = \frac{|N_i|}{n_j+1} - \frac{|N_i|}{n_i} > \frac{|N_j|n}{k|N_j|} - \frac{|N_i|n}{k|N_i|} = 0.$$

Since, we decreased the cost of the clients, this could not be an optimal solution. As a result we see that indeed for each i we have $n_i = \frac{k|N_i|}{n}$.

▶ **Example 4** (Geometric weights: probabilities of failures)**.** Assume that we want to select k facilities and that each client will be using his or her favorite facility only. Yet, when a client wants to use a facility, it can be malfunctioning with some probability p; in such a case the client goes to her second most preferred facility; if the second facility is not working properly, the client goes to the third one, etc. Thus, a client uses her most preferred facility with probability $1-p$, her second most preferred facility with probability $p(1-p)$, the third one with probability $p^2(1-p)$, etc. As a result, the expected cost of a client D_j for the bundle of k facilities C is equal to $w(C,j)$ for the weight vector $w = \big(1-p, (1-p)p, \ldots, (1-p)p^{k-1}\big)$. Finding a set of facilities, that minimize the expected cost of all clients is equivalent to solving OWA k-MEDIAN for the p-geometric sequence of weights (in fact, the sequence that we use is a p-geometric sequence multiplied by $(1-p)$, yet multiplication of the weight vector by a constant does not influence the structure of the optimal solutions).

1.2 Our Results and Techniques

Our main result is showing, that there exists a 2.3589-approximation algorithm for HARMONIC k-MEDIAN for general connection costs (not assuming triangle inequalities). This is in contrast to the innaproximability of most clustering settings with general connection costs.

Our algorithm is based on dependent rounding of a solution to a natural linear program (LP) relaxation of the problem. We use the *dependent rounding* (DR) studied by Srinivasan et al. [31, 16], which transforms in a randomized way a fractional vector into an integral one. The sum-preservation property of DR ensures that exactly k facilities are opened.

DR satisfies, what is well known as *negative correlation (NC)* – intuitively, this implies that the sums of subsets of random variables describing the outcome are more centered around their expected values than if the fractional variables were rounded independently. More precisely, negative correlation allows one to use standard concentration bounds such as the Chernoff-Hoeffding bound. Yet, interestingly, we find out that NC is not sufficient for our analysis in which we need a conditional variant of the concentration bound. The property that is sufficient for conditional bounds is *negative association* (NA) [20]. In fact its special case that we call *binary negative association* (BNA), is sufficient for our analysis. It captures the capability of reasoning about conditional probabilities. Thus, our work demonstrates how to apply the (B)NA property in the analysis of approximation algorithms based on DR. To the best of our knowledge, Harmonic k-median is the first natural computational problem, where it is essential to use BNA in the analysis of the algorithm.

We additionally show that the 93-approximation algorithm of Hajiaghayi et al. [17] can be extended to OWA k-median (our technique is summarized in Section 3) – this time we additionally need to assume that the costs satisfy the triangle inequality. Indeed, without this assumption the problem is hard to approximate for a large class of weight vectors; for instance, for p-geometric sequences with $p < 1/e$ or for sequences where there exists $\lambda \in (0, 1)$ such that clients care only about the λ-fraction of opened facilities. Due to space constraints the formulation and the discussion on these hardness results are redelegated to the full version of the paper [6, Appendix E].

For the paper to be self-contained in [6, Appendix A] we discuss in detail the process of dependent rounding (including a few illustrative examples); in particular, we provide an alternative proof that DR satisfies binary negative association. Our proof is more direct and shorter than the proofs known in the literature [24].

2 Harmonic k-median and Proportional Approval Voting: a 2.3589-approximation Algorithm

In this section we demonstrate how to use the Binary Negative Association (BNA) property of Dependent Rounding (DR) to derive our main result – a randomized constant-factor approximation algorithm for Harmonic k-median. In [6, Appendix A] we provide a detailed discussion on DR and BNA, including a proof that DR satisfies BNA, and several examples.

▶ **Theorem 5.** *There exists a polynomial time randomized algorithm for* Harmonic k-median *that gives* 2.3589*-approximation in expectation.*

▶ **Corollary 6.** *There exists a polynomial time randomized algorithm for the minimization* Proportional Approval Voting *that gives* 2.3589*-approximation in expectation.*

In the remainder of this section we will prove the statement of Theorem 5. Consider the following linear program (1–5) that is a relaxation of a natural ILP for Harmonic k-median.

$$\min \quad \sum_{j=1}^{n}\sum_{\ell=1}^{k}\sum_{i=1}^{m} w_\ell \cdot x_{ij}^{\ell} \cdot c_{ij} \qquad (1)$$

$$\sum_{i=1}^{m} y_i = k \qquad (2)$$

$$\sum_{\ell=1}^{k} x_{ij}^{\ell} \le y_i \qquad \forall i \in [m],\ j \in [n] \qquad (3)$$

$$\sum_{i=1}^{m} x_{ij}^{\ell} \ge 1 \qquad \forall j \in [n],\ \ell \in [k] \qquad (4)$$

$$y_i, x_{ij}^{\ell} \in [0,1] \qquad \forall i \in [m],\ j \in [n],\ \ell \in [k] \qquad (5)$$

The intuitive meaning of the variables and constraints of the above LP is as follows. Variable y_i denotes how much facility F_i is opened. Integral values 1 and 0 correspond

to, respectively, opening and not opening the i-th facility. Constraint (2) encodes opening exactly k facilities. Each client $D_j \in \mathcal{D}$ has to be assigned to each among k opened facilities with different weights. For that we copy each client k times: the ℓ-th copy of a client D_j is assigned to the ℓ-th closest to D_j open facility. Variable x_{ij}^{ℓ} denotes how much the ℓ-th copy of D_j is assigned to facility F_i. In an integral solution we have $x_{ij}^{\ell} \in \{0,1\}$, which means that the ℓ-th copy of a client can be either assigned or not to the respective facility. The objective function (1) encodes the cost of assigning all copies of all clients to the opened facilities, applying proper weights. Constraint (3) prevents an assignment of a copy of a client to a not-opened part of a facility. In an integer solution it also forces assigning different copies of a client to different facilities. Observe that, due to non-increasing weights w_ℓ, the objective (1) is smaller if an ℓ'-th copy of a client is assigned to a closer facility than an ℓ''-th copy, whenever $\ell' < \ell''$. Constraint (4) ensures that each copy of a client is served by some facility.

Just like in most facility location settings it is crucial to select the facilities to open, and the later assignment of clients to facilities can be done optimally by a simple greedy procedure. We propose to select the set of facilities in a randomized way by applying the DR procedure to the y vector from an optimal fractional solution to linear program (1–5). This turns out to be a surprisingly effective methodology for Harmonic k-median.

2.1 Analysis of the Algorithm

Let $\mathrm{OPT}^{\mathrm{LP}}$ be the value of an optimal solution (x^*, y^*) to the linear program (1–5). Let OPT be the value of an optimal solution $(x^{\mathrm{OPT}}, y^{\mathrm{OPT}})$ for Harmonic k-median. Easily we can see that $(x^{\mathrm{OPT}}, y^{\mathrm{OPT}})$ is a feasible solution to the linear program (1–5), so $\mathrm{OPT}^{\mathrm{LP}} \leq \mathrm{OPT}$. Let $Y = (Y_1, \ldots, Y_m)$ be the random solution obtained by applying the DR procedure described in [6, Appendix A] to the vector y^*. Recall that DR preserves the sum of entries (see [6, Appendix A]), hence we have exactly k facilities opened. It is straightforward to assign clients to the open facilities, so the variables $X = (X_{ij}^{\ell})_{j\in[n],i\in[m],\ell\in[k]}$ are easily determined.

We will show that $\mathbb{E}[\mathrm{cost}(Y)] \leq 2.3589 \cdot \mathrm{OPT}^{\mathrm{LP}}$. In fact, we will show that $\mathbb{E}[\mathrm{cost}_j(Y)] \leq 2.3589 \cdot \mathrm{OPT}_j^{\mathrm{LP}}$, where the subindex j extracts the cost of assigning client D_j to the facilities in the solution returned by the algorithm. In our analysis we focus on a single client $D_j \in \mathcal{D}$. Next, we reorder the facilities $\{F_1, F_2, \ldots, F_m\}$ in the non-decreasing order of their connection costs to D_j (i.e., in the non-decreasing order of c_{ij}). Thus, from now on, facility F_i is the i-th closest facility to client D_j; ties are resolved in an arbitrary but fixed way.

The ordering of the facilities is depicted in Figure 2, which also includes information about the fractional opening of facilities in y^*, i.e., facility F_i is represented by an interval of length y_i^*. The total length of all intervals equals k. Next, we subdivide each interval into a set of (small) ϵ-size pieces (called ϵ-subintervals); ϵ is selected so that $1/\epsilon$, and y_i^*/ϵ for each i, are integers. Note that the values y_i^*, which originate from the solution returned by an LP solver, are rational numbers. The subdivision of $[0, k]$ into ϵ-subintervals is shown in Figure 2 on the "$(Z_r)_{r\in\{1,2,\ldots,k/\epsilon\}}$" level.

The idea behind introducing the ϵ-subintervals is the following. Although computationally the algorithm applies DR to the y^* variables, for the sake of the analysis we may think that the DR process is actually rounding z variables corresponding to ϵ-subinterval under the additional assumption that rounding within individual facilities is done before rounding between facilities. Formally, we replace the vector $Y = (Y_1, Y_2, \ldots, Y_m)$ by an equivalent vector of random variables $Z = (Z_1, Z_2, \ldots, Z_{k/\epsilon})$. Random variable Z_r represents the r-th ϵ-subinterval. We will use the following notation to describe the bundles of ϵ-subintervals

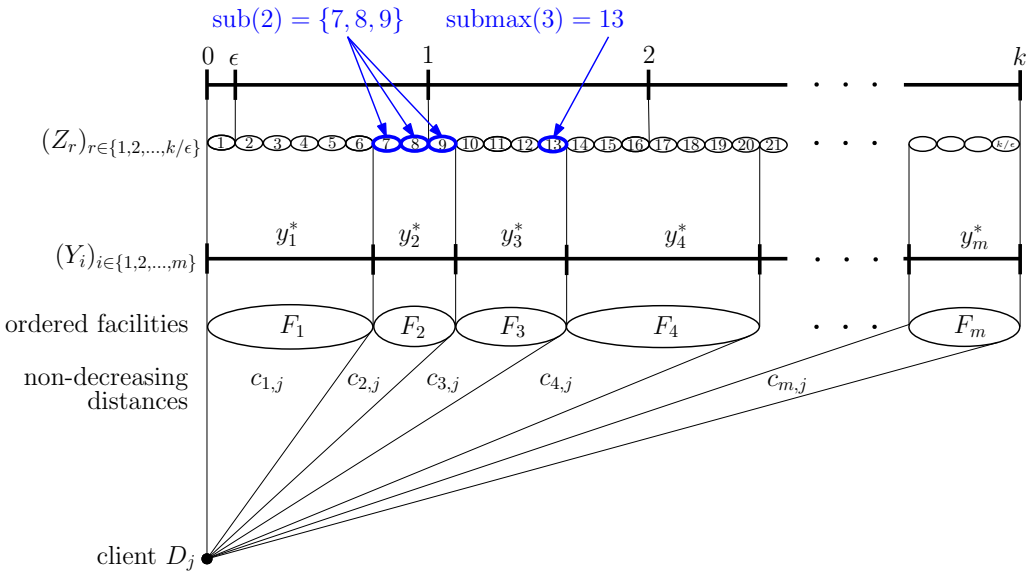


Figure 2 Ordering of the facilities by $c_{i,j}$ for the chosen client D_j. Definitions of the variables Y_i, Z_r and of the indices $\mathrm{sub}(i)$ and $\mathrm{submax}(i)$.

that correspond to particular facilities:

$$\mathrm{submax}(0) = 0 \qquad \text{and} \qquad \mathrm{submax}(i) = \mathrm{submax}(i-1) + \frac{y_i^*}{\epsilon}, \tag{6}$$

$$\mathrm{sub}(i) = \{\mathrm{submax}(i-1)+1, \ldots, \mathrm{submax}(i)\}. \tag{7}$$

Intuitively, $\mathrm{sub}(i)$ is the set of indexes r such that Z_r represents an interval belonging to the i-th facility. Examples for both definitions are shown in Figure 2 in the upper level. Formally, the random variables Z_r are defined so that:

$$Y_i = \sum_{r \in \mathrm{sub}(i)} Z_r \qquad \text{and} \qquad Y_i = 1 \implies \exists!\, r \in \mathrm{sub}(i) \quad Z_r = 1. \tag{8}$$

For each $r \in \{1, 2, \ldots, k/\epsilon\}$ we can write that:

$$\Pr[Z_r = 1] = \Pr[Z_r = 1 \big| Y_{\mathrm{sub}^{-1}(r)} = 1] \cdot \Pr[Y_{\mathrm{sub}^{-1}(r)} = 1] = \frac{\epsilon}{y^*_{\mathrm{sub}^{-1}(r)}} \cdot y^*_{\mathrm{sub}^{-1}(r)} = \epsilon \tag{9}$$

and $\Pr[Z_r = 0] = 1 - \epsilon$, hence $\mathbb{E}[Z_r] = \epsilon$. Also we have:

$$\Pr\left[Y_i = 1\right] = \Pr\left[\sum_{r \in \mathrm{sub}(i)} Z_r = 1\right] = \Pr\left[\bigvee_{r \in \mathrm{sub}(i)} Z_r = 1\right] = \sum_{r \in \mathrm{sub}(i)} \Pr\left[Z_r = 1\right]. \tag{10}$$

When $Y_i = 1$ its representative is chosen randomly among $(Z_r)_{r \in \mathrm{sub}(i)}$ independently of the choices of representatives of other facilities. Therefore

$$\forall_{i \in [m]} \quad \forall_{r \in \mathrm{sub}(i)} \quad \mathbb{E}\left[f(Y) \mid Y_i = 1\right] = \mathbb{E}\left[f(Y) \mid Y_i = 1 \ \wedge \ Z_r = 1\right], \tag{11}$$

for any function f on vector $Y = (Y_1, Y_2, \ldots, Y_m)$. Now we are ready to analyze the expected cost for any client $D_j \in \mathcal{D}$. Here we use the special assumption on the harmonic weights.

$$
\begin{aligned}
\mathbb{E}[\mathrm{cost}_j(Y)] \leq & \sum_{i=1}^{m}\left(\mathbb{E}\left[\frac{c_{ij}}{1+\sum_{i'=1}^{i-1}Y_{i'}}\middle| Y_i=1\right]\cdot\Pr\left[Y_i=1\right]\right)\\
\overset{(10)}{=} & \sum_{i=1}^{m}\left(c_{ij}\cdot\mathbb{E}\left[\frac{1}{1+\sum_{i'=1}^{i-1}Y_{i'}}\middle| Y_i=1\right]\cdot\sum_{r\in\mathrm{sub}(i)}\Pr\left[Z_r=1\right]\right)\\
= & \sum_{i=1}^{m}\left(c_{ij}\cdot\sum_{r\in\mathrm{sub}(i)}\mathbb{E}\left[\frac{1}{1+\sum_{i'=1}^{i-1}Y_{i'}}\middle| Y_i=1\right]\cdot\Pr\left[Z_r=1\right]\right)\\
\overset{(11)}{=} & \sum_{i=1}^{m}\left(c_{ij}\cdot\sum_{r\in\mathrm{sub}(i)}\mathbb{E}\left[\frac{1}{1+\sum_{i'=1}^{i-1}Y_{i'}}\middle| Y_i=1\wedge Z_r=1\right]\cdot\Pr\left[Z_r=1\right]\right)\\
\overset{(8),(9)}{=} & \sum_{i=1}^{m}\left(\epsilon\cdot c_{ij}\cdot\sum_{r\in\mathrm{sub}(i)}\mathbb{E}\left[\frac{1}{1+\sum_{r'=1}^{\mathrm{submax}(i-1)}Z_{r'}}\middle| Z_r=1\right]\right)\\
\overset{(8)}{=} & \sum_{i=1}^{m}\left(\epsilon\cdot c_{ij}\cdot\sum_{r\in\mathrm{sub}(i)}\mathbb{E}\left[\frac{1}{1+\sum_{r'=1}^{r-1}Z_{r'}}\middle| Z_r=1\right]\right) \qquad (12)
\end{aligned}
$$

W.l.o.g., assume that $\mathrm{OPT}_j^{\mathrm{LP}}>0$. Hence the approximation ratio for any client D_j is

$$
\frac{\mathbb{E}[\mathrm{cost}_j(Y)]}{\mathrm{OPT}_j^{\mathrm{LP}}}\overset{(7),(12)}{\leq}\frac{\sum_{r=1}^{k/\epsilon}\epsilon\cdot c_{\mathrm{sub}^{-1}(r),j}\cdot\mathbb{E}\left[\frac{1}{1+\sum_{r'=1}^{r-1}Z_{r'}}\middle| Z_r=1\right]}{\sum_{r=1}^{k/\epsilon}\epsilon\cdot c_{\mathrm{sub}^{-1}(r),j}\cdot\frac{1}{\lceil r\epsilon\rceil}}=
$$

note that $\mathrm{sub}^{-1}(r)$ is an index of a facility that contains Z_r. Now we convert the sum over facilities into a sum over unit intervals. A unit interval is represented as a sum of $1/\epsilon$ many ϵ-subintervals:

$$
=\frac{\sum_{\ell=1}^{k}\sum_{r=(\ell-1)/\epsilon+1}^{\ell/\epsilon}c_{\mathrm{sub}^{-1}(r),j}\cdot\mathbb{E}\left[\frac{1}{1+\sum_{r'=1}^{r-1}Z_{r'}}\middle| Z_r=1\right]}{\sum_{\ell=1}^{k}\sum_{r=(\ell-1)/\epsilon+1}^{\ell/\epsilon}c_{\mathrm{sub}^{-1}(r),j}\cdot\frac{1}{\ell}}\leq
$$

W.l.o.g., we can assume that first interval has non-zero costs: $\sum_{r=1}^{1/\epsilon}c_{\mathrm{sub}^{-1}(r),j}>0$, otherwise the LP pays 0 and our algorithm pays 0 in expectation on intervals from non-empty prefix of $(1,2,\ldots,k)$. With this assumption we can take maximum over intervals:

$$
\overset{\text{Lemma 17 in [6]}}{\leq}\max_{\ell\in[k]}\left(\frac{\sum_{r=(\ell-1)/\epsilon+1}^{\ell/\epsilon}c_{\mathrm{sub}^{-1}(r),j}\cdot\mathbb{E}\left[\frac{1}{1+\sum_{r'=1}^{r-1}Z_{r'}}\middle| Z_r=1\right]}{\sum_{r=(\ell-1)/\epsilon+1}^{\ell/\epsilon}c_{\mathrm{sub}^{-1}(r),j}\cdot\frac{1}{\ell}}\right)\leq
$$

Costs $c_{\mathrm{sub}^{-1}(r),j}$ can be general and they could be hard to analyze. Therefore we would like to remove costs from the analysis. We will use Lemma 18 from [6] for which the technique of

splitting variables Y_i into Z_r was needed. We are using the fact that the variables Z_r have the same expected values; otherwise the coefficient in front of the expected value would be $c_{ij} \cdot y_i^*$, i.e., not monotonic. Thus

$$\overset{\text{Lemma 18 in [6]}}{\leq} \max_{\ell \in [k]} \left(\epsilon \cdot \ell \cdot \sum_{r=(\ell-1)/\epsilon+1}^{\ell/\epsilon} \mathbb{E}\left[\frac{1}{1+\sum_{r'=1}^{r-1} Z_{r'}} \middle| Z_r = 1 \right] \right). \tag{13}$$

Consider the expected value in the above expression for a fixed $r \in \{(\ell-1)/\epsilon + 1, \ldots, \ell/\epsilon\}$:

$$\begin{aligned} E_r &= \mathbb{E}\left[\frac{1}{1+\sum_{r'=1}^{r-1} Z_{r'}} \middle| Z_r = 1 \right] = \sum_{t=1}^{k} \frac{1}{t} \Pr\left[\sum_{r'=1}^{r-1} Z_{r'} = t-1 \middle| Z_r = 1 \right] = \\ &= \sum_{t=1}^{\ell} \frac{1}{t} \Pr\left[\sum_{r'=1}^{r-1} Z_{r'} = t-1 \middle| Z_r = 1 \right] + \sum_{t=\ell+1}^{k} \frac{1}{t} \Pr\left[\sum_{r'=1}^{r-1} Z_{r'} = t-1 \middle| Z_r = 1 \right]. \end{aligned} \tag{14}$$

For $t \in \{1, 2, \ldots, \ell\}$ we consider the conditional probability in the above expression, denote it by $p_r(t-1)$, and analyze the corresponding cumulative distribution function $H_r(t-1)$:

$$p_r(t-1) = \Pr\left[\sum_{r'=1}^{r-1} Z_{r'} = t-1 \middle| Z_r = 1 \right], \tag{15}$$

$$H_r(t-1) = \Pr\left[\sum_{r'=1}^{r-1} Z_{r'} \leq t-1 \middle| Z_r = 1 \right] = \sum_{t'=0}^{t-1} p_r(t'), \tag{16}$$

We continue the analysis of E_r:

$$\begin{aligned} E_r &\overset{(14),(15)}{=} \sum_{t=1}^{\ell} \frac{1}{t} p_r(t-1) + \sum_{t=\ell+1}^{k} \frac{1}{t} p_r(t-1) \\ &\overset{(16)}{=} H_r(0) + \sum_{t=2}^{\ell} \frac{1}{t} \left(H_r(t-1) - H_r(t-2) \right) + \sum_{t=\ell+1}^{k} \frac{1}{t} p_r(t-1) \\ &= H_r(0) + \sum_{t=2}^{\ell} \frac{1}{t} H_r(t-1) - \sum_{t=2}^{\ell} \frac{1}{t} H_r(t-2) + \sum_{t=\ell+1}^{k} \frac{1}{t} p_r(t-1) \\ &= \sum_{t=1}^{\ell} \frac{1}{t} H_r(t-1) - \sum_{t=1}^{\ell-1} \frac{1}{t+1} H_r(t-1) + \sum_{t=\ell+1}^{k} \frac{1}{t} p_r(t-1) \\ &= \sum_{t=1}^{\ell-1} \frac{1}{t} H_r(t-1) - \sum_{t=1}^{\ell-1} \frac{1}{t+1} H_r(t-1) + \frac{1}{\ell} H_r(\ell-1) + \sum_{t=l+1}^{k} \frac{1}{t} p_r(t-1) \\ &\leq \sum_{t=1}^{\ell-1} \left(\frac{1}{t} - \frac{1}{t+1} \right) H_r(t-1) + \frac{1}{\ell} \left(H_r(\ell-1) + \sum_{t=\ell+1}^{k} p_r(t-1) \right) \\ &= \sum_{t=1}^{\ell-1} \frac{1}{t(t+1)} H_r(t-1) + \frac{1}{\ell} \left(H_r(\ell-1) + \sum_{t=\ell+1}^{k} p_r(t-1) \right) \\ &\leq \sum_{t=1}^{\ell-1} \frac{1}{t(t+1)} H_r(t-1) + \frac{1}{\ell}. \end{aligned} \tag{17}$$

▶ **Lemma 7.** *For any $\ell \in [k]$, $t \in [\ell-1]$ and $r \in \{(\ell-1)/\epsilon + 1, (\ell-1)/\epsilon + 2, \ldots, \ell/\epsilon\}$ we have*

$$H_r(t-1) \leq e^{-r \cdot \epsilon} \cdot \left(\frac{e \cdot r \cdot \epsilon}{t} \right)^t.$$

The proof of Lemma 7 combines the use of the BNA property of variables $\{Z_1, Z_2, \ldots, Z_{k/\epsilon}\}$ with applications of Chernoff-Hoeffding bounds. Due to the space constraints, the proof is moved to the full version of the paper [6, Appendix C]. In the end, we get the following bound on the approximation ratio.

▶ **Lemma 8.** *For any $j \in [n]$ we have*

$$\frac{\mathbb{E}[\text{cost}_j(Y)]}{\text{OPT}_j^{\text{LP}}} \leq 2.3589.$$

A proof uses inequalities (13), (17) as well as Lemma 7 with an upper bound derived by an integral of the function $f_t(x) = e^{-x}$. We made numerical calculation for $\ell \in \{1, 2, \ldots, 88\}$ and for other case we used Stirling formula and Taylor series for e^{ℓ} to derive analytical upper bound. Full proof, including a plot of numericaly obtained values, is presented in [6, Appendix C].

3 OWA k-median with Costs Satisfying the Triangle Inequality

In this section we construct an algorithm for OWA k-median with costs satisfying the triangle inequality. Thus, the problem we address in this section is more general than Harmonic k-median (i.e., the problem we have considered in the previous section) in a sense that we allow for arbitrary non-increasing sequences of weights. On the other hand, it is less general in a sense that we require the costs to form a specific structure (a metric).

In our approach we first adapt the algorithm of Hajiaghayi et al. [17] for Fault Tolerant k-median so that it applies to the following, slightly more general setting: for each client D_j we introduce its multiplicity $m_j \in \mathbb{N}$ – intuitively, this corresponds to cloning D_j and co-locating all such clones in the same location as D_j. However, this will require a modification of the original algorithm for Fault Tolerant k-median, since we want to allow the multiplicities $\{m_j\}_{D_j \in \mathcal{D}}$ to be exponential with respect to the size of the instance (otherwise, we could simply copy each client a sufficient number of times, and use the original algorithm of Hajiaghayi et al.).

Next, we provide a reduction from OWA k-median to such a generalization of Fault Tolerant k-median. The resulting Fault Tolerant k-median with Clients Multiplicities problem can be cast as the following integer program:

$$\begin{aligned}
\min \quad & \sum_{j=1}^{n}\sum_{i=1}^{m} m_j \cdot x_{ij} \cdot c_{ij} && \sum_{i=1}^{m} x_{ij} = r_j && \forall j \in [n] \\
& \sum_{i=1}^{m} y_i = k && x_{ij} \leq y_i && \forall i \in [m],\ j \in [n] \\
& && y_i, x_{ij} \in \{0,1\} && \forall i \in [m] \\
& && m_j \in \mathbb{N} && \forall j \in [n]
\end{aligned}$$

▶ **Theorem 9.** *There is a polynomial-time* 93*-approximation algorithm for* Metric Fault Tolerant k-median with Clients Multiplicities.

Proof can be found in [6, Appendix D]. Consider reduction from OWA k-median to Fault Tolerant k-median with Clients Multiplicities depicted on Figure 3.

▶ **Lemma 10.** *Let I be an instance of* OWA k-median*, and let I' be an instance of* Fault Tolerant k-median with Clients Multiplicities *constructed from I through reduction from Figure 3. An α-approximate solution to I' is also an α-approximate solution to I.*

Proof can be found in [6, Appendix D].

▶ Reduction. Let us take an instance I of OWA k-median $\left(\mathcal{D}, \mathcal{F}, k, w, \{c_{ij}\}_{F_i \in \mathcal{F}, D_j \in \mathcal{D}}\right)$ where $w_i = \frac{p_i}{q_i}, i \in [k]$ are rational numbers in the canonical form. We construct an instance I' of Fault Tolerant k-median with Clients Multiplicities with the same set of facilities and the same number of facilities to open, k. Each client $D_j \in \mathcal{D}$ is replaced with clients $D_{j,1}, D_{j,2}, \ldots, D_{j,k}$ with requirements $1, 2, \ldots, k$, respectively. For $Q = \prod_{r=1}^{k} q_r$, the multiples of the clients are defined as follows:

- $m_{j,\ell} = (w_\ell - w_{\ell+1}) \cdot Q$, for each $\ell \in [k-1]$, and
- $m_{j,k} = w_k \cdot Q$.

Figure 3 Reduction from OWA k-median to Fault Tolerant k-median with Clients Multiplicities.

▶ **Corollary 11.** *There exists a* 93*-approximation algorithm for* Metric OWA k-median *that runs in polynomial time.*

4 Concluding Remarks and Open Questions

We have introduced a new family of k-median problems, called OWA k-median, and we have shown that our problem with the harmonic sequence of weights allows for a constant factor approximation even for general (non-metric) costs. This algorithm applies to Proportional Approval Voting. In the analysis of our approximation algorithm for Harmonic k-median, we used the fact that the dependent rounding procedure satisfies Binary Negative Association.

We showed that any Metric OWA k-median can be approximated within a factor of 93 via a reduction to Fault Tolerant k-median with Clients Multiplicities. We also obtained that OWA k-median with p-geometric weights with $p < 1/e$ cannot be approximated without the assumption of the costs being metric. The status of the non-metric problem with p-geometric weights with $p > 1/e$ remains an intriguing open problem.

Using approximation and randomized algorithms for finding winners of elections requires some comment. First, the multiwinner election rules such as PAV have many applications in the voting theory, recommendation systems and in resource allocation. Using (randomized) approximation algorithms in such scenarios is clearly justified. However, even for other high-stake domains, such as political elections, the use of approximation algorithms is a promising direction. One approach is to view an approximation algorithm as a new, full-fledged voting rule (for more discussion on this, see the works of Caragiannis et al. [7, 8], Skowron et al. [30], and Elkind et al. [13]). In fact, the use of randomized algorithms in this context has been advocated in the literature as well – e.g., one can arrange an election where each participant is allowed to suggest a winning committee, and the best out of the suggested committees is selected; in such case the approximation guaranty of the algorithm corresponds to the quality of the outcome of elections (for a more detailed discussion see [30]) [2]. Nonetheless, we think that it would be beneficial to learn whether our algorithm can be efficiently derandomized.

[2] Indeed, approximation algorithms for many election rules have been extensively studied in the literature. In the world of single-winner rules, there are already very good approximation algorithms known for the Kemeny's rule [2, 10, 21] and for the Dodgson's rule [27, 19, 7, 14, 8]. A hardness of approximation has been proven for the Young's rule [7]. For the multiwinner case we know good (randomized) approximation algorithms for Minimax Approval Voting [11], Chamberlin–Courant rule [30], Monroe rule [30], or maximization variant of PAV [29].

References

1 S. Ahmadian, A. Norouzi-Fard, O. Svensson, and J. Ward. Better Guarantees for k-Means and Euclidean k-Median by Primal-Dual Algorithms. In *Proceedings of the 58th IEEE Symposium on Foundations of Computer Science*, pages 61–72, 2017. doi:10.1109/FOCS.2017.15.

2 N. Ailon, M. Charikar, and A. Newman. Aggregating Inconsistent Information: Ranking and Clustering. *Journal of the ACM*, 55(5):23:1–23:27, 2008. doi:10.1145/1411509.1411513.

3 H. Aziz, M. Brill, V. Conitzer, E. Elkind, R. Freeman, and T. Walsh. Justified Representation in Approval-Based Committee Voting. *Social Choice and Welfare*, 48(2):461–485, 2017. doi:10.1007/s00355-016-1019-3.

4 M. Brill, J-F. Laslier, and P. Skowron. Multiwinner Approval Rules as Apportionment Methods. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 414–420, 2017. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14782.

5 J. Byrka, T. Pensyl, B. Rybicki, A. Srinivasan, and K. Trinh. An Improved Approximation for k-Median and Positive Correlation in Budgeted Optimization. *ACM Transactions on Algorithms*, 13(2):23:1–23:31, 2017. doi:10.1145/2981561.

6 J. Byrka, P. Skowron, and K. Sornat. Proportional Approval Voting, Harmonic k-Median, and Negative Association. *CoRR*, abs/1704.02183v3, 2017.

7 I. Caragiannis, J. A. Covey, M. Feldman, C. M. Homan, C. Kaklamanis, N. Karanikolas, A. D. Procaccia, and J. S. Rosenschein. On the Approximability of Dodgson and Young Elections. *Artificial Intelligence*, 187:31–51, 2012. doi:10.1016/j.artint.2012.04.004.

8 I. Caragiannis, C. Kaklamanis, N. Karanikolas, and A. D. Procaccia. Socially Desirable Approximations for Dodgson's Voting Rule. *ACM Transactions on Algorithms*, 10(2):6:1–6:28, 2014. doi:10.1145/2556950.

9 B. Chamberlin and P. Courant. Representative Deliberations and Representative Decisions: Proportional Representation and the Borda Rule. *American Political Science Review*, 77(3):718–733, 1983.

10 D. Coppersmith, L. Fleischer, and A. Rudra. Ordering by Weighted Number of Wins Gives a Good Ranking for Weighted Tournaments. *ACM Transactions on Algorithms*, 6(3):55:1–55:13, 2010. doi:10.1145/1798596.1798608.

11 M. Cygan, Ł. Kowalik, A. Socała, and K. Sornat. Approximation and Parameterized Complexity of Minimax Approval Voting. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 459–465, 2017 (full version will appear in Journal of Artificial Intelligence Research).

12 C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank Aggregation Methods for the Web. In *Proceedings of the 10th International World Wide Web Conference*, pages 613–622, 2001.

13 E. Elkind, P. Faliszewski, P. Skowron, and A.i Slinko. Properties of Multiwinner Voting Rules. *Social Choice and Welfare*, 48(3):599–632, 2017. doi:10.1007/s00355-017-1026-z.

14 P. Faliszewski, E. Hemaspaandra, and L. A. Hemaspaandra. Multimode Control Attacks on Elections. *Journal of Artificial Intelligence Research*, 40:305–351, 2011. doi:10.1613/jair.3136.

15 P. Faliszewski, J. Sawicki, R. Schaefer, and M. Smolka. Multiwinner Voting in Genetic Algorithms for Solving Ill-Posed Global Optimization Problems. In *Proceedings of the 19th International Conference on the Applications of Evolutionary Computation*, pages 409–424, 2016. doi:10.1007/978-3-319-31204-0_27.

16 R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan. Dependent Rounding and Its Applications to Approximation Algorithms. *Journal of the ACM*, 53(3):324–360, 2006.

17 M. T. Hajiaghayi, W. Hu, J. Li, S. Li, and B. Saha. A Constant Factor Approximation Algorithm for Fault-Tolerant k-Median. *ACM Transactions on Algorithms*, 12(3):36:1–36:19, 2016. doi:10.1145/2854153.

18 D.S. Hochbaum and D.B. Shmoys. A Best Possible Heuristic for the k-Center Problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

19 C. M. Homan and L. A. Hemaspaandra. Guarantees for the Success Frequency of an Algorithm for Finding Dodgson-Election Winners. *Journal of Heuristics*, 15(4):403–423, 2009. doi:10.1007/s10732-007-9068-5.

20 K. Joag-Dev and F. Proschan. Negative Association of Random Variables with Applications. *The Annals of Statistics*, 11(1):286–295, 1983.

21 C. Kenyon-Mathieu and W. Schudy. How to Rank with Few Errors. In *Proceedings of the 39th ACM Symposium on Theory of Computing*, pages 95–103, 2007. doi:10.1145/1250790.1250806.

22 D. Kilgour. Approval Balloting for Multi-Winner Elections. In J. Laslier and R. Sanver, editors, *Handbook on Approval Voting*, pages 105–124. Springer, 2010.

23 J. M. Kleinberg, C. H. Papadimitriou, and P. Raghavan. Segmentation Problems. *Journal of the ACM*, 51(2):263–280, 2004. doi:10.1145/972639.972644.

24 J. B. Kramer, J. Cutler, and A. J. Radcliffe. Negative Dependence and Srinivasan's Sampling Process. *Combinatorics, Probability and Computing*, 20(3):347–361, 2011.

25 T. Lu and C. Boutilier. Budgeted Social Choice: From Consensus to Personalized Decision Making. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 280–286, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-057.

26 T. Lu and C. Boutilier. Value-Directed Compression of Large-Scale Assignment Problems. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 1182–1190, 2015. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9557.

27 J. C. McCabe-Dansted, G. Pritchard, and A. M. Slinko. Approximability of Dodgson's rule. *Social Choice and Welfare*, 31(2):311–330, 2008. doi:10.1007/s00355-007-0282-8.

28 B. Monroe. Fully Proportional Representation. *American Political Science Review*, 89(4):925–940, 1995.

29 P. Skowron, P. Faliszewski, and J. Lang. Finding a Collective Set of Items: From Proportional Multirepresentation to Group Recommendation. *Artificial Intelligence*, 241:191–216, 2016. doi:10.1016/j.artint.2016.09.003.

30 P. Skowron, P. Faliszewski, and A. M. Slinko. Achieving Fully Proportional Representation: Approximability Results. *Artificial Intelligence*, 222:67–103, 2015. doi:10.1016/j.artint.2015.01.003.

31 A. Srinivasan. Distributions on Level-Sets with Applications to Approximation Algorithms. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 588–597, 2001.

32 C. Swamy and D.B. Shmoys. Fault-Tolerant Facility Location. *ACM Transactions on Algorithms*, 4(4):51:1–51:27, 2008.

33 T. N. Thiele. Om flerfoldsvalg. In *Oversigt over det Kongelige Danske Videnskabernes Selskabs Forhandlinger*, pages 415–441. København: A.F. Høst., 1895.

34 R. R. Yager. On Ordered Weighted Averaging Aggregation Operators in Multicriteria Decisionmaking. *IEEE Trans. Systems, Man, and Cybernetics*, 18(1):183–190, 1988. doi:10.1109/21.87068.

Fine-Grained Derandomization: From Problem-Centric to Resource-Centric Complexity

Marco L. Carmosino[1]
Department of Computer Science, University of California San Diego, La Jolla, CA, USA
marco@ntime.org

Russell Impagliazzo[2]
Department of Computer Science, University of California San Diego, La Jolla, CA, USA
russell@cs.ucsd.edu

Manuel Sabin[3]
Computer Science Division, University of California Berkeley, Berkeley, CA, USA
msabin@berkeley.edu

Abstract

We show that popular hardness conjectures about problems from the field of fine-grained complexity theory imply structural results for resource-based complexity classes. Namely, we show that if either k-Orthogonal Vectors or k-CLIQUE requires $n^{\epsilon k}$ time, for some constant $\epsilon > 1/2$, to *count* (note that these conjectures are significantly weaker than the usual ones made on these problems) on randomized machines for all but finitely many input lengths, then we have the following derandomizations:

- BPP can be decided in polynomial time *using only* n^{α} *random bits* on average over *any* efficient input distribution, for any constant $\alpha > 0$
- BPP can be decided in polynomial time with *no randomness* on average *over the uniform distribution*

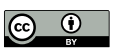
This answers an open question of Ball et al. (STOC '17) in the positive of whether derandomization can be achieved from conjectures from fine-grained complexity theory. More strongly, these derandomizations improve over all previous ones achieved from *worst-case uniform* assumptions by succeeding on all but finitely many input lengths. Previously, derandomizations from worst-case uniform assumptions were only know to succeed on infinitely many input lengths. It is specifically the structure and moderate hardness of the k-Orthogonal Vectors and k-CLIQUE problems that makes removing this restriction possible.

Via this uniform derandomization, we connect the problem-centric and resource-centric views of complexity theory by showing that exact hardness assumptions about specific problems like k-CLIQUE imply quantitative and qualitative relationships between randomized and deterministic time. This can be either viewed as a barrier to proving some of the main conjectures of fine-grained complexity theory lest we achieve a major breakthrough in unconditional derandomization or, optimistically, as route to attain such derandomizations by working on very concrete and weak conjectures about specific problems.

2012 ACM Subject Classification Theory of computation → Complexity classes

Keywords and phrases Derandomization, Hardness vs Randomness, Fine-Grained Complexity, Average-Case Complexity, k-Orthogonal Vectors, k-CLIQUE

[1] Supported by the Simons Foundation
[2] Supported by the Simons Foundation
[3] Supported by the National Science Foundation Graduate Research Fellowship under Grant #DGE-1106400



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 27; pp. 27:1–27:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.27

Related Version A full version of the paper is available at `https://eccc.weizmann.ac.il/report/2018/092/`.

Acknowledgements We would like to thank Benjamin Caulfield, Andrea Lincoln, Luca Trevisan, Prashant Nalini Vasudevan, Ryan Williams, and Virginia Vassilevska Williams for helpful discussions and comments.

1 Introduction

Computational complexity can be viewed through two main perspectives: problem-centric or resource-centric. Problem-centric complexity theory asks what resources are required to solve *specific problems*, while resource-centric complexity deals with the relative power of different computational models given different resource budgets such as time, memory, non-determinism, randomness, etc. (see [17] for a discussion). Through complete problems, these two perspectives often coincide, so that a resource-centric view acts as a fine proxy for answering questions about the complexity of specific problems. The rapidly progressing field of fine-grained complexity theory, however, brings attention back to the problem-centric viewpoint, raising fine distinctions even between problems complete for the same complexity class, and making connections between problems at very different levels of complexity. To what extent are these two approaches linked, i.e., to what extent can inferences about the fine-grained complexities of specific problems be made from general assumptions about complexity classes, and vice versa?

Here, we examine such links between the fine-grained complexity of specific problems such as the k-Orthogonal Vectors and k-CLIQUE problems and general results about derandomization of algorithms. Derandomization has been a very fruitful study in complexity theory, with many fascinating connections between lower bounds, showing that problems require large amounts of resources to solve, and upper bounds, showing that classes of probabilistic algorithms can be 'derandomized' by simulating them deterministically in a non-trivial fashion. In particular, the hardness-to-randomness framework shows that in many cases, the existence of any "hard" problem can be used to derandomize classes of algorithms. We reconsider this framework from the fine-grained, problem-centric perspective. We show that replacing a generic hard problem with specific hardness conjectures from fine-grained complexity leads to quantitatively and qualitatively stronger derandomization results than one gets from the analogous assumption about a generic problem. In particular, we show that starting from these assumptions, we can simulate any polynomial-time probabilistic algorithm (on any samplable distribution on inputs with a very small fraction of errors) by a polynomial time probabilistc algorithm that uses only n^α random coins, for any $\alpha > 0$. This type of derandomization previously either assumed the existence of cryptographic One-Way Functions or *exponential non-uniform* hardness of Boolean functions.

Thus, the problem-centric conjectures of fine-grained complexity cannot live in isolation from classical resource-centric consequences about the power of randomness. Viewed another way, our results can be seen as a barrier to proving some of the key hardness assumptions used by fine-grained complexity theory. That is, despite recent progress towards proving hardness for k-Orthogonal Vectors, one of fine-grained complexity's key problems, in restricted models of computation [29], doing so for general randomized algorithms would immediately prove *all* problems in BPP are easy on average (over, say, uniformly chosen inputs).

1.1 Our Results

We obtain two main theorems about the power of BPP from uniform worst-case assumptions about well-studied problems from fine-grained complexity theory. We consider the k-Orthogonal Vectors (k-OV) and the k-CLIQUE problems, defined and motivated in Section 2.1, and show that (even weaker versions of) popular conjectures about their hardness give two flavors of average-case derandomization that improve over the classical *uniform* derandomizations.

All previous derandomizations from *uniform* assumptions on worst-case hardness only succeed on *infinitely many input lengths*. Our work is the first to use worst-case uniform assumptions to derandomize BPP *for all but finitely many input lengths*, giving a standard inclusion. The only other worst-case uniform assumptions known to imply such results are those so strong as to imply cryptographic assumptions or circuit lower bounds, fitting closer to the cryptographic or non-uniform derandomization literature. In contrast, our *uniform derandomizations* are from extremely weak worst-case uniform conjectures on simple, natural, combinatorial problems. Informally, we prove the following:

▶ **Informal Theorem 1.** *If k-OV or k-CLIQUE requires $n^{\epsilon k}$ time, for some constant $\epsilon > 1/2$, to **count** on randomized machines in the worst-case for all but finitely many input lengths, then BPP can be decided in polynomial time **using only** n^{α} **random bits** on average over **any** efficient input distribution, for any constant $\alpha > 0$.*

Randomness can be removed entirely by simply brute-forcing all random bits and taking the majority of the outputs to give the following more standard full derandomization.

▶ **Corollary.** *If k-OV or k-CLIQUE requires $n^{\epsilon k}$ time, for some constant $\epsilon > 1/2$, to **count** on randomized machines in the worst-case for all but finitely many input lengths, then BPP can be decided with **no randomness** in sub-exponential time on average over **any** efficient input distribution.*

This conclusion is strictly stronger than the classic uniform derandomizations of [25, 38]. The weakest uniform assumption previously known to imply such a conclusion was from those already strong enough to imply the cryptographic assumption of the existence of One-Way Functions that are hard to invert for polynomial time adversaries [9, 19, 20, 23, 42] or those implying *non-uniform* circuit lower bounds [4].

Our second main theorem, using techniques from [31], shows how to remove all randomness *within polynomial time* if the distribution over inputs is uniform. The only stronger derandomization from uniform assumptions were, again, from those already strong enough to imply circuit lower bounds or the cryptographic assumption of the existence of One-Way Permutations that require *exponential* time to invert [9, 20, 42].

▶ **Informal Theorem 2.** *If k-OV or k-CLIQUE require $n^{\epsilon k}$ time, for some constant $\epsilon > 1/2$, to **count** on randomized machines in the worst-case for all but finitely many input lengths, then BPP can be decided in polynomial time with **no randomness** on average over **the uniform distribution**.*

1.2 Related Work

Connections Between Problem-Centric and Resource-Centric Complexity. Most connections from problem-centric to resource-centric complexity show that faster algorithms for OV or related problems give circuit lower bounds. For instance, improvements in EDIT-DISTANCE algorithms imply circuit lower bounds [2] and solving OV faster (and thus

CNF-SAT [40]) implies circuit lower bounds [26]. These are all non-uniform results, however, whereas in this paper we are concerned with *machines* and their resource-bounds as opposed to circuits. On the uniform side, [18] recently showed that the exact complexity of k-Orthogonal Vectors is closely related to the complexity of uniform AC^0, although a connection between more powerful machine models and fine-grained assumptions was still not known until now. Further, most of these results follow from OV being *easy*. Our work shows instead that there are interesting resource-centric consequences if our fine-grained problems are *hard*.

Uniform Derandomization Framework. The uniform derandomization framework was introduced in [25], a breakthrough paper that showed the first derandomization from a uniform assumption ($\mathsf{EXP} \neq \mathsf{BPP}$) in the *low-end* setting: a weak assumption gives a *slow* (subexponential-time) deterministic simulation of BPP. This is in contrast to our simulation which retains small amounts of randomness but is *fast* (this is a strictly stronger result as it recovers the [25] derandomization as a corollary).

We build on [38], which simplifies the proof of [25] to prove that $\mathsf{PSPACE} \neq \mathsf{BPP}$ implies a non-trivial deterministic simulation of BPP. The technique of [38] carefully arithmetizes the PSPACE-complete problem TQBF and uses this as a hard function in the generator of [25]. Our proof substitutes a carefully-arithmetized k-OV problem from [8]. Numerous other works study derandomization from uniform assumptions ([27, 33, 24, 22, 37]), but these all focus on assumptions and consequences about *nondeterministic* classes.

All worst-case uniform derandomizations, including [38] and [25], seem to only be able to achieve simulations of BPP that succeed for infinitely many input lengths because of how their proofs use downward self-reductions. Our is the first work to achieve simulations on *all but finitely many input lengths*, because the k-OV and k-CLIQUE-inspired problems have very parallelizable downward self reductions so that we can reduce to a *single* much smaller input length rather than recurse through a chain of incrementally smaller input lengths in our downward self-reduction.

Heuristics by Extracting Randomness From the Input. A separate line of work began when [21] introduced the idea of using the *input itself* as a source of randomness to heuristically simulate randomized algorithms over uniformly-distributed inputs. While their assumptions contain oracles and are mostly non-uniform and average-case, they construct an algebraic problem inside P whose worst-case uniform hardness can be used in the framework of [25] to get an *infinitely-often* simulation of BPP in polynomial time. Our work differs in that we achieve an *almost-everywhere* simulation, that our assumptions are based on canonical fine-grained problems, and that our assumptions aren't against machines with SAT-oracles. Further, the downward self-reduction of their problem requires an expansion by minors of the determinant and so they cannot also obtain an *almost-everywhere* heuristic using our techniques without placing the determinant in NC^1 (as our modification to [25] exploits *embarrassingly parallel* downward self reductions).

The work of [31], generalizing [36], removes the SAT-oracles needed in the assumptions of [21] by showing that the Nisan-Wigderson generator (see [35]) remains secure against *non-uniform* adversaries even if the seed is revealed to potential distinguishers. In Section 3.2.2 we will show their arguments can be made *uniform* so we can derandomize from uniform assumptions. Seed-revealing Nisan-Wigderson generators are used in [31] to obtain polynomial-time heuristics for randomized algorithms, where the uniformly distributed input is used as a seed to the generator. The derandomizations in [31] are achieved from *non-uniform* assumptions of polynomial *average-case* hardness. From *worst-case uniform* assumptions we achieve the same derandomizations.

2 Preliminaries

All complexity measures of fine-grained problems will refer to time on a randomized word RAM with $O(\log(n))$-bit word length, as is standard for the fine-grained literature [41, 8]. Specifically, we will consider two-sided bounded error as in [8].

2.1 Fine-Grained Hardness

The problem-centric field of fine-grained complexity theory has had impressive success in showing the fixed polynomial time ("fine-grained") hardness of many practical problems by assuming the fine-grained hardness of four "key" well-studied problems, as discussed in [8]. We obtain our results under hardness conjectures about two of these four key problems: the k-Orthogonal Vectors (k-OV) problem and the k-CLIQUE problem.

***k*-CLIQUE.** Denote the matrix multiplication constant by ω. The fastest known algorithm for deciding if a graph has a k-CLIQUE (given its adjacency matrix) runs in time $O(n^{\omega k/3})$, and was discovered in 1985 [34] for k a multiple of three (for other k different ideas are needed [14]). It is conjectured that no algorithm can improve the exponent to a better constant. The parameterized version of the famous NP-hard MAX-CLIQUE problem [30], k-CLIQUE is one of the most heavily studied problems in theoretical computer science and is the canonical intractable (W[1]-complete) problem in parameterized complexity (see [1] for a review of the copious evidence of k-CLIQUE's hardness and consequences of its algorithm's exponent being improved). Recent work has shown that conjecturing k-CLIQUE to require $n^{\omega k/3-o(1)}$ time, for k a multiple of three, leads to interesting hardness results for other important problems such as parsing languages and RNA folding [1, 12, 5, 7], and it is known that refuting this conjecture deterministically would give a faster exact algorithm for MAX-CUT [40]. Our results hold under a *much weaker version of the conjecture*:

▶ **Definition 1** (Weak k-CLIQUE Conjecture)**.** There exists an absolute constant $\epsilon_0 > 1/2$ such that, for all $k \in \mathbb{N}$ a multiple of three, any randomized algorithm that *counts* the number of k-CLIQUE's in an n node graph requires $n^{\epsilon_0 k}$ time.

Note that this conjecture gives leeway for the exponent of the k-CLIQUE algorithm to be improved so long as it doesn't get down to $k/2$; even finding a linear time algorithm for Boolean matrix multipliaction ($\omega = 2$) would not contradict this conjecture! Further, even if it is possible to *decide* the k-CLIQUE problem that quickly, this conjecture still holds unless it is possible to *count all of the k-CLIQUE's in that time.*

***k*-Orthogonal Vectors.** Although the k-CLIQUE problem is certainly *at least* as important as the k-OV problem, for concreteness we will use the k-OV problem to demonstrate our techniques throughout the paper. Proofs based on hardness of k-CLIQUE follow identically.

▶ **Definition 2** (k-Orthogonal Vectors Problem, k-OV$_{n,d}$)**.** For an integer $k \geq 2$, the k-OV$_{n,d}$ problem on vectors of dimension d is to determine, given k sets $(U_1, \ldots, U_k)$ of n vectors from $\{0,1\}^d$ each, whether there exist $u_i \in U_i$ for each i such that over $\mathbb{Z}$,

$$\sum_{\ell \in [d]} u_{1\ell} \cdots u_{k\ell} = 0$$

If left unspecified, d is to be taken to be $d(n) = \lceil \log^2 n \rceil$.

▶ **Definition 3** (k-Orthogonal Vectors Conjecture, k-OVC)**.** For any $d = \omega(\log n)$, for all $k \geq 2$, any randomized algorithm for the k-$\mathsf{OV}_{n,d}$ problem requires $n^{k-o(1)}$ time.

For $k = 2$ the Orthogonal Vectors conjecture for deterministic algorithms has been extensively studied and is supported by the *Strong Exponential Time Hypothesis* (SETH) [40], which states that there is no $\epsilon > 0$ such that t-SAT can be solved in time $\widetilde{O}(2^{n(1-\epsilon)})$ for all values of t. The natural generalization to k-OV is studied in [8, 18] and its deterministic hardness is also supported by SETH. While SETH has been controversial , the deterministic k-OV conjecture seems to be a much weaker assumption and is independently believable and supported: it has been shown that it holds unless *all* first-order graph properties become easy to decide [18] and the 2-OV conjecture has recently been proven unconditionally when the model of computation is restricted to branching programs [29]. This conjecture has also been used to support the hardness of many practical and well-studied fine-grained problems [3, 6, 13]. As with k-CLIQUE, our main results will hold *using a much weaker version of the randomized k-OV conjecture* introduced below.

▶ **Definition 4** (Weak k-Orthogonal Vectors Conjecture)**.** For any $d = \omega(\log n)$, there exists an absolute constant $\epsilon_0 > 1/2$ such that, for all $k \geq 2$, any randomized algorithm *counting* the number of k-$\mathsf{OV}_{n,d}$ solutions requires $n^{\epsilon_0 k}$ time.

▶ Remark. For all of these conjectures we will also consider the strengthened versions that assume that all algorithms running in time less than what is required will fail *on all but finitely many input lengths*, as opposed to only on infinitely many input lengths. For natural problems we expect that hardness grows, instead of oscillates, asymptotically.

For the purposes of derandomization, for a given k, we will use a family of polynomials introduced in [8], $\left\{f^k_{n,d,p} : \mathbb{F}_p^{knd} \to \mathbb{F}_p\right\}_{n,d,p\in\mathbb{N}}$, such that the variables are grouped into sets of size nd in the form of a matrix $U_i \in \mathbb{F}_p^{n\times d}$ where the n rows $u_i \in U_i$ are each collections of d variables:

$$f^k_{n,d,p}(U_1, \ldots, U_k) = \sum_{u_1\in U_1,\ldots,u_k\in U_k} \prod_{\ell\in[d]} (1 - u_{1\ell} \cdots u_{k\ell})$$

The worst-case hardness of evaluating these polynomials was related to the worst-case hardness of k-OV_n in [8].

▶ **Lemma 5.** *Let p be the smallest prime number larger than n^k and $d = \lceil \log^2(n) \rceil$. If $f^k_{n,d,p}$ can be computed in $O(n^{k/2+c})$ time for some $c > 0$, then k-OV_n can be **counted** in time $\widetilde{O}(n^{k/2+c})$*

Derandomization from uniform assumptions typically requires two other properties of the assumed hard problem: random self-reducibility and downward self-reducibility. We recall from [8] that $f^k_{n,d,p}$ satisfies both of these properties. We give a polynomial for k-CLIQUE and show that it also has the necessary properties in the full version.

Random Self-Reducible. $f^k_{n,d,p}$ is *random self-reducible* by the following classical lemma [32, 15] (see [8] for a proof). Note that degree $\log^2 n$ adds negligibly to the random self-reduction time.

▶ **Lemma 6** (Random Self-Reducibility of Polynomials)**.** *If $f : \mathbb{F}_P^N \to \mathbb{F}_P$ is a degree $9 < D < P/12$ polynomial, then there exists a randomized algorithm that takes a circuit $\widehat{C}$ 3/4-approximating f and produces a circuit C exactly computing f, such that the algorithm succeeds with high probability and runs in time $poly(N, D, \log P, |\widehat{C}|)$.*

Downward Self-Reducible. We will show that $f^k_{n,d,p}$ is downward self-reducible in the sense that, if we we have a way to produce an oracle for $f^k_{\sqrt{n},d,p}$, we can quickly compute $f^k_{n,d,p}$ with it. Compare this to downward self reducibility going from input size n to $n-1$ in previous uniform derandomizations. We exploit our more dramatic shrinkage and parallelism to later give an almost-everywhere derandomization, instead of an infinitely-often one. The proof of the following lemma is in the full version.

▶ **Lemma 7.** *If there exists an algorithm A that, on input 1^n, outputs a circuit C computing $f^k_{\sqrt{n},d,p}$, then there exists an algorithm that computes $f^k_{n,d,p}$ in time $O(n^{k/2}|C| + \mathsf{TIME}(A))$.*

2.2 Derandomization

We now define pseudorandom generators (PRGs) in terms of their distinguishers.

▶ **Definition 8** (Distinguishers)**.** A test $T : \{0,1\}^{m^\ell} \to \{0,1\}$ is an ϵ-*distinguisher* against $G : \{0,1\}^m \to \{0,1\}^{m^\ell}$, denoted $T \in \mathsf{DIS}(G, \epsilon)$, if:

$$\left| \Pr_{r \sim \mathcal{U}_{m^\ell}} [T(r)] - \Pr_{z \sim \mathcal{U}_m} [T(G(z))] \right| > \epsilon$$

We also will consider the seemingly weaker object of distinguishers that succeed if they are also given the seed to the PRG. These were studied in [38] to relate uniform derandomization to average-case hardness and in [31] to derandomize over the uniform distribution by *using the random input itself as the seed* to the PRG.

▶ **Definition 9** (Seed-Aware Distinguishers)**.** A test $T : \{0,1\}^m \times \{0,1\}^{m^\ell} \to \{0,1\}$ is an ϵ-*seed-aware distinguisher* against G, denoted $T \in \mathsf{DIS}(G, \epsilon)$, if:

$$\left| \Pr_{x \sim \mathcal{U}_m, r \sim \mathcal{U}_{m^\ell}} [T(x, r)] - \Pr_{x \sim \mathcal{U}_m} [T(x, G(x))] \right| > \epsilon$$

Standard hardness-to-randomness arguments typically derandomize using generators that are based on some 'hard' function by contrapositive: if derandomization fails, then a distinguisher for the generator can be produced. Further, from a distinguisher, we can create a small circuit for the supposedly hard function that the generator was based on. For our purposes, we require an algorithmic version of this argument for derandomization from *uniform* hardness assumptions. More specifically, we will use the following lemma which was originally proved for distinguishers [38, 25] but Lemma 2.9 of [31] proves that it also holds for seed-aware distinguishers (while the proof of [31] is non-uniform, it is easy to see that it can be made constructive, in the same way that [25] gave a constructive version of [35]). Thus, $\mathsf{DIS}(G, \epsilon)$ in the lemma below can be thought to refer to either regular or seed-aware distinguishers (which justifies overloading this notation).

▶ **Lemma 10** (Algorithmic Distinguishers to Predictors ([38, 25]))**.** *For every random self-reducible f, there exists a function G with stretch m bits to m^ℓ bits and a constant c such that*

- *$G(z)$ is in time $(|z|^\ell)^c$ with oracle access to f on inputs of length at most $|z|$*
- *There exists a polynomial-time randomized algorithm A that, with high probability, given as input circuit $D \in \mathsf{DIS}(G, \epsilon)$ for ϵ at least inverse polynomial and an oracle for f, prints a circuit computing f exactly.*

2.3 Uniform Derandomization

Average-Case Tractability. We give standard definitions of average-case tractability (for an extensive survey of these notions, see [10]).

▶ **Definition 11** ($t(n)$-Samplable Ensemble)**.** An ensemble $\mu = \{\mu_n\}$ is $t(n)$-samplable if there is a randomized algorithm A that, on input a number n, outputs a string in $\{0,1\}^*$ and:

- A runs in time at most $t(n)$ on input n, regardless of its internal coin tosses
- for every n and for every $x \in \{0,1\}^*$, $\Pr[A(n) = x] = \mu_n(x)$

With this notion of samplable ensemble we can now consider *heuristic algorithms* that perform well on some language $\mathcal{L} : \{0,1\}^* \to \{0,1\}$ over some μ. The pair $(\mathcal{L}, \mu)$ is a *distributional problem.*

▶ **Definition 12** (Heuristics for Distributional Problems)**.** For $t : \mathbb{N} \to \mathbb{N}$, $\delta : \mathbb{N} \to \mathbb{R}^+$, we say $(\mathcal{L}, \mu) \in \text{Heur}_{\delta(n)}\mathsf{DTIME}[t(n)]$ if there is a time $t(n)$ deterministic algorithm A such that, for all but finitely many n: $\Pr_{x\sim\mu_n}[A(x) \neq \mathcal{L}(x)] \leq \delta(n)$.

For a class of languages $\mathcal{C}$ we say $(\mathcal{C}, \mu) \in \text{Heur}_{\delta(n)}\mathsf{DTIME}[t(n)]$ if $(\mathcal{L}, \mu) \in \text{Heur}_{\delta(n)}\mathsf{DTIME}[t(n)]$ for all $\mathcal{L} \in \mathcal{C}$. As in [10], $\text{Heur}_\delta\mathsf{P}$ is defined as the union over all polynomials p of $\text{Heur}_\delta\mathsf{DTIME}(p(n))$ and HeurP is the intersection over all inverse polynomial $\delta(n)$ of $\text{Heur}_\delta\mathsf{P}$. $\mathsf{HeurSUBEXP}$ is defined similarly where $\mathsf{SUBEXP} = \cap_{\epsilon>0}\mathsf{DTIME}\left[2^{n^\epsilon}\right]$. Finally, to discuss the *randomness-reduced* simulations we construct, we define BPTIME with a limited number of random coins in the natural way.

▶ **Definition 13** (Randomized Heuristics with Bounded Coins)**.** For $t : \mathbb{N} \to \mathbb{N}$, $\delta : \mathbb{N} \to \mathbb{R}^+$, and coin bound $r : \mathbb{N} \to \mathbb{N}$ we say $(\mathcal{L}, \mu) \in \text{Heur}_{\delta(n)}\mathsf{BPTIME}_{[r(n)]}[t(n)]$ if there is randomized algorithm A running in time $t(n)$ and flipping $r(n)$ coins such that, for all but finitely many n: $\Pr_{x\sim\mu_n}\left[\Pr_{r\sim\mathcal{U}_{r(n)}}[A(x,r) \neq \mathcal{L}(x)] > 1/3\right] \leq \delta(n)$

For example, $\text{Heur}\mathsf{BPP}_{[r(n)]}$ denotes the class of distributional problems that, for every inverse polynomial error, have a polynomial time randomized algorithm using only $r(n)$ random coins.

Infinitely-Often Simulation. As opposed to an algorithm that decides a language (possibly on average) "for all but finitely many n" as in Definition 12, an infinitely-often (io-) qualifier can be added to any complexity class to specify that an algorithm need only succeed on infinitely many input lengths within the time and error bounds. Thus, to derandomize BPP into $\mathsf{io\text{-}HeurP}$ over the uniform distribution is to say that every language in BPP can be simulated on average in polynomial time by an algorithm that is only guaranteed to succeed for infinitely many input lengths. *There is no guarantee on what those input lengths are or how large the gaps could be between them.* This is obviously a very undesirable notion of 'tractability'.

Non-uniform hardness to randomness trade-offs can derandomize almost-everywhere (the desired notion of tractability for asymptotics) by assuming almost-everywhere hardness: that no algorithm works *for all sufficiently large input lengths.* That is, the 'infinitely-often' qualifier on the consequent can be *flipped* across the implication to be an 'almost-everywhere' qualifier on the assumption and vice-versa. Thus, the unrealistic 'infinitely-often' notion of tractability can be dropped by slightly strengthening the assumption to the (as argued in Section 2.1's remark) realistic 'almost-everywhere' hardness. For *non-uniform* derandomizations this is possible.

Starting with [25] and the techniques it introduced, all *uniform* derandomizations have been infinitely-often derandomizations *without* being able to flip the io- qualifier to an 'almost-everywhere' assumption. Our work is the first that is able to do this in the uniform derandomization framework, thus removing the 'infinitely-often' qualifier from our derandomizations.

3 Fine-Grained Derandomization

We will prove our main derandomization results (Theorems 17 and 20) here. Under either the (weak) k-OV or k-CLIQUE conjectures, we derandomize BPP on average, where 'on average' will have two different flavors. Although all techniques apply to k-CLIQUE, for concreteness we will use k-OV throughout this section.

We show in Section 3.1 that if we base pseudorandom generators on $f^k_{n,d,p}$, then an algorithm printing distinguishers for this PRG can be used to count k-OV solutions quickly. We will then show in Section 3.2 how to attain these distinguisher-printing algorithms if derandomization *doesn't* work on average (for both flavors of on average). Thus, a failed derandomization using these PRGs refutes the k-OV conjecture (similarly for k-CLIQUE).

3.1 Counting k-OV from Distinguishers

In this section we show that any algorithm producing a distinguisher for $G^{f^k_{m,d,p}}$ (the generator guaranteed to exist from Lemma 14, using the hard function $f^k_{m,d,p}$) can be used to quickly count k-OV solutions. First, Lemma 14 follows immediately by combining the distinguisher to predictor algorithm of Lemma 10 with the fact that $f^k_{m,d,p}$ is random self-reducible as in Lemma 6.

▶ **Lemma 14.** *There is a randomized algorithm $A^{f^k_{m,d,p}}$ that takes any circuit D that is a distinguisher for $G^{f^k_{m,d,p}}$ and produces a circuit C exactly computing $f^k_{m,d,p}$, such that A succeeds with high probability and runs in time* $\mathsf{poly}(m, d, \log p, |D|)$

As $f^k_{m,d,p}$ is efficiently computable (unlike the hard problems of [25]) in time $O(m^k \mathsf{poly}(d, \log p))$, we get the following theorem *without* an oracle by running the algorithm guaranteed in Lemma 14 with each oracle call answered by naïve brute force computation of $f^k_{m,d,p}$.

▶ **Lemma 15.** *There is a randomized algorithm B that takes any circuit D that is a distinguisher for $G^{f^k_{m,d,p}}$ and produces a circuit C of size* $\mathsf{poly}(m, d, \log p, |D|)$ *exactly computing $f^k_{m,d,p}$. B succeeds with high probability and runs in time $O(m^k \mathsf{poly}(m, d, \log p, |D|))$.*

Now we show that, if we have an algorithm producing a distinguisher, then we have an algorithm counting k-OV.

▶ **Theorem 16.** *Let p be the smallest prime number larger than n^k and $d = \lceil \log^2(n) \rceil$. If there is an algorithm A that, on input 1^n, outputs a distinguisher D of* $\mathsf{poly}(n)$ *size for $G^{f^k_{\sqrt{n},d,p}}$, then there exists a randomized algorithm counting k-OV_n that runs in time $O(n^{k/2+c} + \mathsf{TIME}(A))$, where c only depends on $|D|$.*

Proof. Using A, we print a distinguisher circuit D for $G^{f^k_{\sqrt{n},d,p}}$. Then, by Lemma 15, we know there exists a randomized algorithm running in time $O(n^{k/2}\mathsf{poly}(\sqrt{n}, d, \log p, |D|)) = O(n^{k/2+c_1})$ that yields a circuit exactly computing $f^k_{\sqrt{n},d,p}$ of size only $\mathsf{poly}(\sqrt{n}, d, \log p, |D|) = O(n^{c_2})$, where c_1 and c_2 only depend on $|D|$. Thus, by Lemma 7, there exists an algorithm

computing $f^k_{n,d,p}$ in time $O(n^{k/2+c_2} + (n^{k/2+c_1} + \mathsf{TIME}(A))) = O(n^{k/2+c} + \mathsf{TIME}(A))$ for $c = \max\{c_1, c_2\}$. Finally, this gives us an algorithm running in time $\widetilde{O}(n^{k/2+c} + \mathsf{TIME}(A))$ to count k-OV_n by Lemma 5. ◀

3.2 Printing Distinguishers from Failed Derandomization

3.2.1 Randomness-Reduced Heuristics Over Any Efficient Distribution

Our first main result in derandomizing BPP is to reduce the amount of randomness required to arbitrarily small quantities, over any efficient distribution of inputs. This simulation trades time for reduced randomness under fine-grained hardness assumptions.

▶ **Theorem 17.** *If the weak k-OV conjecture holds almost everywhere, then, for all polynomially samplable ensembles μ and for all constants $\alpha > 0$, $(\mathsf{BPP}, \mu) \in \mathrm{Heur}\mathsf{BPP}_{[n^\alpha]}$.*

Thus, for any efficient distribution over inputs that nature might be drawing from and for any inverse polynomial error rate we specify, we can simulate BPP using only n^α random bits for any constant $\alpha > 0$ we want. In contrast to typical full derandomizations which brute-force all seeds to a pseudorandom generator and take majority answer (which we can also do with our randomness-reduced derandomization to get a subexponential-time full derandomization), we show that choosing a *single random seed* and using the generator's output as our randomness yields randomness-reduced simulations so long as the generator is efficient enough (which it typically is not; 'quick' complexity-theoretic PRGs are usually given exponential time in their seed length).

▶ **Definition 18** (Randomness-Reduced Simulations)**.** Let $A : \{0,1\}^N \times \{0,1\}^{N^\ell} \to \{0,1\}$ be a randomized algorithm that uses N^ℓ random bits and let $G : \{0,1\}^{N^\alpha} \to \{0,1\}^{N^\ell}$ be a function. Then for constant $\alpha > 0$, define the randomness-reduced simulation to be a randomized algorithm $B : \{0,1\}^N \times \{0,1\}^{N^\alpha} \to \{0,1\}$ using only N^α random bits as $B(x,r) = A(x, G(r))$.

Lemma 19 states that if this simulation fails, we can uniformly print a distinguisher for the function G. This proof is identical to that of Lemma 18 in [25] and is recalled in the full version.

▶ **Lemma 19** (Failed Randomness-Reduction to Distinguishers)**.** *Let A, B, and G be as in Definition 18 such that for language $\mathcal{L} : \{0,1\}^N \to \{0,1\}$, $\Pr_{r \sim \mathcal{U}_{N^\ell}}[A(x,r) \neq \mathcal{L}(x)] \leq 1/10$. That is, that A as a good randomized algorithm deciding $\mathcal{L}$ for all $x \in \{0,1\}^N$. Yet, also assume that, for μ samplable in time N^{a_1} and $\delta(n) = 1/N^{a_2}$, it holds that*

$$\Pr_{x \sim \mu_N}\left[\Pr_{r \sim \mathcal{U}_{N^\alpha}}[B(x,r) \neq \mathcal{L}(x)] > 1/3\right] \geq \delta(N)$$

So B is a (randomness-reduced) randomized algorithm that does not decide $\mathcal{L}$ on average over μ. Then $1^N \mapsto \mathsf{DIS}(G, 1/5)$ is in randomized time N^c $\mathsf{TIME}(G)$ for c depending on a_1 and a_2.

Randomness-Reduced Simulation from k-OV. To finish defining a randomness-reduced simulation, we need to use a specific pseudorandom generator G that, for input length N, stretches N^α coins to N^ℓ. Thus, consider the family of simulations B_k using the standard generators $G^{f^k_{\sqrt{n},d,p}}$ of Lemma 10 that map $\sqrt{n}^s$ bits to $\sqrt{n}^b$ bits, for some fixed s and any b we choose, using $f^k_{\sqrt{n},d,p}$ as our hard function, for $d = \log^2 n$ and p the smallest

prime number larger than n^k. Set $b = s\ell/\alpha$ and $\sqrt{n} = N^{\alpha/s}$. Note that $\mathsf{TIME}(G^{f^k_{\sqrt{n},d,p}}) = \mathsf{poly}(N)\ n^{k/2} = \mathsf{poly}(N)$ by naïvely evaluating $f^k_{\sqrt{n},d,p}$ at each oracle call, giving an efficient randomness-reduced simulation. Further, since $N = \mathsf{poly}(n)$, $\mathsf{TIME}(G^{f^k_{\sqrt{n},d,p}})$ also equals $n^{k/2+c}$ for some constant c not depending on k (this will be useful in quickly counting k-OV_n using downward self-reducibility in the following proof). Thus, given an N^ℓ-coin machine A, we have the N^α-coin machine $B_k(x,r) = A\left(x, G^{f^k_{\sqrt{n},d,p}}(r)\right)$. We now prove our main Theorem 17 using this simulation and the above lemma.

Proof of Theorem 17. We proceed by contradiction. Assume that the weak k-OV_n conjecture holds for all but finitely many input lengths, where $\epsilon_0 = 1/2 + \gamma$ for some constant $\gamma > 0$, but that there exists $\mathcal{L} \in \mathsf{BPP}$, a polynomially samplable distribution μ, constant α, and an inverse polynomial function $\delta(N)$ such that *any* polynomial-time randomness-reduced algorithm with coin bound N^α fails in deciding $\mathcal{L}$ on average over μ within $\delta(N)$ error for infinitely many input lengths N.

Since $\mathcal{L} \in \mathsf{BPP}$ there is a randomized algorithm A deciding $\mathcal{L}$ with probability of error at most $1/10$ over its randomness yet, since *any* polynomial-time randomness-reduced algorithm fails to decide $\mathcal{L}$ on average, B_k, the randomness-reduced simulation of A described above, fails on average infinitely often, for *any* constant k. Thus, the antecedents of Lemma 19 are satisfied and we can uniformly print $D \in \mathsf{DIS}(G^{f^k_{\sqrt{n},d,p}}, 1/5)$ in time $n^{c_1}\ \mathsf{TIME}\left(G^{f^k_{\sqrt{n},d,p}}\right) = n^{c_1}\ n^{c_2} n^{k/2}$.

This uniform printing of D allows us to apply Theorem 16 to count k-OV_n in time $O(n^{k/2+c_3} + n^{k/2+c_1+c_2}) = O(n^{k/2+c}) = O(n^{(\frac{1}{2}+\frac{c}{k})k})$ for *any* k, where $c = \max\{c_1 + c_2, c_3\}$. Setting k such that $\frac{c}{k} < \gamma$ yields our contradictions: on the infinitely many input lengths where B_k fails to derandomize $\mathcal{L}$, the algorithm counts k-OV faster than $n^{\epsilon_0 k}$ time. ◀

3.2.2 Fast Heuristics for BPP Over the Uniform Distribution

Here we present our second flavor of derandomization: a fully deterministic heuristic for BPP when inputs are sampled according to the uniform distribution.

▶ **Theorem 20.** *If the weak k-OV conjecture holds almost everywhere, then $(\mathsf{BPP}, \mathcal{U}) \in \mathrm{HeurP}$.*

This strictly improves previous uniform derandomizations over the uniform distribution. Specifically, [21] can be seen to achieve our derandomization identically from a worst-case uniform assumption if combined with techniques from [31] *except only on infinitely many input lengths.*

We proceed by showing that if a PRG fails to give a good heuristic for BPP over the uniform distribution, a seed-aware distinguisher for the PRG can be produced uniformly and efficiently, which can then be used to count k-OV solutions quickly using Theorem 16.

▶ **Definition 21** (Input-As-Seed Heuristics)**.** Let $A : \{0,1\}^N \times \{0,1\}^{N^\ell} \to \{0,1\}$ be a polynomial-time randomized algorithm using N^ℓ random bits. Let $G : \{0,1\}^N \to \{0,1\}^{N^\ell}$ be a deterministic function. Define the heuristic $B : \{0,1\}^N \to \{0,1\}$ that uses its input as G's seed as $B(x) = A(x, G(x))$.

We prove a uniform analog of the Main Lemma of [31], which gave the consequences of failed heuristics in the *non-uniform* setting. Namely, we prove:

▶ **Lemma 22** (Failed Heuristics to Distinguishers). *Let* $A : \{0,1\}^N \times \{0,1\}^{N^\ell} \to \{0,1\}$ *and* $\mathcal{L} : \{0,1\}^N \to \{0,1\}$ *be functions such that* $\Pr_{x\sim\mathcal{U}_N, r\sim\mathcal{U}_{N^\ell}}[A(x,r) \neq \mathcal{L}(x)] \leq \rho$. *Let* B *be the input-as-seed heuristic for* A *using function* G. *Then, if* B *does* ***not*** *succeed on a* $(5\rho+\epsilon)$ *fraction of the inputs of a given length, the map* $1^N \mapsto \mathsf{DIS}(G,\epsilon)$ *is uniform and in randomized polynomial time, for infinitely many* N.

We sketch the proof here (see the full version for a proof). If B is a *bad* heuristic for $\mathcal{L}$, then we could use $B(x) = A(x, G(x))$ as a seed-aware distinguisher for G by comparing $B(x)$ to $\mathcal{L}(x)$. Unfortunately we cannot afford to print distinguishers with $\mathcal{L}$-oracles. But since we are guaranteed that A is a *good* heuristic for $\mathcal{L}$, we can obtain a deterministic circuit close to $\mathcal{L}$ from A, by fixing a string of good random bits r'. The proof of the analogous lemma in [31] uses non-uniformity to obtain a good r' for distinguishing, but we can instead obtain good strings r' by showing that there are *many* good random strings. We find a good r' by a sample-and-test procedure. If we compare $B(x)$ and the fixed-coin algorithm $A(x,r')$, they will also tend to disagree, giving the necessary distinguishing gap.

Fully Deterministic Heuristics from k-OV. Here we specify a family of heuristics B_k, by specifying the generator G, that stretches a seed of length N to N^ℓ, as the generators $G^{f^k_{\sqrt{n},d,p}}$ of Lemma 10. These map $\sqrt{n}^s$ bits to $\sqrt{n}^b$ bits, for some fixed s and any b we choose, using $f^k_{\sqrt{n},d,p}$, for $d = \log^2 n$ and p the smallest prime number larger than n^k. Set $b = s\ell$ and $\sqrt{n} = N^{1/s}$. All comments about the runtime of the randomness-reduced heuristic in Section 3.2.1 also apply to this fully deterministic heuristic. Thus, given an N^ℓ-coin machine A, we have the *deterministic* machine $B_k(x) = A\left(x, G^{f^k_{\sqrt{n},d,p}}(x)\right)$.

This can now be used to prove Theorem 20, although we defer this proof to the full version as it is very similar to the proof of Theorem 17 in Section 3.2.1.

4 Open Questions

- We derandomize under hardness conjectures about two of four 'key' problems in fine-grained complexity: k-OV and k-CLIQUE. What about k-SUM and APSP? APSP doesn't seem to have a natural hierarchy and so doesn't fit our framework (although it does reduce to ZERO-TRIANGLE which generalizes to ZERO-k-CLIQUE and should easily work in our framework using polynomials similar to those in [8]). k-SUM however is actually computable in $O(n^{\lceil k/2\rceil})$ time and so our downward self-reducibility techniques are not fast enough to break this conjecture in the contrapositive. The clearest path we see to getting derandomization without reintroducing the io- qualifier is to find a polynomial for k-SUM that is also computable in $\widetilde{O}(n^{\lceil k/2\rceil})$ time (unlike the one found in [8]).
- Our derandomizations hold under (randomized) SETH, since SETH implies the k-OV conjecture. Can a better derandomization be obtained directly from SETH, the stronger assumption? A stumbling block here is the random self-reduction, an ingredient in all known uniform derandomization techniques: If t-SAT has a straightforward and efficient random-self-reduction, PH collapses [16, 11]. So derandomizing from SETH directly could require new ideas, or a strange random self-reduction. An *inefficient* random self-reduction for t-SAT shouldn't collapse PH except to say that t-SAT has a mildly exponential MA proof which is already known to be true [39], although most random self-reductions we know are through arithmetization which seems to always have 'low' degree to the point that such a polynomial would still collapse PH.
- Is a strong "derandomization to hardness" converse possible for these heuristic simulations of BPP? In the full version of this paper, we show a weak converse: our simulation

is impossible without separting DTIME[$n^{\omega(1)}$] from BPP. But this is a very different statement from the k-OV or k-CLIQUE conjectures. In [31], they show that herusitic simulations of BPP with inverse-subexponential error rates imply circuit lower bounds, by generalizing techniques of [28]. Do the efficient inverse-polynomial error heuristics we obtain imply any circuit lower bounds?

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 98–117. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.16.

2 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends or: A polylog shaved is a lower bound made. *CoRR*, abs/1511.06022, 2015. URL: http://arxiv.org/abs/1511.06022.

3 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2014. doi:10.1007/978-3-662-43948-7_4.

4 László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993. doi:10.1007/BF01275486.

5 Arturs Backurs, Nishanth Dikkala, and Christos Tzamos. Tight hardness results for maximum weight rectangles. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 81:1–81:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.81.

6 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58. ACM, 2015. doi:10.1145/2746539.2746612.

7 Arturs Backurs and Christos Tzamos. Improving viterbi is hard: Better runtimes imply faster clique algorithms. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 311–321. PMLR, 2017. URL: http://proceedings.mlr.press/v70/backurs17a.html.

8 Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Average-case fine-grained hardness. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 483–496. ACM, 2017. doi:10.1145/3055399.3055466.

9 Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984. doi:10.1137/0213053.

10 Andrej Bogdanov and Luca Trevisan. Average-case complexity. *Foundations and Trends in Theoretical Computer Science*, 2(1), 2006. doi:10.1561/0400000004.

11 Andrej Bogdanov and Luca Trevisan. On worst-case to average-case reductions for NP problems. *SIAM J. Comput.*, 36(4):1119–1159, 2006. doi:10.1137/S0097539705446974.

12 Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 307–318. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.36.

13 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.15.

14 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1-3):57–67, 2004. doi:10.1016/j.tcs.2004.05.009.

15 Joan Feigenbaum and Lance Fortnow. On the random-self-reducibility of complete sets. In *Proceedings of the Sixth Annual Structure in Complexity Theory Conference, Chicago, Illinois, USA, June 30 - July 3, 1991*, pages 124–132. IEEE Computer Society, 1991. doi:10.1109/SCT.1991.160252.

16 Joan Feigenbaum and Lance Fortnow. Random-self-reducibility of complete sets. *SIAM J. Comput.*, 22(5):994–1005, 1993. doi:10.1137/0222061.

17 Jiawei Gao and Russell Impagliazzo. Orthogonal vectors is hard for first-order properties on sparse graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:53, 2016. URL: http://eccc.hpi-web.de/report/2016/053.

18 Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and R. Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2162–2181. SIAM, 2017. doi:10.1137/1.9781611974782.141.

19 Oded Goldreich, Hugo Krawczyk, and Michael Luby. On the existence of pseudorandom generators. *SIAM J. Comput.*, 22(6):1163–1175, 1993. doi:10.1137/0222069.

20 Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 25–32, 1989. doi:10.1145/73007.73010.

21 Oded Goldreich and Avi Wigderson. Derandomization that is rarely wrong from short advice that is typically good. In José D. P. Rolim and Salil P. Vadhan, editors, *Randomization and Approximation Techniques, 6th International Workshop, RANDOM 2002, Cambridge, MA, USA, September 13-15, 2002, Proceedings*, volume 2483 of *Lecture Notes in Computer Science*, pages 209–223. Springer, 2002. doi:10.1007/3-540-45726-7_17.

22 Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma. Uniform hardness vs. randomness tradeoffs for arthur-merlin games. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 33–47. IEEE Computer Society, 2003. doi:10.1109/CCC.2003.1214408.

23 Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999. doi:10.1137/S0097539793244708.

24 Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: exponential time vs. probabilistic polynomial time. *J. Comput. Syst. Sci.*, 65(4):672–694, 2002. doi:10.1016/S0022-0000(02)00024-7.

25 Russell Impagliazzo and Avi Wigderson. Randomness vs time: Derandomization under a uniform assumption. *J. Comput. Syst. Sci.*, 63(4):672–688, 2001. doi:10.1006/jcss.2001.1780.

26 Hamid Jahanjou, Eric Miles, and Emanuele Viola. Local reductions. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 749–760. Springer, 2015. doi:10.1007/978-3-662-47672-7_61.

27 Valentine Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. *J. Comput. Syst. Sci.*, 63(2):236–252, 2001. doi:10.1006/jcss.2001.1763.

28 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1-2):1–46, 2004. doi:10.1007/s00037-004-0182-6.

29 Daniel M. Kane and R. Ryan Williams. The orthogonal vectors conjecture for branching programs and formulas. *CoRR*, abs/1709.05294, 2017. URL: http://arxiv.org/abs/1709.05294.

30 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. URL: http://www.cs.berkeley.edu/~luca/cs172/karp.pdf.

31 Jeff Kinne, Dieter van Melkebeek, and Ronen Shaltiel. Pseudorandom generators, typically-correct derandomization, and circuit lower bounds. *Computational Complexity*, 21(1):3–61, 2012. doi:10.1007/s00037-011-0019-z.

32 Richard J. Lipton. New directions in testing. In Joan Feigenbaum and Michael Merritt, editors, *Distributed Computing And Cryptography, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, October 4-6, 1989*, volume 2 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 191–202. DIMACS/AMS, 1989.

33 Chi-Jen Lu. Derandomizing arthur-merlin games under uniform assumptions. *Computational Complexity*, 10(3):247–259, 2001. doi:10.1007/s00037-001-8196-9.

34 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

35 Noam Nisan and Avi Wigderson. Hardness vs randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994. doi:10.1016/S0022-0000(05)80043-1.

36 Ronen Shaltiel. Weak derandomization of weak algorithms: Explicit versions of yao's lemma. *Computational Complexity*, 20(1):87–143, 2011. doi:10.1007/s00037-011-0006-4.

37 Ronen Shaltiel and Christopher Umans. Low-end uniform hardness versus randomness tradeoffs for AM. *SIAM J. Comput.*, 39(3):1006–1037, 2009. doi:10.1137/070698348.

38 Luca Trevisan and Salil P. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. *Computational Complexity*, 16(4):331–364, 2007. doi:10.1007/s00037-007-0233-x.

39 Richard Ryan Williams. Strong ETH breaks with merlin and arthur: Short non-interactive proofs of batch evaluation. In Ran Raz, editor, *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, volume 50 of *LIPIcs*, pages 2:1–2:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CCC.2016.2.

40 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

41 Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In Thore Husfeldt and Iyad A. Kanj, editors, *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, volume 43 of *LIPIcs*, pages 17–29. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.IPEC.2015.17.

42 Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 80–91. IEEE Computer Society, 1982. doi:10.1109/SFCS.1982.45.

Ranking with Fairness Constraints

L. Elisa Celis
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Damian Straszak
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Nisheeth K. Vishnoi
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Abstract

Ranking algorithms are deployed widely to order a set of items in applications such as search engines, news feeds, and recommendation systems. Recent studies, however, have shown that, left unchecked, the output of ranking algorithms can result in decreased diversity in the type of content presented, promote stereotypes, and polarize opinions. In order to address such issues, we study the following variant of the traditional ranking problem when, in addition, there are *fairness* or *diversity* constraints. Given a collection of items along with 1) the value of placing an item in a particular position in the ranking, 2) the collection of *sensitive* attributes (such as gender, race, political opinion) of each item and 3) a collection of fairness constraints that, for each k, bound the number of items with each attribute that are allowed to appear in the top k positions of the ranking, the goal is to output a ranking that maximizes the value with respect to the original rank quality metric while respecting the constraints. This problem encapsulates various well-studied problems related to bipartite and hypergraph matching as special cases and turns out to be hard to approximate even with simple constraints. Our main technical contributions are fast exact and approximation algorithms along with complementary hardness results that, together, come close to settling the approximability of this constrained ranking maximization problem. Unlike prior work on the approximability of constrained matching problems, our algorithm runs in linear time, even when the number of constraints is (polynomially) large, its approximation ratio does not depend on the number of constraints, and it produces solutions with small constraint violations. Our results rely on insights about the constrained matching problem when the objective function satisfies certain properties that appear in common ranking metrics such as discounted cumulative gain (DCG), Spearman's rho or Bradley-Terry, along with the nested structure of fairness constraints.

2012 ACM Subject Classification Information systems → Retrieval models and ranking, Theory of computation → Approximation algorithms analysis, Theory of computation → Discrete optimization

Keywords and phrases Ranking, Fairness, Optimization, Matching, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.28

Related Version A full version of this paper is available at [13], `https://arxiv.org/abs/1704.06840`.

1 Introduction

Selecting and ranking a subset of data is a fundamental problem in information retrieval and at the core of ubiquitous applications including ordering search results such (e.g., Google), personalized social media feeds (e.g., Facebook, Twitter or Instagram), ecommerce websites (e.g., Amazon or eBay), and online media sites (e.g., Netflix or YouTube). The basic

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 28; pp. 28:1–28:15


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

algorithmic problem that arises is as follows: There are m *items* (e.g., webpages, images, or documents), and the goal is to output a list of $n \ll m$ items in the order that is most *valuable* to a given user or company. For each item $i \in [m]$ and a position $j \in [n]$ one is given a number W_{ij} that captures the *value* that item i contributes to the ranking if placed at position j. These values can be tailored to a particular query or user and a significant effort has gone into developing models and mechanisms to learn these parameters [31]. In practice there are many ways one could arrive at W_{ij}, each of which results in a slightly different metric for the value of a ranking – prevalent examples include versions of discounted cumulative gain (DCG) [26], Bradley-Terry [6] and Spearman's rho [41]. Note that for many of these metrics, one does not necessarily need nm parameters to specify W and typically m "degrees of freedom" is enough (just specifying the "quality of each item"). Still, we choose to work with this general setting, and only abstract out the most important properties such a weight matrix W satisfies. Generally, for such metrics, W_{ij} is non-increasing in both i and j, and if we interpret $i_1 < i_2$ to mean that i_1 has better *quality* than i_2, then the value of the ranking can only increase by placing i_1 above i_2 in the ranking. Formally, such values satisfy the following property (known as monotonicity and the Monge condition)

$$W_{i_1 j_1} \geq W_{i_2 j_1} \quad \text{and} \quad W_{i_1 j_1} \geq W_{i_1 j_2} \quad \text{and} \quad W_{i_1 j_1} + W_{i_2 j_2} \geq W_{i_1 j_2} + W_{j_1 i_2} \tag{1}$$

for all $1 \leq i_1 < i_2 \leq m$ and $1 \leq j_1 < j_2 \leq n$. The *ranking maximization* problem is to find an assignment of the items to each of the n positions such that the total value obtained is maximized. In this form, the problem is equivalent to finding the maximum weight matching in a complete $m \times n$ bipartite graph and has a well known solution – the Hungarian algorithm.

However, recent studies have shown that producing rankings in this manner can result in one type of content being overrepresented at the expense of another. This is a form of *algorithmic bias* and can lead to grave societal consequences – from search results that inadvertently promote stereotypes by over/under-representing sensitive attributes such as race and gender [28, 5], to news feeds that can promote extremist ideology [18] and possibly even influence the results of elections [2, 3]. For example, [21] demonstrated that by varying the ranking of a set of news articles the voting preferences of undecided voters can be manipulated. Towards ensuring that no type of content is overrepresented in the context of the ranking problem as defined above, we introduce the *constrained ranking maximization* problem that restricts allowable rankings to those in which no type of content dominates – i.e., *to ensure the rankings are fair*.

Since fairness (and bias) could mean different things in different contexts, rather than fixing one specific notion of fairness, we allow the *user* to specify a set of *fairness constraints*; in other words, we take the constraints as input. As a motivating example, consider the setting in which the set of items consists of m images of computer scientists, each image is associated with several (possibly non-disjoint) sensitive attributes or *properties* such as gender, ethnicity and age, and a subset of size n needs to be selected and ranked. The user can specify an upper-bound $U_{k\ell} \in \mathbb{Z}_{\geq 0}$ on the number of items with property ℓ that are allowed to appear in the top k positions of the ranking, and similarly a lower-bound $L_{k\ell}$. Formally, let $\{1, 2, \ldots, p\}$ be a set of properties and let $P_\ell \subseteq [m]$ be the set of items that have the property ℓ (note that these sets need not be disjoint). Let x be an $m \times n$ binary assignment matrix whose j-th column contains a one in the i-th position if item i is assigned to position j (each position must be assigned to exactly one item and each item can be assigned to at most one position). We say that x satisfies the fairness constraints if for all

$\ell \in [p]$ and $k \in [n]$, we have

$$L_{k\ell} \leq \sum_{1 \leq j \leq k} \sum_{i \in P_\ell} x_{ij} \leq U_{k\ell},$$

If we let $\mathcal{B}$ be the family of all assignment matrices x that satisfy the fairness constraints, the constrained ranking optimization problem is: Given the sets of items with each property $\{P_1, \ldots, P_p\}$, the fairness constraints, $\{L_{k\ell}\}$, $\{U_{k\ell}\}$, and the values $\{W_{ij}\}$, find

$$\arg\max_{x \in \mathcal{B}} \sum_{i \in [m], j \in [n]} W_{ij} x_{ij}.$$

This problem is equivalent to finding a maximum weight matching of size n *that satisfies the given fairness constraints* in a weighted complete $m \times n$ bipartite graph, and now becomes non-trivial – its complexity is the central object of study in this paper.

Beyond the fairness and ethical considerations, traditional *diversification* concerns in information retrieval such as query ambiguity (does "jaguar" refer to the car or the animal?) or user context (does the user want to see webpages, news articles, academic papers or images?) can also be cast in this framework. Towards this, a rich literature on diversifying rankings has emerged in information retrieval. On a high-level, several approaches redefine the objective function to incorporate a notion of diversity and leave the ranking maximization problem unconstrained. E.g., a common approach is to re-weight the w_{ij}s to attempt to capture the amount of diversity item i would introduce at position k conditioned on the items that were placed at positions $1, \ldots, k-1$ (see [7, 47, 17, 46, 48]), or casting it directly as an (unconstrained) multi-objective optimization problem [43]. Alternate approaches mix together or aggregate different rankings, e.g., as generated by different interpretations of a query [36, 20]. Diversity has also been found to be desirable by users [14], and has been observed to arise inherently when the ranking is determined by user upvotes [12]. Despite these efforts and the fact that all major search engines now diversify their results, highly uniform content is often still displayed – e.g., certain image searches can display results that have almost entirely the same attributes [28]. Further, [23] showed that no single diversification function can satisfy a set of natural axioms that one would want any fair ranking to have. In essence, there is a tension between relevance and fairness – if the w_{ij}s for items that have a given property are much higher than the rest, the above approaches cannot correct for overrepresentation. Hence the reason to cast the problem as a constrained optimization problem: The objective is still determined by the values but the solution space is restricted by fairness constraints.

Theoretically, the fairness constraints come with a computational price: The constrained ranking maximization problem can be seen to generalize various **NP**-hard problems such as independent set, hypergraph matching and set packing. Unlike the unconstrained case, even checking if there is a complete feasible ranking (i.e., $\mathcal{B} \neq \emptyset$) is **NP**-hard. As a consequence, in general, we cannot hope to produce a solution that does not violate any constraints. Some variants and generalizations of our problem have been studied in the TCS and optimization literature; here we mention the three most relevant. Note that some may leave empty positions in the ranking as opposed to selecting n elements to rank as we desire. [1] considered the bipartite perfect matching problem with poly(m) constraints. They present a polynomial time randomized algorithm that finds a near-perfect matching which violates each constraint additively by at most $O(\sqrt{m})$. [24] improved the above result to a $(1+\varepsilon)$-approximation algorithm; however, the running time of their algorithm is roughly $m^{K^{2.5}/\varepsilon^2}$ where K is the number of hard constraints and the output is a matching. [42] studied the approximability of

the packing integer program problem which, when applied to our setting and gives an $O(\sqrt{m})$ approximation algorithm. In the constrained ranking maximization problem presented above, all of these results seem inadequate; the number of fairness constraints is $2np$ which would make the running time of [24] too large and an additive violation of $O(\sqrt{m})$ would render the upper-bound constraints impotent.

The main technical contributions of this paper are fast, exact and approximation algorithms for this constrained ranking maximization problem along with complementary hardness results which, together, give a solid understanding of the computational complexity of this problem. To overcome the limitations of the past work on constrained matching problems, our results often make use of two structural properties of such a formulation: A) The set of constraints can be broken into p groups; for each property $\ell \in [p]$ we have n (nested) upper-bound constraints, one for each $k \in [n]$, and B) The objective function satisfies the property stated in (1). Using properties A) and B) we obtain efficient – polynomial, or even linear time algorithms for this problem in various interesting regimes. Both these properties are natural in the information retrieval setting and could be useful in other algorithmic contexts involving rankings.

2 Our model

We study the following *constrained ranking maximization problem*

$$\arg\max_{x \in R_{m,n}} \sum_{i \in [m], j \in [n]} W_{ij} x_{ij} \qquad \text{s.t.} \quad L_{k\ell} \leq \sum_{1 \leq j \leq k} \sum_{i \in P_\ell} x_{ij} \leq U_{k\ell} \qquad \forall\, \ell \in [p], k \in [n], \tag{2}$$

where $R_{m,n}$ is the set of all matrices $\{0,1\}^{m \times n}$ which represent ranking m items into n positions. Recall that W_{ij} represents the profit of placing item i at position j and for every property $\ell \in [p]$ and every position k in the ranking, $L_{k\ell}$ and $U_{k\ell}$ are the lower and upper bound on the number of items having property ℓ that are allowed to be in the top k positions in the ranking. For an example, we refer to Figure 1.

We distinguish two important special cases of the problem: when only the upper-bound constraints are present, and when only the lower-bound constraints are present. These variants are referred to as the *constrained ranking maximization problem (U)* and the *constrained ranking maximization problem (L)* respectively, and to avoid confusion we sometimes add (LU) when talking about the general problem with both types of constraints present. Furthermore, most our results hold under the assumption that the weight function W is monotone and satisfies the Monge property (1), whenever these assumptions are not necessary, we emphasize this fact by saying that *general weights* are allowed.

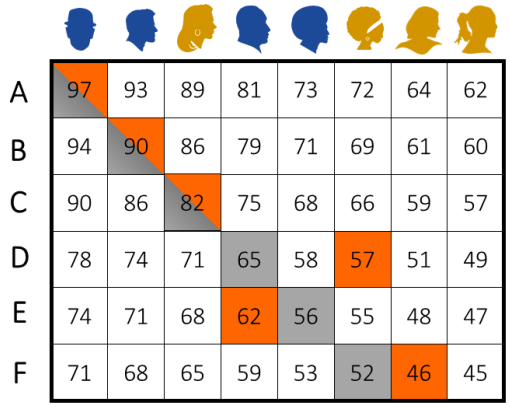
3 Our results

In this section, we present an overview of our results. The statements of theorems here are informal for the ease of readability and for the formal statements our results, we refer the reader to the full version of the paper [13].

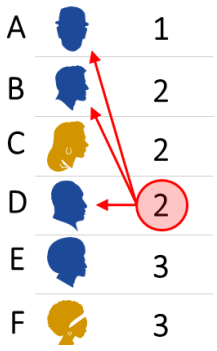
Let the *type*

$$T_i := \{\ell \in [p] : i \in P_\ell\}$$

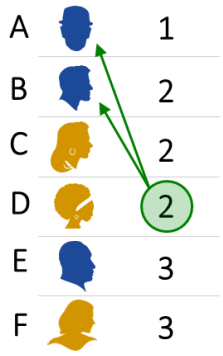
of item i be the set of properties that the item i has. Our first result is an exact algorithm for solving the constrained ranking maximization problem whose running time is polynomial if the number of distinct T_is, denoted by q, is constant.



(a) An example of a value matrix W. The values corresponding to the optimal (unconstrained) ranking in (b) and the optimal constrained ranking in (c) are depicted by gray and orange respectively. (Note that, for clarity of the rank order, the above is the transpose of the matrix referred to as W in the text.)



(b) The optimal unconstrained ranking. The upper-bound constraint at position D is violated as there are 3 men, but only 2 are allowed.



(c) The optimal constrained ranking. The upper-bound constraint at position D is no longer violated; in fact all constraints are satisfied.

Figure 1 A simple example of our framework: In (a) a matrix of W_{ij}s is presented. Here, the options are people who are either male (blue) or female (yellow), and 6 of them must be ranked. We assume that there is a single upper-bound constraint for each position in the ranking which is applied to both genders as depicted in figures (b) and (c). The constraints are satisfied in the latter, but not the former. The weights of these two rankings are depicted in figure (a).

▶ **Theorem 3.1** (Exact dynamic programming-based algorithm). *There is an algorithm that solves the constrained ranking maximization problem (LU) in $O(pqn^q + pm)$ time when the values W satisfy property* (1).

This algorithm combines a geometric interpretation of our problem along with dynamic programming and proceeds by solving a sequence of $q-$dimensional sub-problems. When q is allowed to be large, the problem is **NP**-hard; see Theorem 3.5.

Generally, we may not be able to assume that q is a constant and, even then, it would be desirable to have algorithms whose running time is close to $(m+n)p$, the size of the input. Towards this we consider a natural parameter of the set of properties: The size of the largest T_i, namely

$$\Delta := \max_{i \in [m]} |T_i|.$$

The complexity of the constrained ranking maximization problem turns out to show interesting behavior with respect to Δ (note that $\Delta \leq p$ and typically $p \ll q$). The case when $\Delta = 1$ corresponds to the simplest practical setting where there are p disjoint properties, i.e., the properties partition the set of items. For instance, a set of images of humans could be partitioned based on the ethnicity or age of the individual. Note that even though $q = p$ for $\Delta = 1$, this q could still be large and the previous theorem may have a prohibitively large running time.

When $\Delta = 1$ we prove that the constrained ranking maximization problem (LU) is polynomial time solvable even when the matrix W does not satisfy the property (1).

▶ **Theorem 3.2** (Polynomial time algorithm for $\Delta = 1$). *The constrained ranking maximization problem (LU) for general weights and $\Delta = 1$ can be solved in $\widetilde{O}(n^2 m)$ time.*

The above is obtained by reducing this variant of the ranking maximization problem to the minimum cost flow problem, that can be solved efficiently (the network is acyclic). We note that even though the running time is polynomial in m, it might be still not satisfactory for practical purposes. With the aim of designing faster – linear time algorithms, we focus on the case when only upper-bound constraints are present. For this case, we analyze a natural linear programming (LP) relaxation for the constrained ranking maximization problem (U). It reveals interesting structure of the problem and motivates a fast greedy algorithm. Formally, the relaxation considers the set $\Omega_{m,n}$ defined as

$$\Omega_{m,n} := \left\{ x \in [0,1]^{m \times n} : \sum_{j=1}^{n} x_{ij} \leq 1 \text{ for all } i \in [m], \quad \sum_{i=1}^{m} x_{ij} = 1, \text{ for all } j \in [n] \right\}$$

and the following linear program

$$\max_{x \in \Omega_{m,n}} \sum_{i=1}^{m} \sum_{j=1}^{n} W_{ij} x_{ij} \qquad \text{s.t.} \qquad \sum_{i \in P_\ell} \sum_{j=1}^{k} x_{ij} \leq U_{k\ell}, \ \forall\ \ell \in [p],\ k \in [n]. \tag{3}$$

Observe that in the absence of fairness constraints, (3) represents the maximum weight bipartite matching problem – it is well known that the feasible region of its fractional relaxation has integral vertices and hence the optimal values of these two coincide. However, in the constrained setting, even for $\Delta = 1$, it can be shown that the feasible region is no longer integral – it can have fractional vertices. For this reason, it is not true that maximizing any linear objective results in an integral solution. Surprisingly, we prove that for $\Delta = 1$ the cost functions we consider are special and never yield optimal fractional (vertex) solutions.

▶ **Theorem 3.3** (Exact LP-based algorithm for $\Delta = 1$)**.** *Consider the linear programming relaxation* (3) *for the constrained ranking maximization problem (U) when* $\Delta = 1$ *and the objective function satisfies* (1)*. Then there exists an optimal solution with integral entries and hence the relaxation is exact. Further, there exists a greedy algorithm to find an optimal integral solution in* $O(np + m)$ *time.*

The proof relies on a combinatorial argument on the structure of tight constraints that crucially uses the assumption that $\Delta = 1$ and the property (1) of the objective function. Note that the result of Theorem 3.3 implies in particular that whenever the linear program (3) is feasible then there is also an integer solution – a feasible ranking. This can be also argued for the general (LU) variant of the problem and its corresponding LP relaxation. However, extending Theorem 3.3 to this case seems more challenging and is left as an open problem.

When trying to design algorithms for larger Δ, the difficulty is that the constrained ranking *feasibility* problem remains **NP**-hard (in fact, even hard to approximate when feasibility is guaranteed) for $\Delta \geq 3$; see Theorem 3.5. Together, these results imply that unless we restrict to feasible instances of the constrained ranking problem, it is impossible to obtain any reasonable approximation algorithm for this problem. In order to bypass this barrier, we focus on the (U) variant of the problem and present an *algorithmically verifiable* condition for feasibility and argue that it is natural in the context of information retrieval. For each $1 \leq k \leq n$, we consider the set

$$S_k := \{l \in [p] : U_{(k-1)\ell} + 1 \leq U_{k\ell}\}$$

of all properties whose constraints increase by at least 1 when going from the $(k-1)$st to the kth position. We observe that the following *abundance of items* condition is sufficient for

feasibility:

$$\forall k \text{ there are at least } n \text{ items } i \text{ s.t. } T_i \subseteq S_k. \tag{4}$$

Intuitively, this says that there should be always at least a few ways to extend a feasible ranking of $(k-1)$ items to a ranking of k items. Simple examples show that this condition can be necessary for certain constraints $\{U_{k\ell}\}$. In practice, this assumption is almost never a problem – the available items m (e.g., webpages) far outnumber the size of the ranking n (e.g., number of results displayed in the first page) and the number of properties p (i.e., there are only so many "types" of webpages).

We show that assuming condition (4), there is a linear-time algorithm that achieves an $(\Delta+2)$-approximation, while only slightly violating the upper-bound constraints. This result does not need assumption (1), rather only that the W_{ij}s are non-negative. This result is near-optimal; we provide an $\Omega\left(\frac{\Delta}{\log \Delta}\right)$ hardness of approximation result (see Theorem 3.5 and the full version of the paper [13] for more details).

▶ **Theorem 3.4** ($(\Delta+2)$-approximation algorithm)**.** *For the constrained ranking maximization problem (U), under the assumption* (4)*, there is an algorithm that in linear time outputs a ranking x with value at least $\frac{1}{\Delta+2}$ times the optimal one, such that x satisfies the upper-bound constraints with at most a twice multiplicative violation, i.e.,*

$$\sum_{i \in P_\ell} \sum_{j=1}^{k} x_{ij} \leq 2U_{k\ell}, \quad \textit{for all } \ell \in [p] \textit{ and } k \in [n].$$

One can construct artificial instances of the ranking problem, where the output of the algorithm indeed violates upper-bound constraints with a 2-multiplicative factor. However, these violations are caused by the presence of high-utility items with a large number of properties. Such items are unlikely to appear in real-life instances and thus we expect the practical performance of the algorithm to be better than the worst-case bound given in Theorem 3.4 suggests. Lastly we summarize our hardness results for the constrained ranking problem.

▶ **Theorem 3.5** (Hardness Results – Informal)**.** *The following variants of the constrained ranking feasibility (U) and constrained ranking maximization (U) problem are* **NP***-hard.*

1. *Deciding feasibility for the case of $\Delta \geq 3$.*
2. *Under the feasibility condition* (4)*, approximating the optimal value of a ranking within a factor $O\left(\Delta/\log \Delta\right)$, for any $\Delta \geq 3$.*
3. *Deciding feasibility when only the number of items m, number of positions n, and upper-bounds u are given as input; the properties are fixed for every m.*
4. *For every constant c, deciding between whether there exists a feasible solution or every solution violates some constraint by a factor of c.*

Organization of the rest of the paper

In Section 4 we discuss other related work. Section 5 contains an overview of the proofs of our main results. For complete proofs, we refer the reader to the full version of the paper [13]. In Section 6 we provide a discussion of possible directions for future work and open problems.

4 Other related work

Information retrieval, which focuses on selecting and ranking subsets of data, has a rich history in computer science, and is a well-established subfield in and of itself; see, e.g., the foundational work by [39]. The probability ranking principle (PRP) forms the foundation of information retrieval research [32, 38]; in our context it states that *a system's ranking should order items by decreasing value.* Our problem formulation and solutions are in line with this – *subject to satisfying the diversity constraints.*

A related problem is diverse data summarization in which a subset of items with varied properties must be selected from a large set [35, 9], or similarly, voting with diversity constraints in which a subset of items of people with varied properties or attributes must be selected via a voting procedure [34, 10]. However, the formulation of these problem is considerably different as there is no need to produce a ranking of the selected items, and hence the analogous notion of constraints is more relaxed. Extending work on fairness in classification problems [45], the fair ranking problem has also been studied as an (unconstrained) multi-objective optimization problem, and various fairness metrics of a ranking have been proposed [43].

Combining the learning of values along with the ranking of items has also been studied [37, 40]; in each round an algorithm chooses an ordered list of k documents as a function of the estimated values W_{ij} and can receive a click on one of them. These clicks are used to update the estimate of the W_{ij}s, and bounds on the regret (i.e., learning rate) can be given using a bandit framework. In this problem, while there are different types of items that can affect the click probabilities, there are no constraints on how they should be displayed.

Recent work has shown that, in many settings, there are impossibility results that prevent us from attaining both *property* and *item* fairness [30]. Indeed, our work focuses on ensuring property fairness (i.e., no property is overrepresented), however this comes at an expense of item fairness (i.e., depending on which properties an item has, it may have much higher / lower probability of being displayed than another item with the same value). In our motivating application we deal with the ranking of documents or webpages, and hence are satisfied with this trade-off. However, further consideration may be required if, e.g., we wish to rank people as this would give individuals different likelihoods of being near the top of the list based on their properties rather than solely on their value.

5 Proof overviews

Overview of the proof of Theorem 3.1. We first observe that the constrained ranking maximization problem has a simple geometric interpretation. Every item $i \in [m]$ can be assigned a *property vector* $t_i \in \{0,1\}^p$ whose ℓ-th entry is 1 if item i has property ℓ and 0 otherwise. We can then think of the constrained ranking maximization problem as finding a sequence of n distinct items $i_1, i_2, \ldots, i_n$ such that $L_k \leq \sum_{j=1}^{k} t_{i_j} \leq U_k$ for all $k \in [n]$, where U_k is the vector whose ℓ-th entry is $U_{\ell k}$. In other words, we require that the partial sums of the vectors corresponding to the top k items in the ranking stay within the region $[L_{k1}, U_{k1}] \times [L_{k2}, U_{k2}] \times \cdots \times [L_{kn}, U_{kn}]$ defined by the fairness constraints.

Let $Q := \{t_i : i \in [m]\}$ be the set of all the different property vectors t_i that appear for items $i \in [m]$, and let us denote its elements by $v_1, v_2, \ldots, v_q$. A simple but important observation is that whenever two items $i_1, i_2 \in [m]$ (with say $i_1 < i_2$) have the same property vector: $t_{i_1} = t_{i_2}$, then in every optimal solution either i_1 will be ranked above i_2, only i_1 is ranked, or neither is used. This follows from the assumption that the weight matrix is monotone in i and j and satisfies the property as stated in (1).

Let us now define the following sub-problem that asks for the *property vectors* of a feasible solution: Given a tuple $(s_1, s_2, \ldots, s_q) \in \mathbb{N}^q$ such that $k = s_1 + s_2 + \cdots + s_q \leq n$, what is the optimal way to obtain a feasible ranking on k items such that s_j of them have property vector equal to v_j for all $j = 1, 2, \ldots, q$? Given a solution to this sub-problem, using the observation above, it is easy to determine which items should be used for a given property vector, and in what order. Further, one can easily solve such a sub-problem given the solutions to smaller sub-problems (with a smaller sum of s_js), resulting in a dynamic programming algorithm with $O(n^q)$ states and, hence, roughly the same running time.

Overview of the proof of Theorem 3.2. The main idea is to reduce ranking maximization to the minimum cost flow problem and then observe several structural properties of the resulting instance which allow one to solve it efficiently (in $\widetilde{O}(n^2 m)$ time).

Given an instance of the constrained ranking maximization problem (U), we construct a weighted flow network $G = (V, E)$ such that every feasible ranking corresponds to a feasible flow of value n in G. Roughly, for every item i and every property ℓ a chain of n vertices is constructed so that placing item i (such that $i \in P_\ell$) at position k corresponds to sending one unit of flow through the chain corresponding to item i up to its kth vertex and then switching to the chain corresponding to property ℓ. Edge weights in these gadgets (chains) are chosen in such a way that the cost of sending a unit through this path is $-W_{i,k}$. The capacities in chains corresponding to properties implement upper-bound constraints. The lower-bound constraints can be also enforced by putting appropriate weights on edges of these chains.

The instance of the minimum cost flow problem we construct has $O(nm)$ vertices and $O(nm)$ edges and is acyclic, which allows to replace the application of the Bellman-Ford algorithm in the first phase of the Successive Shortest Path algorithm by a linear-time procedure. This then easily leads to an implementation in $O(n^2 m \log m)$ time.

Overview of the proof of Theorem 3.3. Unlike the $\Delta = 0$ case where the LP-relaxation (3) has no non-integral vertex (it is the assignment polytope) , even when $p = 1$, fractional vertices can arise (see the full version of the paper [13]). Theorem 3.3 implies that for $\Delta = 1$, although the feasible region of (3) is not integral in all directions, it is along the directions of interest. In the proof we first reduce the problem to the case when $m = n$ (i.e., when one has to rank all of the items) and w has the *strict* form of property (1) (i.e., when the inequalities in assumption (1) are strict). Our strategy then is to prove that for every fractional feasible solution $x \in \Omega_{m,m}$ there is a direction $y \in \mathbb{R}^{m \times m}$ such that the solution $x' := x + \varepsilon y$ is still feasible (for some $\varepsilon > 0$) and its weight is larger than the weight of x. This implies that every optimal solution is necessarily integral.

Combinatorially, the directions we consider correspond to 4-cycles in the underlying complete bipartite graph, such that the weight of the matching can be improved by swapping edges along the cycle. The argument that shows the existence of such a cycle makes use of the special structure of the constraints in this family of instances.

To illustrate the approach, suppose that there exist two items $i_1 < i_2$ that have the same property $\ell \in [p]$, and for some ranking positions $j_1 < j_2$ we have

$$x_{i_1 j_2} > 0 \quad \text{and} \quad x_{i_2 j_1} > 0. \tag{5}$$

Following the strategy outlined above, consider $x' = x + \varepsilon y$ with $y \in \mathbb{R}^{m \times m}$ to be zero everywhere except $y_{i_1 j_1} = y_{i_2 j_2} = 1$ and $y_{i_1 j_2} = y_{i_2 j_1} = -1$. We would like to prove that the weight of x' is larger than the weight of x and that x' is feasible for some (possibly small)

$\varepsilon > 0$. The reason why we gain by moving in the direction of y follows from property (1). Feasibility in turn follows because y is orthogonal to every constraint defining the feasible region. Indeed, the only constraints involving items i_1, i_2 are those corresponding to the property ℓ. Further, every such constraint is of the form[1] $\langle 1_{R_k}, x\rangle \leq U_{k\ell}$ where 1_{R_k} is the indicator vector of a rectangle $R_k := P_\ell \times [k]$. Such a rectangle contains either all non-zero entries of y, two non-zero entries (with opposite signs), or none. In any of these cases, $\langle 1_{R_k}, y\rangle = 0$.

Using a reasoning as above, one can show that no configuration of the form (5) can appear in any optimal solution for i_1, i_2 that share a property ℓ. This implies that the support of every optimal solution has a certain structure when restricted to items that have any given property $\ell \in [p]$; this structure allows us to find an improvement direction in case the solution is not integral. To prove integrality we show that for every fractional solution $x \in \mathbb{R}^{m\times m}$ there exists a fractional entry $x_{ij} \in (0, 1)$ that can be slightly increased without violating the fairness constraints. Moreover since the i-th row and the j-th column must contain at least one more fractional entry each (since the row- and column-sums are 1), we can construct (as above) a direction y, along which the weight can be increased. The choice of the corresponding entries that should be altered requires some care, as otherwise we might end up violating fairness constraints.

The second part of Theorem 3.3 is an algorithm for solving the constrained ranking maximization problem for $\Delta = 1$ in optimal (in the input size) running time of $O(np + m)$. We show that a natural greedy algorithm can be used. More precisely, one iteratively fills in ranking positions by always selecting the *highest value* item that is still available and does not lead to a constraint violation. An inductive argument based that relies on property 1 and the $\Delta = 1$ assumption gives the correctness of such a procedure.

Overview of the proof of Theorem 3.4. Let $\Delta > 1$ be arbitrary. The most important part of our algorithm is a greedy procedure that finds a large weight solution to a slightly relaxed problem in which not all positions in the ranking have to be occupied. It processes pairs $(i, j) \in [m] \times [n]$ in non-increasing order of weights W_{ij} and puts item i in position j whenever this does not lead to constraint violation.

To analyze the approximation guarantee of this algorithm let us first inspect the combinatorial structure of the feasible set. In total there are $p \cdot n$ fairness constraints in the problem and additionally $m + n$ "matching" constraints, saying that no "column" or "row" can have more than a single one in the solution matrix $x \in \{0, 1\}^{m\times n}$. However, after relaxing the problem to the one where not all ranking positions have to be filled, one can observe that the feasible set is just an intersection of $p + 2$ matroids on the common ground set $[m] \times [n]$. Indeed, two of them correspond to the matching constraints, and are partition matroids. The remaining p matroids correspond to properties: for every property ℓ there is a chain of subsets $S_1 \subseteq S_2 \subseteq \cdots \subseteq S_n$ of $[m] \times [n]$ such that

$$\mathcal{I}_\ell = \{S \subseteq [m] \times [n] : |S \cap S_k| \leq U_{k\ell} \text{ for all } k = 1, 2, \ldots, n\}$$

is the set of independent sets in this (laminar) matroid. In the work [27] it is shown that the greedy algorithm run on an intersection of K matroids yields K-approximation, hence $(p + 2)$-approximation of our algorithm follows.

[1] By $\langle \cdot, \cdot \rangle$ we denote the inner product between two matrices, i.e., if $x, y \in \mathbb{R}^{m\times n}$ then $\langle x, y\rangle := \sum_{j=1}^{m} \sum_{i=1}^{n} x_{ij} y_{ij}$.

To obtain a better – $(\Delta + 2)$-approximation bound, a more careful analysis is required. The proof is based on the fact that, roughly, if a new element is added to a feasible solution S, then at most $\Delta + 2$ elements need to be removed from S to make it again feasible. Thus adding greedily one element can cost us absence of $\Delta + 2$ other elements of weight at most the one we have added. This idea can be formalized and used to prove the $(\Delta + 2)$-approximation of the greedy algorithm. This is akin to the framework of K-extendible systems by [33] in which this greedy procedure can be alternatively analyzed. Finally, we observe that since the problem solved was a relaxation of the original ranking maximization problem, the approximation ratio we obtain with respect to the original problem is still $(\Delta + 2)$.

It remains to complete the ranking by filling in any gaps that may have been left by the above procedure. This can be achieved in a greedy manner that only increases the value of the solution, and violates the constraints by at most a multiplicative factor of 2.

Overview of the proof of Theorem 3.5. Our hardness results are based on a general observation that one can encode various types of packing constraints using instances of the constrained ranking maximization (U) and feasibility (U) problem. The first result (part 1. in Theorem 3.5) – **NP**-hardness of the feasibility problem (for $\Delta \geq 3$) is established by a reduction from the hypergraph matching problem. Given an instance of the hypergraph matching problem one can think of its hyperedges as items and its vertices as properties. Degree constraints on vertices can then be encoded by upper-bound constraints on the number of items that have a certain property in the ranking. The inapproximability result (part 2. in Theorem 3.5) is also established by a reduction from the hypergraph matching problem, however in this case one needs to be more careful as the reduction is required to output instances that are feasible.

Our next hardness result (part 3. in Theorem 3.5) illustrates that the difficulty of the constrained ranking optimization problem (U) could be entirely due to the upper-bound numbers $U_{k\ell}$s. In particular, even when the part of the input corresponding to which item has which property is fixed, and only depends on m (and, hence, can be *pre-processed* as in [22]), the problem remains hard. This is proven via a reduction from the independent set problem. The properties consists of all pairs of items $\{i_1, i_2\}$ for $i_1, i_2 \in [m]$. Given any graph $G = (V, E)$ on m vertices, we can set up a constrained ranking problem whose solutions are independent sets in G of a certain size. Since every edge $e = \{i_1, i_2\} \in E$ is a property, we can set a constraint that allows at most one item (vertex) from this property (edge) in the ranking.

Finally, part 4. in Theorem 3.5 states that it is not only hard to decide feasibility but even to find a solution that does not violate any constraint by more than a constant multiplicative factor $c \in \mathbb{N}$. The obstacle in proving such a hardness result is that, typically, even if a given instance is infeasible, it is easy to find a solution that violates *many* constraints by a small amount. To overcome this problem we employ an inapproximability result for the maximum independent set problem by [25] and an idea by [16]. Our reduction (roughly) puts a constraint on every $(c+1)-$clique in the input graph $G = (V, E)$, so that at most one vertex (item) is picked from it. Then a solution that does not violate any constraint by a multiplicative factor more than c corresponds to a set of vertices S such that the induced subgraph $G[S]$ has no c-clique. Such a property allows us to prove (using elementary bounds on Ramsey numbers) that G has a large independent set. Hence, given an algorithm that is able to find a feasible ranking with no more than a c-factor violation of the constraints, we can approximate the maximum size of an independent set in a graph $G = (V, E)$ up to a factor of roughly $|V|^{1-1/c}$; which is hard by [25].

6 Discussion and future work

In this paper, motivated by controlling and alleviating algorithmic bias in information retrieval, we initiate the study of the complexity of a natural constrained optimization problem concerning rankings. Our results indicate that the constrained ranking maximization problem, which is a generalization of the classic bipartite matching problem, shows fine-grained complexity. Both the structure of the constraints and the numbers appearing in upper-bounds play a role in determining its complexity. Moreover, this problem generalizes several hypergraph matching/packing problems. Our algorithmic results bypass the obstacles implicit in the past theory work by leveraging on the structural properties of the constraints and common objective functions from information retrieval. More generally, our results not only contribute to the growing set of algorithms to counter algorithmic bias for fundamental problems [19, 4, 44, 9, 11, 8, 29, 15], the structural insights obtained may find use in other algorithmic settings related to the rather broad scope of ranking problems.

Our work also suggests some open problems and directions. The first question concerns the $\Delta = 1$ case and its (LU) variant; Theorem 3.2 implies that it can be solved in $\widetilde{O}(n^2m)$ time, can this be improved to nearly-linear time, as we do for the (U) variant (Theorem 3.3)? Another question is the complexity of the constrained ranking maximization problem (in all different variants) when $\Delta = 2$ – is it in $\mathbf{P}$? The various constants appearing in our approximation algorithms are unlikely to be optimal and improving them remains important. In particular, our approximation algorithm for the case of large Δ in Theorem 3.4 may incur a 2-multiplicative violation of constraints. This could be significant when dealing with instances where the upper bound constraints are rather large (i.e., $U_{k\ell} \gtrsim \frac{k}{2}$) in which case, such a violation effectively erases all the constraints. It is an interesting open problem to understand whether this 2-violation can be avoided, either by providing a different algorithm or by making different assumptions on the instance.

In this work we consider linear objective functions for the the ranking optimization problem, i.e., the objective is an independent sum of profits for individual item placements. While this model might be appropriate in certain settings, there may be cases where one would prefer to measure the quality of a ranking as a whole, and in particular the utility of placing a given item at the kth position should also depend on what items were placed above it [46]. Thus defining and studying a suitable variant of the problem for a class of objectives on rankings that satisfy some version of the diminishing returns principle (submodularity), is of practical interest.

A related question that deserves independent exploration is to study the complexity of sampling a constrained ranking from the probability distribution induced by the objective (rather than outputting the ranking that maximizes its value, output a ranking with probability proportional to its value). Finally, extending our results to the online setting seems like an important technical challenge which is also likely to have practical consequences.

References

1 Sanjeev Arora, Alan M. Frieze, and Haim Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 21–30, 1996. doi:10.1109/SFCS.1996.548460.

2 Drake Baer. The 'Filter Bubble' Explains Why Trump Won and You Didn't See It Coming, November 2016. NY Mag.

3 Eytan Bakshy, Solomon Messing, and Lada A Adamic. Exposure to ideologically diverse news and opinion on facebook. *Science*, 348(6239):1130–1132, 2015.

4 S. Barocas and A.D. Selbst. *Big Data's Disparate Impact*. SSRN eLibrary, 2015.

5 Tolga Bolukbasi, Kai-Wei Chang, James Y Zou, Venkatesh Saligrama, and Adam T Kalai. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings. In *Advances in Neural Information Processing Systems*, pages 4349–4357, 2016.

6 Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

7 Jaime Carbonell and Jade Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336. ACM, 1998.

8 L. Elisa Celis, Amit Deshpande, Tarun Kathuria, Damian Straszak, and Nisheeth K. Vishnoi. On the complexity of constrained determinantal point processes. In *APPROX/RANDOM 2017*, pages 36:1–36:22, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.36.

9 L. Elisa Celis, Amit Deshpande, Tarun Kathuria, and Nisheeth K Vishnoi. How to be fair and diverse? *Fairness, Accountability and Transparency in Machine Learning*, 2016.

10 L. Elisa Celis, Lingxiao Huang, and Nisheeth K. Vishnoi. Multiwinner voting with fairness constraints. In *IJCAI-ECAI*, 2018.

11 L. Elisa Celis, Vijay Keswani, Damian Straszak, Amit Deshpande, Tarun Kathuria, and Nisheeth K. Vishnoi. Fair and diverse DPP-based data summarization. *CoRR*, abs/1802.04023, 2018. arXiv:1802.04023.

12 L. Elisa Celis, Peter M. Krafft, and Nathan Kobe. Sequential voting promotes collective discovery in social recommendation systems. In *Proceedings of the Tenth International Conference on Web and Social Media, Cologne, Germany, May 17-20, 2016.*, pages 42–51, 2016. URL: http://www.aaai.org/ocs/index.php/ICWSM/ICWSM16/paper/view/13160.

13 L. Elisa Celis, Damian Straszak, and Nisheeth K. Vishnoi. Ranking with fairness constraints. *CoRR*, abs/1704.06840, 2017. arXiv:1704.06840.

14 L. Elisa Celis and Siddartha Tekriwal. What Do Users Want in Q&A Sites: Quality or Diversity? In *International Conference on Computational Social Science (IC2S2)*, 2017.

15 L. Elisa Celis and Nisheeth K. Vishnoi. Fair Personalization. *Fairness, Accountability, and Transparency in Machine Learning*, 2017.

16 Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35(3):713–728, 2005. doi:10.1137/S0097539700382820.

17 Charles LA Clarke, Maheedhar Kolla, Gordon V Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 659–666. ACM, 2008.

18 Matthew Costello, James Hawdon, Thomas Ratliff, and Tyler Grantham. Who views online extremism? Individual attributes leading to exposure. *Computers in Human Behavior*, 63:311–320, 2016.

19 Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *ITCS*, New York, NY, USA, 2012. ACM. doi:10.1145/2090236.2090255.

20 Cynthia Dwork, Ravi Kumar, Moni Naor, and Dandapani Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th international conference on World Wide Web*, pages 613–622. ACM, 2001.

21 Robert Epstein and Ronald E. Robertson. The search engine manipulation effect (SEME) and its possible impact on the outcomes of elections. *Proceedings of the National Academy of Sciences*, 112(33):E4512–E4521, 2015. doi:10.1073/pnas.1419828112.

22 Uriel Feige and Shlomo Jozeph. Universal factor graphs. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, pages 339–350, 2012. doi:10.1007/978-3-642-31594-7_29.

23 Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *Proceedings of the 18th international conference on World wide web*, pages 381–390. ACM, 2009.

24 Fabrizio Grandoni, R Ravi, Mohit Singh, and Rico Zenklusen. New approaches to multi-objective optimization. *Mathematical Programming*, 146(1-2):525–554, 2014.

25 J. Hastad. Clique is Hard to Approximate Within $n^{1-\varepsilon}$. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, FOCS '96. IEEE Computer Society, 1996.

26 Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

27 T.A. Jenkyns. The Efficacy of the 'greedy' Algorithm. In *Proc. of 7th S-E. Conf. on Combinatorics, Graph Theory and Computing*, pages 341–350, 1976.

28 Matthew Kay, Cynthia Matuszek, and Sean A Munson. Unequal representation and gender stereotypes in image search results for occupations. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3819–3828. ACM, 2015.

29 Keith Kirkpatrick. Battling algorithmic bias: how do we ensure algorithms treat us fairly? *Communications of the ACM*, 59(10):16–17, 2016.

30 Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. Inherent trade-offs in the fair determination of risk scores. *Innovations in Theoretical Computer Science*, 2017.

31 Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*. Cambridge university press Cambridge, 2008.

32 Melvin Earl Maron and John L Kuhns. On relevance, probabilistic indexing and information retrieval. *Journal of the ACM (JACM)*, 7(3):216–244, 1960.

33 Julián Mestre. Greedy in approximation algorithms. In *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, pages 528–539, 2006. doi:10.1007/11841036_48.

34 Burt L Monroe. Fully proportional representation. *American Political Science Review*, 89(4):925–940, 1995.

35 Debmalya Panigrahi, Atish Das Sarma, Gagan Aggarwal, and Andrew Tomkins. Online selection of diverse results. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 263–272. ACM, 2012.

36 Filip Radlinski, Paul N Bennett, Ben Carterette, and Thorsten Joachims. Redundancy, diversity and interdependent document relevance. In *ACM SIGIR Forum*, volume 43, pages 46–52. ACM, 2009.

37 Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th International conference on Machine learning*, pages 784–791. ACM, 2008.

38 Stephen E Robertson. The probability ranking principle in ir. *Journal of documentation*, 33(4):294–304, 1977.

39 Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24(5):513–523, 1988. doi:10.1016/0306-4573(88)90021-0.

40 Aleksandrs Slivkins, Filip Radlinski, and Sreenivas Gollapudi. Ranked bandits in metric spaces: learning diverse rankings over large document collections. *Journal of Machine Learning Research*, 14(Feb):399–436, 2013.

41 Charles Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 15(1):72–101, 1904.

42 Aravind Srinivasan. Improved approximations of packing and covering problems. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 268–276, 1995. doi:10.1145/225058.225138.

43 Ke Yang and Julia Stoyanovich. Measuring fairness in ranked outputs. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management, Chicago, IL, USA, June 27-29, 2017*, pages 22:1–22:6, 2017. doi:10.1145/3085504.3085526.

44 Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rodriguez, and Krishna Gummadi. Fairness Constraints: A Mechanism for Fair Classification. In *Fairness, Accountability, and Transparency in Machine Learning*, 2015. URL: http://www.fatml.org/cfp.html.

45 Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In *Proceedings of The 30th International Conference on Machine Learning*, pages 325–333, 2013.

46 Cheng Xiang Zhai, William W Cohen, and John Lafferty. Beyond independent relevance: methods and evaluation metrics for subtopic retrieval. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 10–17. ACM, 2003.

47 Mi Zhang and Neil Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 123–130. ACM, 2008.

48 Cai-Nicolas Ziegler, Sean M McNee, Joseph A Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, pages 22–32. ACM, 2005.

Interpolating between k-Median and k-Center: Approximation Algorithms for Ordered k-Median

Deeparnab Chakrabarty
Dept. of Computer Science, Dartmouth College, Hanover, NH 03755-3510, USA
deeparnab@dartmouth.edu

Chaitanya Swamy[1]
Dept. of Combinatorics and Optimization, Univ. Waterloo, Waterloo, ON N2L 3G1, Canada
cswamy@uwaterloo.ca

Abstract

We consider a generalization of k-median and k-center, called the *ordered k-median* problem. In this problem, we are given a metric space $(\mathcal{D}, \{c_{ij}\})$ with $n = |\mathcal{D}|$ points, and a non-increasing weight vector $w \in \mathbb{R}^n_+$, and the goal is to open k centers and assign each point $j \in \mathcal{D}$ to a center so as to minimize $w_1 \cdot$ (largest assignment cost) $+ w_2 \cdot$ (second-largest assignment cost) $+ \ldots + w_n \cdot$ (n-th largest assignment cost). We give an $(18 + \epsilon)$-approximation algorithm for this problem. Our algorithms utilize Lagrangian relaxation and the primal-dual schema, combined with an enumeration procedure of Aouad and Segev. For the special case of $\{0, 1\}$-weights, which models the problem of minimizing the ℓ largest assignment costs that is interesting in and of by itself, we provide a novel reduction to the (standard) k-median problem, showing that LP-relative guarantees for k-median translate to guarantees for the ordered k-median problem; this yields a nice and clean $(8.5 + \epsilon)$-approximation algorithm for $\{0, 1\}$ weights.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Approximation algorithms, Clustering, Facility location, Primal-dual method

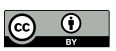
Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.29

Related Version A full version of the paper is available at [5], `https://arxiv.org/abs/1711.08715`.

1 Introduction

Clustering is an ubiquitous problem that finds applications in various fields including data mining, machine learning, image processing, and bioinformatics. Many clustering problems involve finding a set F of at most k "centers" from an underlying set $\mathcal{D}$ of data points located in some metric space $\{c_{ij}\}_{i,j \in \mathcal{D}}$, and an assignment of data points to centers, so as to minimize some objective function of the assignment costs, i.e., the distances between data points and their assigned centers. These problems can typically also be stated as *facility-location* problems, wherein we seek a cost-effective way of opening facilities ($\equiv$ centers) and assigning clients ($\equiv$ data points) to open facilities. Given their widespread applicability, clustering and facility-location problems have been extensively studied in the Computer Science and Operations Research literature; see, e.g., [16, 22], as also the literature on the classical *k-median* (minimize *sum* of the assignment costs) [6, 13, 15, 4]), and *k-center* (minimize *maximum* assignment cost [10, 11]) problems.

[1] Supported in part by NSERC grant 327620-09 and an NSERC Discovery Accelerator Supplement Award.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 29; pp. 29:1–29:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We consider a common generalization of k-median and k-center, called the *ordered k-median problem* [17, 9]. As before, we are given a metric space $(\mathcal{D}, \{c_{ij}\}_{i,j\in\mathcal{D}})$, and an integer $k \geq 0$. We will often refer to points in $\mathcal{D}$ as clients. We are also given non-increasing, nonnegative weights $w_1 \geq w_2 \geq \ldots \geq w_n \geq 0$, where $n = |\mathcal{D}|$. For a vector $v \in \mathbb{R}^{\mathcal{D}}$, we use $v^{\downarrow}$ to denote the vector v with coordinates sorted in non-increasing order. That is, we have $v_i^{\downarrow} = v_{\sigma(i)}$, where σ is a permutation of $\mathcal{D}$ such that $v_{\sigma(1)} \geq v_{\sigma(2)} \geq \ldots v_{\sigma(n)}$. The goal in the ordered k-median problem is to choose a set F of k points from $\mathcal{D}$ as centers (or "facilities"), and assign each client $j \in \mathcal{D}$ to a center $i(j) \in F$, so as to minimize

$$cost\big(w; \vec{c} := \{c_{i(j)j}\}_{j\in\mathcal{D}}\big) := w^T \vec{c}^{\downarrow} = \sum_{j=1}^{n} w_j \vec{c}_j^{\downarrow}.$$

Observe that when all the w_is are 1, we obtain the k-median problem; on the other hand, setting $w_1 = 1$, $w_2 = \ldots = w_n = 0$, yields the k-center problem. Indeed the special case with $\{0, 1\}$ weights is already interesting: that is, for some $\ell \in [n]$, we have $w_1 = \ldots = w_\ell = 1$ and all the remaining w_is are 0; this captures the problem of minimizing the ℓ largest assignment costs, which Tamir [23] calls the *ℓ-centrum problem*.

The ordered k-median problem can be motivated from various perspectives. The problem was proposed in network location theory as a convenient way of unifying the k-median and k-center objectives, as also some other objective functions considered in location theory (see, e.g., [17]). Such a versatile model is also useful in the context of clustering applications, wherein the clustering objective (e.g., k-median or k-center) is often a means to an end, namely, producing a "good" clustering. The ordered k-median problem yields a suite of clustering objectives, including those that interpolate between the k-median and k-center objectives, and thereby offers a useful means of obtaining a variety of clustering solutions (which motivates the question of developing efficient algorithms for (approximately) solving this problem). Another motivation for studying ordered k-median comes from a fairness perspective: if the weights decrease geometrically (at a sufficiently large rate), then an optimal ordered-k-median solution yields a *min-max fair* assignment-cost vector: that is, a solution that minimizes the maximum assignment cost, subject to which, it minimizes the second largest assignment cost, and so on. Finally, the ℓ-centrum problem can also be interpreted as the following *robust-optimization* version of k-median. Suppose there is some uncertainty in the client-set that needs to be clustered: in every scenario, some (at most) ℓ clients need to be clustered, and we need to determine the k centers and the assignment of clients to centers before knowing the scenario realization. Robust optimization seeks to minimize the maximum scenario cost, which leads to precisely the ℓ-centrum problem.

While the special cases of k-median and k-center have been considered extensively from the viewpoint of developing approximation algorithms, much less is known about the approximability of the ordered k-median problem, especially in general metrics. Aouad and Segev [2] obtained a logarithmic-approximation ratio for general metrics, and Alamdari and Shmoys [1] obtain a bicriteria approximation for the special case, where w is a convex combination of $(1, 0, \ldots, 0)$ and $\left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right)$, which is called the *centridian* problem [12].

Our results. We obtain constant-factor approximation algorithms for the ordered k-median problem. Together with the concurrent work of [3], these constitute the first constant-factor approximation guarantees for ordered k-median. Our main result is an (deterministic) $(18+\epsilon)$-approximation algorithm for the ordered k-median problem (Theorem 7). Our algorithm utilizes the *primal-dual schema and Lagrangian relaxation*, and, hence, is *combinatorial*.

En route, in Section 2, we first develop constant-factor approximation algorithms for the case of $\{0, 1\}$-weights. This introduces many of the ideas needed to handle the general

setting. We design two algorithms for this setting. Both algorithms are derived using a novel LP-relaxation that we propose for the problem, which leverages a key insight to circumvent the issue that the natural LP-relaxation has a large (non-constant) integrality gap.

Our first algorithm is a clean, combinatorial $(12+\epsilon)$-approximation algorithm that is based on the Jain-Vazirani primal-dual schema coupled with Lagrangian relaxation (Theorem 4). Both the algorithm and its analysis are versatile, and we show in Section 3 that the underlying ideas extend easily and, in combination with an enumeration procedure of [2], yield an $(18+\epsilon)$-approximation for the general setting. Our second algorithm for $\{0,1\}$-weights is based on LP-rounding, and yields an improved approximation factor via a novel *black-box reduction* to LP-relative algorithms for (standard) k-median. We show that an LP-relative α-approximation for k-median yields (essentially) a $2(\alpha+1)$-approximation; taking $\alpha = 3.25$ [7], we obtain an $(8.5+\epsilon)$-approximation for ordered k-median with $\{0,1\}$-weights (Theorem 5); we believe that this reduction is of independent interest.

Relationship with the work of [3]. Recently, we learnt that Byrka et al. [3] have also obtained a (randomized) $O(1)$-approximation guarantee (equal to $38+\epsilon$) for the ordered k-median problem. Our work was done independently and concurrently; a manuscript with the same approximation guarantees was posted on the arXiv in November 2017 [5]. In particular, our results for $\{0,1\}$ weights were obtained without knowledge of the work of [3]. But it was after we learnt of the results in [3] that we realized that our results can be extended to the general weighted setting.

While we use similar LP relaxations, our techniques are different. Whereas [3] crucially exploit properties of the Charikar-Li [7] LP-rounding algorithm, we leverage the (primal-dual + Lagrangian relaxation) methodology for k-median due to Jain and Vazirani [13]. Our algorithms are thus combinatorial. Our approximation factors improve upon those obtained in [3], *both* for $\{0,1\}$ weights and general weights; we believe that our algorithms and analyses are also simpler. Finally, our reduction to LP-relative algorithms for k-median shows that we do not need to rely on a *specific* k-median LP-rounding algorithm in order to tackle ordered k-median with $\{0,1\}$ weights, and suggests that the same might be true for general weights.

Our techniques. It is instructive to first discuss the $\{0,1\}$-weighted case. One of the main challenges is in coming up with a good LP-relaxation for this ℓ-centrum problem. The natural LP-relaxation augments the natural LP for k-median by imposing constraints encoding that the total assignment cost of any set of ℓ clients is at most B, where B is a new variable that we seek to minimize. It is well known that, even for (standard) k-median, one cannot hope to round an LP solution while approximately preserving the assignment cost of *each* client [6].[2] More significantly, whereas we can round and approximately preserve the sum of *all* assignment costs (as shown by k-median rounding), it turns out that we cannot preserve the sum of the ℓ largest assignment costs: the natural LP has a large (non-constant) integrality gap. This integrality gap is robust and cannot be alleviated by guessing the maximum assignment cost and incorporating this in the LP and the lower bound.[3] In essence, the cause for this disparity (between k-median and ℓ-centrum) is that the k-median objective crucially also includes the contribution from clients with small assignment costs.

[2] This is possible if we open $O(k)$ centers, using, e.g., the filtering-based algorithm of [21] for facility location.

[3] This is in contrast with k-center, where such preprocessing does mitigate the bad integrality gap of the natural LP and reduces it to a constant.

The key insight that allows us to circumvent this difficulty is the following. Suppose we aim to find a solution of objective value $O(B)$. Then, it suffices to find a solution where the total assignment cost of clients having assignment cost at least B/ℓ is $O(B)$: the remaining clients can contribute at most additional B towards the ℓ-centrum objective, since we consider at most ℓ clients in the ℓ-centrum objective value. Moreover, if there is a solution of ℓ-centrum objective value at most B, then the total assignment cost of clients with assignment cost at least B/ℓ is at most B. Thus, given a "guess" B of the optimal value, *our new LP* (P_B) *seeks to minimize the total assignment cost of clients having assignment cost larger than* B/ℓ.

The LP (P_B) corresponds to the LP-relaxation for k-median with *non-metric distances* given by $\{f_B(c_{ij})\}_{i,j\in\mathcal{D}}$, where $f_B(d) = d$ if $d \geq B/\ell$, and is 0 otherwise. Despite this complication, we devise two ways of leveraging (P_B) to obtain a solution of ℓ-centrum cost $O(OPT_B + B)$ (which yields an $O(1)$-approximation for the correct choice of B), both of which involve simple procedures with a clean analysis; here, OPT_B denotes the optimal value of (P_B). Our first algorithm is based on the Jain-Vazirani (JV) template [13]. This is our main result for $\{0,1\}$ weights (see Section 2.1), and this algorithm extends easily to the setting with general weights. We Lagrangify the cardinality constraint and move to the facility-location (FL) version where we may choose any number of centers but incur a fixed cost of (say) λ for each center we choose. We adapt the JV primal-dual algorithm and its analysis to obtain a so-called Lagrangian-multiplier-preserving guarantee for this FL version. By fine-tuning λ, we can then find two solutions, one with less than k centers and the other with more than k centers, whose convex combination has low cost; rounding this *bipoint solution* yields the final solution. This yields our 12-approximation algorithm.

The second algorithm utilizes LP-rounding. We show that after a clustering step, where we merge clients that are distance at most $\frac{B}{\ell}$-apart, the problem of rounding a solution to (P_B) reduces to that of rounding a fractional k-median solution on the cluster centers. Thus, any LP-relative α-approximation algorithm for k-median can be used to obtain a solution of cost at most $2(\alpha+1)\overline{B}$.

For general weights, the key again is to consider k-median with suitable (non-metric) proxy distances analogous to the $f_B(c_{ij})$s. We utilize a clever enumeration idea due to [2] to obtain these proxy distances. Whereas with $\{0,1\}$ weights, we created two distance buckets ($c_{ij} \geq B/\ell$ and $c_{ij} < B/\ell$) with weight multipliers 1 and 0, we now create $O\big(\log_{1+\epsilon}(\frac{n}{\epsilon})\big)$ buckets by grouping distances in powers of $(1+\epsilon)$. We guess the average weight (roughly speaking) incurred for a bucket by an optimal solution, and use this as the weight multiplier for the bucket. As argued in [2]: (a) if we enumerate average weights in powers of $(1+\epsilon)$ then there are only polynomially many choices; and (b) the resulting proxy distances provide a good approximation for the actual $cost(w;.)$-cost. Finally, we show that the primal-dual algorithm and its analysis developed in Section 2.1 extends to solve the k-median problem with these new proxy distances. Combining these ingredients, we obtain an $(18+\epsilon)$-approximation.

Other related work. While the ordered k-median problem, and its special cases, have been well studied in the Operations Research literature (see, e.g., [18, 14]), much of this work has focused either on modeling issues and formulations, or on solving the problem exactly in special cases, or via (non-polynomial time) heuristics. There is little prior work (i.e., discounting [3]) on the design of approximation algorithms for this problem, in general metrics. As mentioned earlier, for general metrics, we are only aware of the work of [2], who obtain a logarithmic-approximation ratio, and [1], who obtain a bicriteria approximation for the special case of the centridian problem.

A significant amount of research has taken place for special cases of the problem, e.g., the $k = 1$ setting [17], and the "continuous" version of the problem where centers can also be opened "in the middle of an edge" [19]. For these settings, fast exact algorithms have been developed in many interesting cases; see, e.g., [8, 23, 20] and the references therein. There is also a large body of work looking at compact integer-programming formulations, branch and bound methods etc.; for a detailed account of this and other work related to location theory and ordered-median models, we refer the reader to the books [18, 14].

2 The setting with $\{0,1\}$-weights

We first consider the setting with $\{0,1\}$ weights. Let $\ell \in [n]$ be such $w_1 = \ldots = w_\ell = 1$, $w_{\ell+1} = 0 = \ldots = w_n$. We abbreviate $cost(w;\vec{c})$ to $cost(\ell;\vec{c})$, or simply $cost(\vec{c})$. The $\{0,1\}$-weight setting serves as a natural starting point for two reasons. First, the problem of minimizing the ℓ most expensive assignment costs is a natural, well-motivated problem that is interesting in its own right. Second, the study of the $\{0,1\}$-case serves to introduce some of the key underlying ideas that are also used to handle the general setting. Notice also that a non-decreasing weight vector w can be written as a nonnegative linear-combination of such $\{0,1\}$ weight vectors.

The natural LP-relaxation for this ℓ-centrum problem has an $\Omega(\ell)$ integrality gap, and, as noted earlier, the integrality gap does not decrease even if we guess the maximum assignment cost and incorporate this in our LP and lower bound. Our constant-factor approximation algorithms are based on an alternate novel LP-relaxation, where, given a "guess" B of the optimal value, we seek to minimize the total assignment cost of clients having assignment cost at least B/ℓ. The rationale is that assignment costs that are smaller than B/ℓ can contribute at most B to the ℓ-centrum cost, and can hence be ignored when searching for a solution of ℓ-centrum cost $O(B)$. For $d \geq 0$, define $f_B(d) = d$ if $d \geq B/\ell$, and 0 otherwise. Throughout, i and j index points of $\mathcal{D}$. We consider the following LP.

$$\min \quad \sum_j \sum_i f_B(c_{ij}) x_{ij} \tag{P_B}$$

$$\text{s.t.} \quad \sum_i x_{ij} \geq 1 \qquad \text{for all } j \tag{1}$$

$$0 \leq x_{ij} \leq y_i \qquad \text{for all } i,j \tag{2}$$

$$\sum_i y_i \leq k. \tag{3}$$

Variable y_i indicates if facility i is open (i.e., i is chosen as a center), and x_{ij} indicates if client j is assigned to facility i. The first two constraints say that each client must be assigned to an open facility, and the third constraint encodes that at most k centers may be chosen.

An atypical aspect of our relaxation is that, while an integer solution corresponds to a solution to our problem, its objective value under (P_B) may *underestimate* the actual objective value; however, as alluded to above, the objective value of (P_B) is within an additive B of the actual objective value. Let OPT_B denote the optimal value of (P_B), and *opt* denote the optimal value of the ℓ-centrum problem.

▶ **Claim 1.** *If $B \geq opt$, then $OPT_B \leq opt \leq B$.*

Proof. Let $(\tilde{x}, \tilde{y})$ be the integer point corresponding to an optimal solution. Clearly, $(\tilde{x}, \tilde{y})$ is feasible to (P_B). There are at most ℓ assignment costs that are at least opt/ℓ (and hence at least B/ℓ). Therefore, the objective value of $(\tilde{x}, \tilde{y})$ is at most *opt*. ◀

▶ **Claim 2.** *Let $\vec{c}$ be an assignment-cost vector (where $\vec{c}_j$ is the assignment cost of j). Then, $cost(\ell;\vec{c}) \leq \sum_j f_B(\vec{c}_j) + B$.*

▶ **Claim 3.** *For any $B \geq 0$, we have: (i) $f_B(x) \leq f_B(y)$ if $x \leq y$; (ii) $\max\{f_B(x), f_B(y), f_B(z)\} \geq f_B\big(\frac{x+y+z}{3}\big)$ for any $x, y, z \geq 0$; and (iii) $3f_B(x/3) = f_{3B}(x)$ for any $x \geq 0$.*

We may assume that we have $\overline{B} \leq (1+\epsilon)opt$ (e.g., by enumerating all possible choices for opt in powers of $(1+\epsilon)$, or using binary search to find, within a $(1+\epsilon)$-factor, the smallest B such that $OPT_B \leq B$). While (P_B) closely resembles the LP-relaxation for k-median, notice that the assignment costs $\{f_B(c_{ij})\}$ used in the objective of (P_B) *do not form a metric*. Despite this complication, we show that ($\mathrm{P}_{\overline{B}}$) can be leveraged to obtain a solution of $cost(\ell;.)$-cost $O(\overline{B})$. We devise two algorithms for obtaining such a guarantee. The first algorithm is based on the primal-dual method and the Jain-Vazirani (JV) template [13]; this yields a 12-approximation algorithm. The second algorithm is based on LP-rounding, and shows that any LP-relative α-approximation algorithm for k-median can be used to obtain a solution of $cost(\ell.)$-cost at most $2(\alpha+1)\overline{B}$.

▶ **Theorem 4.** *We can obtain a solution to the ℓ-centrum problem of cost at most $\big(12 + O(\epsilon)\big) \cdot \overline{B} \leq \big(12 + O(\epsilon)\big) opt$.*

▶ **Theorem 5.** *Let (kmed-P) denote the k-median LP: $\min\big\{\sum_{j,i} c_{ij}x_{ij} : (1)\text{–}(3)\big\}$. Let $\mathcal{A}$ be an α-approximation algorithm for k-median whose approximation guarantee is proved relative to (kmed-P). We can obtain a solution to the ℓ-centrum problem of cost at most $2(\alpha+1)\overline{B}$. Thus, taking $\mathcal{A}$ to be the 3.25-approximation algorithm in [7], we obtain an $(8.5+\epsilon)$-approximation algorithm for the ℓ-centrum problem.*

Although Theorem 4 yields a worse approximation factor, the underlying primal-dual algorithm and analysis are quite versatile and extend easily to the setting with general weights. We prove Theorem 4 in this extended abstract. The proof of Theorem 5 can be found in Appendix A of the arXiv version [5] of this paper.

2.1 Proof of Theorem 4

As noted earlier, the proof relies on the primal-dual method. The dual of ($\mathrm{P}_{\overline{B}}$) is as follows.

$$\max \quad \sum_j \alpha_j - k \cdot \lambda \tag{$\mathrm{D}_{\overline{B}}$}$$

$$\text{s.t.} \quad \alpha_j \leq f_{\overline{B}}(c_{ij}) + \beta_{ij} \qquad \forall i,j \tag{4}$$

$$\sum_j \beta_{ij} \leq \lambda \qquad \forall i \tag{5}$$

$$\alpha, \lambda \geq 0.$$

Let $OPT := OPT_{\overline{B}}$ denote the optimal value of ($\mathrm{P}_{\overline{B}}$). We first fix λ and construct a solution that may open more than k centers but will have some near-optimality properties (see Theorem 6).

P1. Dual-ascent. Initialize $\mathcal{D}' = \mathcal{D}$, $\alpha_j = \beta_{ij} = 0$ for all $i, j \in \mathcal{D}$, $F = \emptyset$. The clients in $\mathcal{D}'$ are called *active clients*. If $\alpha_j \geq f_{\overline{B}}(c_{ij})$, we say that j reaches i. (So if $c_{ij} \leq \overline{B}/\ell$, then j reaches i from the very beginning.)

We repeat the following until all clients become inactive. Uniformly raise the α_js of all active clients, and the β_{ij}s for (i,j) such that $i \notin F$, j is active, and can reach i until one of the following events happen.

- Some client $j \in \mathcal{D}$ reaches some i (and previously could not reach i): if $i \in F$, we *freeze* j, and remove j from $\mathcal{D}'$.
- Constraint (5) becomes tight for some $i \notin F$: we add i to F; for every $j \in \mathcal{D}'$ that can reach i, we freeze j and remove j from $\mathcal{D}'$.

P2. **Pruning.** Pick a maximal subset T of F with the following property: for every $j \in \mathcal{D}$, there is at most one $i \in T$ such that $\beta_{ij} > 0$. Let $P = \{j : \exists i \in T \text{ s.t. } \beta_{ij} > 0\}$.

P3. Return T as the set of centers, and assign every j to the nearest point (in terms of c_{ij}) in T, which we denote by $i(j)$.

▶ **Theorem 6.** *The solution satisfies* $3\lambda|T| + \sum_{j\in P} f_{\overline{B}}(c_{i(j)j}) + \sum_{j\notin P} f_{3\overline{B}}(c_{i(j)j}) \le 3\sum_j \alpha_j$.

Proof. The proof resembles the analysis of the JV primal-dual algorithm for facility location, but the subtlety is that we need to deal with the complication that the $\{f_{\overline{B}}(c_{ij})\}_{i,j\in\mathcal{D}}$ "distances" do not form a metric.

Observe that for every $i \in T$, every client $j \in P$ for which $\beta_{ij} > 0$ satisfies $i(j) = i$. So

$$\sum_{j\in P} 3\alpha_j \ge \sum_{j\in P}\Big(3\beta_{i(j)j} + f_{\overline{B}}(c_{i(j)j})\Big) = 3\lambda|T| + \sum_{j\in P} f_{\overline{B}}(c_{i(j)j}).$$

We show that for each client $j \notin P$, there is some $i'' \in T$ such that $f_{3\overline{B}}(c_{i''j}) \le 3\alpha_j$, which will complete the proof. Let $i \in F$ be the facility that caused j to freeze, so $f_{\overline{B}}(c_{ij}) \le \alpha_j$. If $i \in T$, then we are done. Otherwise, since T is maximal, there is some $i' \in T$ and some client $k \in P$ such that $\beta_{i'k}, \beta_{ik} > 0$. Notice that $\alpha_j \ge \alpha_k$, since α_j grows at least until the time point when i joins F, and α_k grows until at most this time point. Therefore, $f_{\overline{B}}(c_{ik}), f_{\overline{B}}(c_{i'k}) \le \alpha_k \le \alpha_j$. We have $c_{i'j} \le c_{i'k} + c_{ik} + c_{ij}$. Now, by Claim 3, we have $f_{3\overline{B}}(c_{i'j}) \le f_{3\overline{B}}(c_{i'k} + c_{ik} + c_{ij}) = 3f_{\overline{B}}((c_{i'k} + c_{ik} + c_{ij})/3) \le 3\max(f_{\overline{B}}(c_{ik}), f_{\overline{B}}(c_{i'k}), f_{\overline{B}}(c_{ij})) \le 3\alpha_j$. ◀

Using standard arguments, by performing binary search on λ, we can achieve one of the following two outcomes.

(a) Obtain some λ such that the above algorithm returns a solution T with $|T| = k$: in this case, Theorem 6 implies that $\sum_j f_{3\overline{B}}(c_{i(j)j}) \le 3OPT$, and Claim 2 then implies that the $cost(\ell;.)$-cost of our solution is at most $3OPT + 3\overline{B} \le 6\overline{B}$.

(b) Obtain $\lambda_1 < \lambda_2$ with $\lambda_2 - \lambda_1 < \frac{\epsilon\overline{B}}{n}$ such that letting T_1 and T_2 be the solutions returned for λ_1 and λ_2, we have $k_1 := |T_1| > k > k_2 := |T_2|$. We describe below the procedure for extracting a low-cost feasible solution from T_1 and T_2, and analyze it, which will complete the proof of Theorem 4.

Extracting a feasible solution from T_1 and T_2 in outcome (b). Let $a, b \ge 0$ be such that $ak_1 + bk_2 = k$, $a + b = 1$. Thus, a convex combination of T_1 and T_2, called a *bipoint solution*, yields a feasible fractional solution and our task is to round this into a feasible solution. Let (α_1, β_1), (α_2, β_2) denote the dual solutions obtained for λ_1 and λ_2 respectively. Let $i_1(j)$ and $i_2(j)$ denote the centers to which j is assigned in T_1 and T_2 respectively. Let $d_{1,j} = f_{3\overline{B}}(c_{i_1(j)j})$ and $d_{2,j} = f_{3\overline{B}}(c_{i_2(j)j})$. Let $C_1 := \sum_j d_{1,j}$ and $C_2 := \sum_j d_{2,j}$. Then,

$$\begin{aligned} aC_1 + bC_2 &\le 3a\Big(\sum_j \alpha_{1,j} - k_1\lambda_1\Big) + 3b\Big(\sum_j \alpha_{2,j} - k_2\lambda_2\Big) \\ &\le 3a\Big(\sum_j \alpha_{1,j} - k\lambda_2\Big) + 3b\Big(\sum_j \alpha_{2,j} - k\lambda_2\Big) + 3ak_1(\lambda_2 - \lambda_1) \le 3OPT + 3\epsilon\overline{B} \end{aligned}$$

where the last inequality follows since $(\alpha_1, \beta_1, \lambda_2)$, $(\alpha_2, \beta_2, \lambda_2)$ are feasible solutions to $(D_{\overline{B}})$. If $b \ge 0.5$, then T_2 yields a feasible solution of $cost(\ell;.)$-cost at most $C_2 + 3\overline{B} \le 6OPT + (3+\epsilon)\overline{B}$. So suppose $a \ge 0.5$. The procedure for rounding the bipoint solution is as follows.

B1. Clustering. We first match facilities in T_2 with a subset of facilities in T_1 as follows. Initialize $\mathcal{D}' \leftarrow \mathcal{D}$, $A \leftarrow \emptyset$, and $M \leftarrow \emptyset$. While $\mathcal{D}' \neq \emptyset$, we repeatedly pick the client $j \in \mathcal{D}'$ with minimum $d_{1,j}+d_{2,j}$ value, and add j to A. We add the tuple $(i_1(j), i_2(j))$ to M, remove from $\mathcal{D}'$ all clients k (including j) such that $i_1(k) = i_1(j)$ or $i_2(k) = i_2(j)$, and set $\sigma(k) = j$ for all such clients. Let $M_1 = M$ denote the matching when $\mathcal{D}' = \emptyset$. Next, for each unmatched $i \in T_2$, we pick an arbitrary unmatched facility $i' \in T_1$, and add (i', i) to M. Let F_1 be the set of T_1-facilities that are matched, and $S := \{j \in \mathcal{D} : i_1(j) \in F_1\}$. Note that $|F_1| = |M| = k_2$.

B2. Opening facilities. We will open k_2 facilities at locations in $A \cup M$, and $k - k_2$ facilities from $T_1 \setminus F_1$. We solve the following LP to determine how to do this. Variables z_i for every $i \in T_1 \setminus F_1$ indicate if we open facility i; variable θ indicates if we give preference to F_1 (i.e., the T_1-facilities in M), or the facilities in T_2 (which are always matched).

$$\min \quad \sum_{j\in S}\big(\theta d_{1,j}+(1-\theta)d_{2,j}\big)+\sum_{k\notin S}\big(z_{i_1(k)}d_{1,k}+(1-z_{i_1(k)})(d_{2,k}+d_{1,\sigma(k)}+d_{2,\sigma(k)})\big) \quad \text{(R-P)}$$

$$\text{s.t.} \quad \sum_{i\in T_1\setminus F_1} z_i \le k-k_2, \qquad \theta \in [0,1], \quad z_i \in [0,1] \ \ \forall i \in T_1 \setminus F_1.$$

The above LP is integral. Given an integral optimal solution $(\tilde{\theta}, \tilde{z})$ to (R-P), we open facilities as follows. We open the facilities in $T_1 \setminus F_1$ specified by the $\tilde{z}_i$ variables that are 1. If $\tilde{\theta} = 1$, we open all the T_1-facilities in $M \setminus M_1$, and if $\tilde{\theta} = 0$, we open all the T_2-facilities in $M \setminus M_1$. For some clients $j \in A$, we may open a facility at j (instead of at $i_1(j)$ or $i_2(j)$). For every $j \in A$, if $\tilde{\theta}d_{1,j} + (1-\tilde{\theta})d_{2,j} = 0$, then we open a facility at j; otherwise, we open a facility at $i_1(j)$ if $\tilde{\theta} = 1$ and at $i_2(j)$ if $\tilde{\theta} = 0$.

Analysis. It suffices to show that (R-P) has a fractional solution of small objective value, and that the integral optimal solution $(\tilde{\theta}, \tilde{z})$ yields a feasible solution to our problem whose $cost(\ell; .)$-cost is comparable to the objective value of $(\tilde{\theta}, \tilde{z})$ in (R-P).

For the former, we argue that setting $\theta = a$, $z_i = a$ for all $i \in T_1 \setminus F_1$ yields a feasible solution of objective value at most $2(aC_1 + bC_2)$. We have $\sum_{i\in T_1\setminus F_1} z_i = a(k_1 - k_2) = k - k_2$. Every $j \in S$ contributes $ad_{1,j} + bd_{2,j}$ to the objective value of (R-P), which is also its contribution to $aC_1 + bC_2$. Consider $k \notin S$ with $\sigma(k) = j$, so $d_{1,j} + d_{2,j} \le d_{1,k} + d_{2,k}$. Its contribution to the objective value of (R-P) is $ad_{1,k}+b(d_{2,k}+d_{1,j}+d_{2,j}) \le (a+b)d_{1,k}+2bd_{2,k}$, which is at most twice its contribution to $aC_1 + bC_2$.

For the latter, we first show that every $k \in S$ has assignment cost at most $\tilde{\theta}d_{1,k} + (1 - \tilde{\theta})d_{2,k} + 6\overline{B}/\ell$. If a facility is opened in $\{k, i_1(k), i_2(k)\}$, then this clearly holds. Otherwise, it must be that $k \notin A$. Let $i = i_1(k)$ if $\tilde{\theta} = 1$, and $i_2(k)$ if $\tilde{\theta} = 0$. Since i is not open, it must be that i belongs to a tuple $(i_1(j), i_2(j))$ of M. Then, $j \in A$, and a facility is opened at j. we have that $c_{i,k} \le \tilde{\theta}d_{1,k} + (1 - \tilde{\theta})d_{2,k} + 3\overline{B}/\ell$ and $c_{i,j} \le 3\overline{B}/\ell$. The last inequality follows since the fact that none of $i_1(j)$, $i_2(j)$ is open implies that $\tilde{\theta}d_{1,j} + (1 - \tilde{\theta})d_{2,j} = 0$.

Now consider $k \notin S$ with $\sigma(k) = j$. If $\tilde{z}_{i_1(k)} = 1$, it's assignment cost is at most $d_{1,k} + 3\overline{B}/\ell$. Otherwise, a facility is opened in $\{j, i_1(j), i_2(j)\}$. If a facility is opened in $\{j, i_2(j)\}$, then k's assignment cost is at most $c_{i_2(k)k} + c_{i_2(j)j} \le d_{2,k} + d_{1,j} + d_{2,j} + 6\overline{B}/\ell$. Otherwise, it must be that $\tilde{\theta} = 1$ and $d_{1,j} = c_{i_1(j)j} > 3\overline{B}/\ell$; in this case, k' assignment cost is at most $c_{i_2(k)k} + c_{i_2(j)j} + c_{i_1(j)j} \le (d_{2,k} + 3\overline{B}/\ell) + (d_{2,j} + 3\overline{B}/\ell) + d_{1,j}$. Thus, the $cost(\ell; .)$-cost of our solution is at most the objective value of $(\tilde{\theta}, \tilde{z}) + 6\overline{B}$, which is at most $2(aC_1 + bC_2) + 6\overline{B} \le 6\,OPT + (6 + 3\epsilon)\overline{B} \le \big(12 + O(\epsilon)\big)\overline{B}$. This completes the proof.

3 The general weighted case

We now consider the general setting, where we have $n = |\mathcal{D}|$ non-increasing nonnegative weights $w_1 \geq \ldots \geq w_n \geq 0$, and the goal is to open k centers from $\mathcal{D}$ and assign each client $j \in \mathcal{D}$ to a center $i(j) \in F$, so as to minimize $cost\big(w; \vec{c} := \{c_{i(j)j}\}_{j\in\mathcal{D}}\big) := w^T\vec{c}^{\downarrow} = \sum_{j=1}^{n} w_j\vec{c}_j^{\downarrow}$.

By combining the ideas in Section 2 with an enumeration procedure due to Aouad and Segev in [2], we obtain the following result.

▶ **Theorem 7.** *We can obtain an $\big(18 + O(\epsilon)\big)$-approximation algorithm for ordered k-median that runs in time $\text{poly}\big((\frac{n}{\epsilon})^{1/\epsilon}\big)$.*

As before, we define suitable proxy costs analogous to the $f_B(c_{ij})$s for the setting with general weights. By defining these appropriately, it will be easy to argue that the primal-dual algorithm and its analysis extend to the setting with general weights, since essentially the only property that we use about $\{f_B(c_{ij})\}$ costs in Section 2 is that they satisfy Claim 3. Instead of creating two distance buckets in the $\{0, 1\}$ weighted case ($c_{ij} \geq B/\ell$ and $c_{ij} < B/\ell$), with weight multipliers 1 and 0, we now create $O\big(\log_{1+\epsilon}(\frac{n}{\epsilon})\big)$ buckets and utilize an enumeration idea due to Aouad and Segev [2]. In Section 3.1, we describe this enumeration procedure using our notation, and restate the main claims in [2] in a simplified form. Next, in Section 3.2, we discuss how to adapt the ideas in Section 2 to the k-median problem for the proxy costs (given by (7)) that we obtain from Section 3.1. At the end of this section, we combine this ingredients to prove Theorem 7.

3.1 Proxy costs and the enumeration idea of [2]

Throughout, let $\vec{o}^{\downarrow}$ denote the assignment-cost vector corresponding to an optimal solution, whose coordinates are sorted in non-increasing order. So the optimal cost opt is $\sum_{i=1}^{n} w_i\vec{o}_i^{\downarrow}$. By a standard argument, we can perturb w to eliminate very small weights w_i: for $i \in [n]$, set $\widetilde{w}_i = w_i$ if $w_i \geq \frac{\epsilon w_1}{n}$, and $\widetilde{w}_i = 0$ otherwise.

▶ **Claim 8.** *For any vector $v \in \mathbb{R}^n_+$, we have $(1-\epsilon)cost(w; v) \leq cost(\widetilde{w}; v) \leq cost(w; v)$.*

Proof. Since $\widetilde{w}_i \leq w_i$ for all $i \in [n]$, the upper bound on $cost(\widetilde{w}; v)$ is immediate. We have

$$cost(\widetilde{w}; v) = \sum_{i=1}^{n} \widetilde{w}_i v_i^{\downarrow} = cost(w; v) - \sum_{i\in[n]:w_i<\epsilon w_1/n} w_i v_i^{\downarrow} \geq cost(w; v) - \tfrac{\epsilon w_1}{n} \cdot n v_1^{\downarrow}. \qquad \blacktriangleleft$$

In the sequel, we always work with the $\widetilde{w}$-weights. We guess an estimate M of $\vec{o}_1^{\downarrow}$, and group distances in the range $\big[\frac{\epsilon M}{n}, M\big]$ (roughly speaking) by powers of $(1+\epsilon)$. Let T be the largest integer such that $\frac{\epsilon M}{n}(1+\epsilon)^T \leq M$. For $r = 0, \ldots, T$, we define the distance interval $I_r := \big[\frac{\epsilon M}{n}(1+\epsilon)^{T-r}, \frac{\epsilon M}{n}(1+\epsilon)^{T-r+1}\big)$. There are at most $1 + \log_{1+\epsilon}\big(\frac{n}{\epsilon}\big) = O\big(\frac{1}{\epsilon}\log\frac{n}{\epsilon}\big)$ intervals.

Finally, we guess a non-increasing vector $w_0^{\mathsf{est}} \geq w_1^{\mathsf{est}} \geq \ldots \geq w_T^{\mathsf{est}}$, where the w_r^{est}s are powers of $(1+\epsilon)$ in the range $[\frac{\widetilde{\epsilon w_1}}{n}, \widetilde{w}_1(1+\epsilon))$. As argued in [2], there are only $\exp\big(O(\frac{1}{\epsilon}\log(\frac{n}{\epsilon}))\big) = O\big((\frac{n}{\epsilon})^{1/\epsilon}\big)$ choices for $w^{\mathsf{est}} := (w_0^{\mathsf{est}}, \ldots, w_T^{\mathsf{est}})$. The intention is for w_r^{est} to represent (within a $(1+\epsilon)$-factor) the average $\widetilde{w}$-weight of the set $\{i \in [n] : \vec{o}_i^{\downarrow} \in I_r\}$. More precisely, we would like w_r^{est} to estimate the following quantity, for all $r \in \{0, \ldots, T\}$.

$$w_r^{\mathsf{avg}} := \begin{cases} \big(\sum_{i\in[n]:\vec{o}_i^{\downarrow}\in I_r} \widetilde{w}_i\big)/|\{i \in [n] : \vec{o}_i^{\downarrow} \in I_r\}| & \text{if } \{i \in [n] : \vec{o}_i^{\downarrow} \in I_r\} \neq \emptyset; \\ \min\{\widetilde{w}_i : \vec{o}_i^{\downarrow} \in \bigcup_{s<r} I_s\} & \text{if } \bigcup_{s<r} I_s \neq \emptyset; \\ \widetilde{w}_1 & \text{otherwise.} \end{cases} \tag{6}$$

The following claim will be useful.

▶ **Claim 9.** *For any $r \in \{0,\ldots,T\}$, we have $w_r^{\mathsf{avg}} \geq \max\{\widetilde{w}_i : \vec{o}_i^{\downarrow} \notin \bigcup_{s\leq r} I_s\}$.*

Proof. If w_r^{avg} is defined by cases 1 or 2 of (6), then the inequality follows since for every $i' \in \bigcup_{s\leq r} I_r$ and $i \notin \bigcup_{s\leq r} I_s$, we have $\widetilde{w}_{i'} \geq \widetilde{w}_i$ (since $\vec{o}_{i'}^{\downarrow} \geq \vec{o}_i^{\downarrow}$). If w_r^{avg} is defined by case 3 of (6), then $w_r^{\mathsf{avg}} = \widetilde{w}_1$, and again, the inequality holds. ◀

Given M and the corresponding intervals $I_0,\ldots,I_T$, and the vector w^{est}, we can now finally define our proxy costs as follows. For $d \geq 0$ and $\gamma \geq 1$, define

$$g_{M,w^{\mathsf{est}}}(\gamma;d) = \begin{cases} \widetilde{w}_1(1+\epsilon)d & \text{if } d/\gamma \geq \frac{\epsilon M}{n}(1+\epsilon)^{T+1}; \\ w_r^{\mathsf{est}} d & \text{if } d/\gamma \in I_r \text{ (where } r \in \{0,\ldots,T\}) \\ 0 & \text{if } d/\gamma < \frac{\epsilon M}{n}. \end{cases} \tag{7}$$

The above definition is essentially the scaled surrogate function in [2]. We abbreviate $g_{M,w^{\mathsf{est}}}(1;d)$ to $g_{M,w^{\mathsf{est}}}(d)$. The following two key lemmas are analogous to Claims 1 and 2, and show that for the right choice of M and w^{est}, evaluating the above proxy costs on an assignment-cost vector $\vec{c}$ yields a good estimate of the actual $cost(\widetilde{w};.)$-cost of $\vec{c}$. Similar statements, albeit stated somewhat differently, are proved in [2].

▶ **Lemma 10** (adapted from [2]). *Suppose $M \geq \vec{o}_1^{\downarrow}$ and the w^{est} satisfies $w_r^{\mathsf{est}} \leq (1+\epsilon)w_r^{\mathsf{avg}}$ for all $r \in \{0,\ldots,T\}$. Then, $\sum_{i=1}^n g_{M,w^{\mathsf{est}}}(\vec{o}_i^{\downarrow}) \leq (1+\epsilon)^2 cost(\widetilde{w};\vec{o}^{\downarrow})$.*

Proof. Since $\frac{\epsilon M}{n}(1+\epsilon)^{T+1} > M \geq \vec{o}_1^{\downarrow}$, there is no i such that $\vec{o}_i^{\downarrow} \geq \frac{\epsilon M}{n}(1+\epsilon)^{T+1}$. Fix $r \in \{0,\ldots,T\}$, and consider all $i \in [n]$ such that $\vec{o}_i^{\downarrow} \in I_r$. We have

$$\begin{aligned} \sum_{i\in[n]:\vec{o}_i^{\downarrow}\in I_r} g_{M,w^{\mathsf{est}}}(\vec{o}_i^{\downarrow}) = w_r^{\mathsf{est}} \sum_{i\in[n]:\vec{o}_i^{\downarrow}\in I_r} \vec{o}_i^{\downarrow} &\leq \frac{\epsilon M}{n}(1+\epsilon)^{T-r+1} \cdot w_r^{\mathsf{est}} \cdot \left|\{i\in[n] : \vec{o}_i^{\downarrow} \in I_r\}\right| \\ &\leq (1+\epsilon)\cdot\frac{\epsilon M}{n}(1+\epsilon)^{T-r+1} \cdot w_r^{\mathsf{avg}} \cdot \left|\{i\in[n] : \vec{o}_i^{\downarrow} \in I_r\}\right| \\ &= (1+\epsilon)\cdot\frac{\epsilon M}{n}(1+\epsilon)^{T-r+1} \cdot \sum_{i\in[n]:\vec{o}_i^{\downarrow}\in I_r} \widetilde{w}_i \leq (1+\epsilon)^2 \sum_{i\in[n]:\vec{o}_i^{\downarrow}\in I_r} \widetilde{w}_i\vec{o}_i^{\downarrow}. \end{aligned}$$

It follows that $\sum_{i=1}^n g_{M,w^{\mathsf{est}}}(\vec{o}_i^{\downarrow}) \leq (1+\epsilon)^2 cost(\widetilde{w};\vec{o}^{\downarrow})$. ◀

▶ **Lemma 11** (adapted from [2]). *Let $\gamma \geq 1$. Let $M \geq 0$, and suppose w^{est} satisfies $w_r^{\mathsf{avg}} \leq w_r^{\mathsf{est}}$ for all $r \in \{0,\ldots,T\}$. Let $\vec{c}$ be an assignment-cost vector. Then, we have the upper bound $cost(\widetilde{w};\vec{c}) \leq \sum_{i=1}^n g_{M,w^{\mathsf{est}}}(\gamma;\vec{c}_i) + \gamma(1+\epsilon)cost(\widetilde{w};\vec{o}^{\downarrow}) + \gamma\epsilon\widetilde{w}_1 M$.*

Proof. We have $cost(\widetilde{w};\vec{c}) = \sum_{i=1}^n \widetilde{w}_i\vec{c}_i^{\downarrow} \leq \sum_{i=1}^n g_{M,w^{\mathsf{est}}}(\gamma;\vec{c}_i) + \sum_{i:\ \widetilde{w}_i\vec{c}_i^{\downarrow} > g_{M,w^{\mathsf{est}}}(\gamma;\vec{c}_i^{\downarrow})} \widetilde{w}_i\vec{c}_i^{\downarrow}$. Consider some $i \in [n]$ for which $\widetilde{w}_i\vec{c}_i^{\downarrow} > g_{M,w^{\mathsf{est}}}(\gamma;\vec{c}_i^{\downarrow})$. It must be that $\vec{c}_i^{\downarrow}/\gamma < \frac{\epsilon M}{n}(1+\epsilon)^{T+1}$ as otherwise (see (7)), we have $g_{M,w^{\mathsf{est}}}(\gamma;\vec{c}_i^{\downarrow}) = (1+\epsilon)\widetilde{w}_1\vec{c}_i^{\downarrow} > \widetilde{w}_i\vec{c}_i^{\downarrow}$. If $g_{M,w^{\mathsf{est}}}(\gamma;\vec{c}_i^{\downarrow}) = 0$, then we have $\widetilde{w}_i\vec{c}_i^{\downarrow}/\gamma < \widetilde{w}_i \cdot \frac{\epsilon M}{n} \leq \widetilde{w}_1 \cdot \frac{\epsilon M}{n}$.

Otherwise, we claim that $\vec{c}_i^{\downarrow}/\gamma \leq (1+\epsilon)\vec{o}_i^{\downarrow}$. Suppose not. Suppose $\vec{c}_i^{\downarrow}/\gamma \in I_r$, where $r \in \{0,\ldots,T\}$. Since $\frac{\vec{c}_i^{\downarrow}/\gamma}{\vec{o}_i^{\downarrow}} > (1+\epsilon)$, we have that $\vec{o}_i^{\downarrow} \notin \bigcup_{s\leq r} I_s$. So by Claim 9, we have $w_r^{\mathsf{avg}} \geq \widetilde{w}_i$. Hence, $g_{M,w^{\mathsf{est}}}(\gamma;\vec{c}_i^{\downarrow}) = w_r^{\mathsf{est}}\vec{c}_i^{\downarrow} \geq w_r^{\mathsf{avg}}\vec{c}_i^{\downarrow} \geq \widetilde{w}_i\vec{c}_i^{\downarrow}$, which contradicts our assumption that $\widetilde{w}_i\vec{c}_i^{\downarrow} > g_{M,w^{\mathsf{est}}}(\gamma;\vec{c}_i^{\downarrow})$.

Putting everything together, we have that $\sum_{i:\widetilde{w}_i\vec{c}_i^{\downarrow} > g_{M,w^{\mathsf{est}}}(\gamma;\vec{c}_i^{\downarrow})} \widetilde{w}_i\vec{c}_i^{\downarrow} \leq n\gamma\widetilde{w}_1 \cdot \frac{\epsilon M}{n} + \gamma(1+\epsilon)\sum_{i\in[n]} \widetilde{w}_i\vec{o}_i^{\downarrow}$, which proves the lemma. ◀

Finally, we show that $g_{M,w^{\text{est}}}$ satisfies the analogue of Claim 3, which will be crucial in arguing that our algorithms and analysis from Section 4 carry over and allow us to solve, in an approximate sense, the k-median problem with the $\{g_{M,w^{\text{est}}}(c_{ij})\}$ proxy costs.

▶ **Lemma 12.** *For any $\gamma \geq 1$, $M \geq 0$, and w^{est}, we have: (i) $g_{M,w^{\text{est}}}(\gamma; x) \leq g_{M,w^{\text{est}}}(\gamma; y)$ if $x \leq y$; and (ii) $3\max\{g_{M,w^{\text{est}}}(\gamma; x), g_{M,w^{\text{est}}}(\gamma; y), g_{M,w^{\text{est}}}(\gamma; z)\} \geq g_{M,w^{\text{est}}}(3\gamma; x+y+z)$ for any $x, y, z \geq 0$.*

3.2 Solving the k-median problem with the $\left\{g_{M,w^{\text{est}}}(c_{ij})\right\}$ proxy costs

We now work with a fixed guess M, w^{est}, and give an algorithm for finding a near-optimal k-median solution with the $\{g_{M,w^{\text{est}}}(c_{ij})\}$ proxy costs. Our algorithm and analysis will be quite similar to the one in Section 4. The primal and dual LPs we consider are the same as (P$_B$) and (D$_{\overline{B}}$), except that we replace all occurrences of $f_B(c_{ij})$ and $f_{\overline{B}}(c_{ij})$ with $g_{M,w^{\text{est}}}(c_{ij})$. Let $OPT_{M,w^{\text{est}}}$ denote the optimal value of this LP.

The primal-dual algorithm for a given center-cost λ (steps P1–P3 in Section 4) is unchanged. The analysis also is essentially identical, since, previously, we only relied on the fact that the proxy costs satisfy an approximate triangle inequality, which is also true here (Lemma 12). We state below the guarantee from the primal-dual algorithm slightly differently, in the form suggested by part (ii) of Lemma 12; the proof mimics the proof of Theorem 6.

▶ **Theorem 13.** *For any $\lambda \geq 0$, the primal-dual algorithm (P1)–(P3) returns a set T of centers, an assignment $i(j) \in T$ for every $j \in \mathcal{D}$, and a dual feasible solution (α, β, λ) such that $3\lambda|T| + \sum_j g_{M,w^{\text{est}}}(3; c_{i(j)j}) \leq 3\sum_j \alpha_j$.*

Given Theorem 13, we can use binary search on λ, to either obtain: (a) some λ such for which we return a solution T with $|T| = k$; or (b) $\lambda_1 < \lambda_2$ with $\lambda_2 - \lambda_1 < \frac{\epsilon \widetilde{w}_1 M}{n}$ such that letting T_1 and T_2 be the solutions returned for λ_1 and λ_2, we have $k_1 := |T_1| > k > k_2 := |T_2|$. In case (a), Theorem 13 implies that $\sum_j g_{M,w^{\text{est}}}(3; c_{i(j)j}) \leq 3OPT_{M,w^{\text{est}}}$. In case (b), we again extract a low-cost feasible solution from T_1 and T_2 by rounding the bipoint solution given by their convex combination. As before, $a, b \geq 0$ be such that $ak_1 + bk_2 = k$, $a + b = 1$. Let (α_1, β_1), (α_2, β_2) denote the dual solutions obtained for λ_1 and λ_2 respectively. Let $i_1(j)$ and $i_2(j)$ denote the centers to which j is assigned in T_1 and T_2 respectively. Let $d_{1,j} = g_{M,w^{\text{est}}}(3; c_{i_1(j)j})$ and $d_{2,j} = g_{M,w^{\text{est}}}(3; c_{i_2(j)j})$. Let $C_1 := \sum_j d_{1,j}$ and $C_2 := \sum_j d_{2,j}$. Similar to before, we have $aC_1 + bC_2 \leq 3OPT_{M,w^{\text{est}}} + 3\epsilon\widetilde{w}_1 M$. The procedure for rounding this bipoint solution requires only minor changes to steps B1, B2 in Section 4.

Rounding the bipoint solution. If $b \geq 1/3$, then T_2 yields a feasible solution with $\sum_j g_{M,w^{\text{est}}}(3; c_{i_2(j)j}) = C_2 \leq 9OPT_{M,w^{\text{est}}} + 9\epsilon\widetilde{w}_1 M$. So suppose $a \geq 2/3$.

G1. Clustering. We match facilities in T_2 with a subset of facilities in T_1 as follows. Initialize $\mathcal{D}' \leftarrow \mathcal{D}$, $A \leftarrow \emptyset$, and $M \leftarrow \emptyset$. We repeatedly pick the client $j \in \mathcal{D}'$ with minimum $\max\{d_{1,j}, d_{2,j}\}$ value, and add j to A. (***This is the only change, compared to step B1.***) We add the tuple $(i_1(j), i_2(j))$ to M, remove from $\mathcal{D}'$ all clients k (including j) such that $i_1(k) = i_1(j)$ or $i_2(k) = i_2(j)$, and set $\sigma(k) = j$ for all such clients. Let $M_1 = M$ denote the matching when $\mathcal{D}' = \emptyset$. Next, for each unmatched $i \in T_2$, we pick an arbitrary unmatched facility $i' \in T_1$, and add (i', i) to M. Let F_1 be the set of T_1-facilities that are matched, and $S := \{j \in \mathcal{D} : i_1(j) \in F_1\}$. Note that $|F_1| = |M| = k_2$.

G2. Opening facilities. This is almost identical to step B2, except that we decide which facilities to open by now solving the following LP.

$$\min \quad \sum_{j\in S}\big(\theta d_{1,j}+(1-\theta)d_{2,j}\big)+\sum_{k\notin S}\big(z_{i_1(k)}d_{1,k}+(1-z_{i_1(k)})\cdot 3\max\{d_{1,k},d_{2,k}\}\big) \quad \text{(GR-P)}$$
$$\text{s.t.} \quad \sum_{i\in T_1\setminus F} z_i \le k-k_2, \qquad \theta\in[0,1], \quad z_i\in[0,1] \ \ \forall i\in T_1\setminus F.$$

Let $(\tilde\theta,\tilde z)$ be an optimal integral solution to (GR-P). If $\tilde\theta=1$, we open all facilities in F_1, and otherwise, all facilities in T_2. We also open the facilities from $T_1\setminus F_1$ for which $\tilde z_i=1$.

To analyze this, we first show that setting $\theta=a$, $z_i=a$ for all $i\in T_1\setminus F_1$ yields a feasible solution to (GR-P) of objective value at most $3(aC_1+bC_2)$. We have $\sum_{i\in T_1\setminus F_1} z_i = a(k_1-k_2)=k-k_2$. Every $j\in S$ contributes $ad_{1,j}+bd_{2,j}$ to the objective value of (GR-P). Consider $k\notin S$. Its contribution to the objective value of (GR-P) is

$$ad_{1,k}+3b\max\{d_{1,k},d_{2,k}\}=\max\{(a+3b)d_{1,k},ad_{1,k}+3bd_{2,k}\}\le 3(ad_{1,k}+bd_{2,k})$$

where the inequality follows since $a+3b\le 3a$ when $a\ge 2/3$. Thus, for every $j\in\mathcal{D}$, its contribution to the objective value of (GR-P) is at most thrice its contribution to aC_1+bC_2.

Suppose $\vec c$ is the assignment-cost vector resulting from $(\tilde\theta,\tilde z)$. We show that $\sum_j g_{M,w^{\mathsf{est}}}(9;\vec c_j)$ is at most the objective value of $(\tilde\theta,\tilde z)$ under (GR-P). For every $k\in S$, we have $g_{M,w^{\mathsf{est}}}(9;\vec c_k)\le g_{M,w^{\mathsf{est}}}(3;\vec c_k)\le \tilde\theta d_{1,k}+(1-\tilde\theta)d_{2,k}$. Now consider $k\notin S$ with $\sigma(k)=j$, so $\max\{d_{1,j},d_{2,j}\}\le\max\{d_{1,k},d_{2,k}\}$. If $\tilde z_{i_1(k)}=1$, then $g_{M,w^{\mathsf{est}}}(9;\vec c_k)\le g_{M,w^{\mathsf{est}}}(3;\vec c_k)\le d_{1,k}$. Otherwise, $\vec c_k\le c_{i_2(k)k}+c_{i_1(j)j}+c_{i_2(j)j}$, and so by Lemma 12, we have

$$\begin{aligned} g_{M,w^{\mathsf{est}}}(9;\vec c_k) &\le g_{M,w^{\mathsf{est}}}(9;c_{i_2(k)k}+c_{i_1(j)j}+c_{i_2(j)j})\\ &\le 3\max\{g_{M,w^{\mathsf{est}}}(3;c_{i_2(k)}),g_{M,w^{\mathsf{est}}}(3;c_{i_1(j)j}),g_{M,w^{\mathsf{est}}}(3;c_{i_2(j)j})\}\le 3\max\{d_{1,k},d_{2,k}\}.\end{aligned}$$

So in every case, $g_{M,w^{\mathsf{est}}}(9;\vec c_k)$ is bounded by the contribution of k to the objective value of $(\tilde\theta,\tilde z)$. Thus, we have proved the following theorem.

▶ **Theorem 14.** *For any $M\ge 0$, w^{est}, we can obtain a solution opening k centers whose assignment-cost vector $\vec c$ satisfies $\sum_j g_{M,w^{\mathsf{est}}}(9;\vec c_j)\le 9OPT_{M,w^{\mathsf{est}}}+9\epsilon\widetilde{w}_1 M$.*

Proof of Theorem 7. The proof follows by combining Theorem 14, Lemmas 10 and 11, and Claim 8. Recall that $\vec o^{\downarrow}$ is the assignment-cost vector corresponding to an optimal solution with coordinates sorted in non-increasing order, and $opt=\sum_{i=1}^n w_i\vec o_i^{\downarrow}$ is the optimal cost.

There are only n^2 choices for M, and $O\big((\frac{n}{\epsilon})^{1/\epsilon}\big)$ choices for w^{est}, so we may assume that in polynomial time, we have obtained $M=\vec o_1^{\downarrow}$, and w_r^{est}s satisfying $w_r^{\mathsf{avg}}\le w_r^{\mathsf{est}}\le(1+\epsilon)w_r^{\mathsf{avg}}$ for all $r\in\{0,\dots,T\}$. By Lemma 10, we know that $OPT_{M,w^{\mathsf{est}}}\le(1+\epsilon)^2 cost(\widetilde w;\vec o^{\downarrow})\le(1+\epsilon)^2 opt$. Let $\vec c$ be the assignment-cost vector of the solution returned by Theorem 14 for this M, w^{est}. Combining Theorem 14, Lemma 11, and Claim 8, we obtain that

$$\begin{aligned}(1-\epsilon)cost(w;\vec c)\le cost(\widetilde w;\vec c)&\le\big(9OPT_{M,w^{\mathsf{est}}}+9\epsilon\widetilde w_1 M\big)+9(1+\epsilon)cost(\widetilde w;\vec o^{\downarrow})+9\epsilon\widetilde w_1 M\\ &\le 9(1+\epsilon)^2 opt+9opt+O(\epsilon)opt=\big(18+O(\epsilon)\big)opt.\end{aligned}$$

4 Conclusions and discussion

We have described algorithms achieving approximation guarantees of $12+\epsilon$ and $18+\epsilon$ for the ℓ-centrum and ordered k-median problems. Our algorithms are combinatorial, utilizing

the primal-dual schema and Lagrangian relaxation, and improve upon the algorithms in [3], both in terms of approximation factors and simplicity of analysis.

One interesting research direction suggested by our work is to investigate the ordered-median and ℓ-centrum (i.e., ordered median with $\{0,1\}$-weights) versions of other optimization problems. In further work, we have been able to develop a general framework for devising algorithms for ordered-median problems. Our framework also yields improved guarantees for the ℓ-centrum and ordered k-median problems studied here. We obtain analogous improvements for ordered k-median. We defer details to a forthcoming manuscript.

References

1. S. Alamdari and D. Shmoys. A bicriteria approximation algorithm for the k-center and k-median problems. In *Proceedings of WAOA*, 2017.
2. A. Aouad and D. Segev. The ordered k-median problem: surrogate models and approximation algorithms. In submission. Available at `https://pdfs.semanticscholar.org/39e8/391a9e918a2ba1eb21c1c2dbb52d3b1f0de1.pdf`, 2017.
3. J. Byrka, K. Sornat, and J. Spoerhase. Constant-factor approximation for ordered k-median. To appear in *Proceedings of STOC*, 2018. Available at `https://arXiv.org/abs/1711.01972`, Nov 6, 2017.
4. Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median, and positive correlation in budgeted optimization. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 737–756. SIAM, 2015.
5. D. Chakrabarty and C. Swamy. Interpolating between k-median and k-center: Approximation algorithms for ordered k-median, Nov 23, 2017. URL: `https://arxiv.org/abs/1711.08715`.
6. M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002.
7. M. Charikar and S. Li. A dependent LP-rounding approach for the k-median problem. In *Proceedings of the 39th ICALP*, 2012.
8. Zvi Drezner and Stefan Nickel. Solving the ordered one-median problem in the plane. *European Journal of Operational Research*, 195(1):46–61, 2009.
9. R. L. Francis, T. J. Lowe, and A. Tamir. Aggregation error bounds for a class of location models,. *Operations Research*, 48:294–307, 2000.
10. Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.
11. Dorit S Hochbaum and David B Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33(3):533–550, 1986.
12. J. N. Hooker, R. S. Garfinkel, and C. K. Chen. Finite dominating sets for network location problems. *Operations Research*, 39:100–118, 1991.
13. K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.
14. G. Laporte, S. Nickel, and F. S. da Gama. *Location Science*. Springer, 2015.
15. Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing*, pages 901–910. ACM, 2013.
16. Pitu B Mirchandani and Richard L Francis. *Discrete location theory*. Wiley-Interscience, 1990.

17 S. Nickel and J. Puerto. A unified approach to network location problems. *Networks*, 34:283–290, 1999.

18 S. Nickel and J. Puerto. *Location Theory: A Unified Approach*. Springer Science & Business Media, 2005.

19 Justo Puerto and Antonio M. Rodríguez-Chía. On the exponential cardinality of FDS for the ordered p-median problem. *Oper. Res. Lett.*, 33(6):641–651, 2005.

20 Justo Puerto, Antonio M. Rodríguez-Chía, and Arie Tamir. Revisiting k-sum optimization. *Math. Program.*, 165(2):579–604, 2017.

21 D. B. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th STOC*, pages 265–274, 1997.

22 David B Shmoys. The design and analysis of approximation algorithms. *Trends in Optimization: American Mathematical Society Short Course, January 5-6, 2004, Phoeniz, Arizona*, 61:85, 2004.

23 A. Tamir. The k-centrum multi-facility location problem. *Discrete Applied Mathematics*, 109(3):293–307, 2001.

Generalized Center Problems with Outliers

Deeparnab Chakrabarty
Department of Computer Science, Dartmouth College, 9 Maynard St, Hanover, NH, USA
https://web.cs.dartmouth.edu/people/deeparnab-chakrabarty
deeparnab@dartmouth.edu

Maryam Negahbani
Department of Computer Science, Dartmouth College, 9 Maynard St, Hanover, NH, USA
maryam@cs.dartmouth.edu

Abstract

We study the $\mathcal{F}$-center problem with outliers: given a metric space (X, d), a general down-closed family $\mathcal{F}$ of subsets of X, and a parameter m, we need to locate a subset $S \in \mathcal{F}$ of centers such that the maximum distance among the closest m points in X to S is minimized.

Our main result is a *dichotomy theorem*. Colloquially, we prove that there is an efficient 3-approximation for the $\mathcal{F}$-center problem with outliers if and only if we can efficiently optimize a *poly-bounded* linear function over $\mathcal{F}$ subject to a partition constraint. One concrete upshot of our result is a polynomial time 3-approximation for the knapsack center problem with outliers for which no (true) approximation algorithm was known.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Approximation Algorithms, Clustering, k-Center Problem

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.30

Related Version A full version of the paper is available at https://arxiv.org/abs/1805.02217.

1 Introduction

The k-center problem is a classic discrete optimization problem with numerous applications. Given a metric space (X, d) and a positive integer k, the objective is to choose a subset $S \subseteq X$ of at most k points such that $\max_{v \in X} d(v, S)$ is minimized, where $d(v, S) = \min_{u \in S} d(v, u)$. Informally, the problem is to open k centers to serve all points, minimizing the maximum distance to service. This problem has been studied for at least 50 years [13, 14], is NP-hard to approximate to a factor better than 2 [18], and has a simple 2-approximation algorithm [11, 16].

In many applications one is interested in a nuanced version of the problem where instead of serving all points in X, the objective is to serve at least a certain number of points. This is the so-called k-center with outliers version, or the *robust k-center* problem. This problem was first studied by Charikar *et al.* in [8] which gives a 3-approximation for the problem. A best possible 2-approximation algorithm was recently given by Chakrabarty *et al.* in [6] (see also the paper [15] by Harris *et al.*).

Another generalization of the k-center problem arises when the location of centers has more restrictions. For instance, if each point in X has a different weight and the constraint is that the total weight of centers opened is at most k. This problem, now called the *knapsack center* problem, was studied by Hochbaum and Shmoys in [17] which gives a 3-approximation for the problem. To take another instance, X could be vectors in high dimension and the


centers picked need to be linearly independent vectors. This motivates the *matroid center* problem where the set of centers must be an independent set in a matroid. Chen *et al.* give a 3-approximation for this problem in [10].

Naturally, the two aforementioned generalizations can be taken together. Indeed, for the *robust matroid center* problem, that is, the problem of picking centers which are an independent set and only m points need to be served, there is a 7-approximation algorithm in [10]. This was recently improved to a 3-approximation in [15]. The *robust knapsack center* problem, however, has had no non-trivial approximation algorithm till this work. Both [10] and [15] give *bi-criteria* 3-approximation algorithms which violate the knapsack constraint by $(1+\varepsilon)$ (the running time of their algorithm is exponential in $1/\varepsilon$).

Our Contributions

Motivated by the state-of-affairs of the robust knapsack center problem, we study a broad generalization of the problems mentioned above. Let $\mathcal{F}$ be a general down-closed[1] family of subsets over X. In the *robust $\mathcal{F}$-center* problem we are given a metric space (X, d), a parameter m, and the objective is to select a subset $S \in \mathcal{F}$ such that $\min_{T \subseteq X, |T|=m} \max_{v \in T} d(v, S)$ is minimized. That is, the maximum distance of service of the closest m points is minimized.

Observe that if $\mathcal{F} := \{A : w(A) \leq k\}$ then we get the robust knapsack center problem, and if $\mathcal{F}$ is the collection of independent sets of a matroid, then we get the robust matroid center problem. But this generalization captures a host of other problems. For instance, one can consider multiple (but constant) knapsack constraints. Indeed, this was studied in both [17] and [10]. The former[2] only looks at the version *without* outliers and gives a polynomial time 3-approximation in the case when the weights are all polynomially bounded. The latter proves that when the weights are not polynomially bounded, there can be no approximation algorithm via a reduction to the Subset Sum problem, and gives a 3-approximation violating each knapsack constraint by at most $(1+\varepsilon)$ multiplicative factor.

Another instance is a single knapsack constraint along with a single matroid constraint. To our knowledge, this problem has not been studied earlier even in the case when outliers are not allowed. This problem seems natural: for instance, when the points are high dimensional vectors with weights and the collection of centers needs to be a linearly independent set with total weight at most k.

The complexity of the robust $\mathcal{F}$-center problem naturally depends on the complexity of $\mathcal{F}$. To understand this, we define the following optimization problem which depends only on the set-system $(X, \mathcal{F})$. We call it the *$\mathcal{F}$-maximization under partition constraints* or simply $\mathcal{F}$-PCM. In this problem, one is given an arbitrary partition $\mathcal{P}$ of X along with $\mathcal{F}$, and a *poly-bounded* (the range is at most a polynomial in $|X|$) value $\mathsf{val}(x)$ on each $x \in X$. The objective is to find a set $S \in \mathcal{F}$ maximizing $\mathsf{val}(S)$ such that S contains at most one element from each part of $\mathcal{P}$. Our main result stated colloquially (and formally stated as Theorem 4 and Theorem 5 in Section 2) is the following dichotomy theorem[3].

---

[1] if $A \in \mathcal{F}$ and $B \subseteq A$, then $B \in \mathcal{F}$.

[2] The complete proofs can be found in the STOC 1984 version of [17]

[3] We are deliberately being inaccurate here. We should state the theorem for the more general *supplier* version where the set X is partitioned into $F \cup C$ and only the points in C need to be covered and only the centers in F can be opened. Being more general, the algorithmic results are therefore stronger. On the other hand, we weren't able (and didn't try too hard) to make our hardness go through for the center version. In the Introduction we stick with the center version and switch to the supplier in the more formal subsequent sections.

▶ **Informal Theorem.** For any down-closed family $(X, \mathcal{F})$, the robust $\mathcal{F}$-center problem has an efficient 3-approximation algorithm if the $\mathcal{F}$-PCM problem can be solved in polynomial time. Otherwise, there is no efficient *non-trivial* approximation algorithm for the robust $\mathcal{F}$-center problem.

Note that in general, we are not concerned about how $\mathcal{F}$ is represented, because the only place the algorithm checks if a set S is in $\mathcal{F}$ is perhaps for solving the $\mathcal{F}$-PCM problem. So one can choose a representation that works best for the $\mathcal{F}$-PCM solver.

A series of corollaries follow from the above theorem. These are summarized in Table 1.

- When $\mathcal{F} = \{A : w(A) \leq k\}$, the $\mathcal{F}$-PCM problem can be solved in polynomial time via dynamic programming. This crucially uses that the val is poly-bounded. Therefore we get a 3-approximation for the robust knapsack center problem. (Theorem 17)
- When $\mathcal{F}$ is the independent set of a matroid, then the $\mathcal{F}$-PCM problem is a matroid intersection problem. Therefore we get a 3-approximation for the robust matroid center problem recovering the result from [15]. (Theorem 19)
- When $\mathcal{F} = \{A : w_1(A) \leq k_1, w_2(A) \leq k_2, \ldots, w_d(A) \leq k_d\}$ is defined by d weight functions *and* each weight function w_i is *poly-bounded*, then $\mathcal{F}$-PCM can be solved efficiently using dynamic programming. Therefore we get a 3-approximation algorithm for the robust multi-knapsack center problem, extending the result in [17] to the case with outliers. (Theorem 18)
- When $\mathcal{F}$ is given by the intersection of a single knapsack and a single matroid constraint, then we don't know the complexity. However, when the weight function $w(\cdot)$ is poly-bounded and the matroid is representable, then we can give a *randomized* algorithm for the $\mathcal{F}$-PCM problem via a reduction to the exact matroid intersection problem. Therefore, we get a randomized 3-approximation for this special case of robust knapsack-and-matroid center problem (Theorem 21).

Remark 1: The Zero Outlier Case. At this juncture, the reader may wonder about the complexity of the $\mathcal{F}$-center problem which doesn't allow any outliers. This is related to the following decision problem. Given $(X, \mathcal{F})$ and an arbitrary sub-partition $\mathcal{P}$ of X, the problem asks whether there is a set $S \in \mathcal{F}$ such that S contains *exactly* one element from each part of $\mathcal{P}$. We call this the *$\mathcal{F}$-feasibility under partition constraints* or simply the $\mathcal{F}$-PCF problem. Analogous to the informal theorem from earlier, the $\mathcal{F}$-center problem (without outliers) has an efficient 3-approximation algorithm if the $\mathcal{F}$-PCF problem can be solved efficiently; otherwise the $\mathcal{F}$-center problem has no non-trivial approximation algorithm. Indeed, this theorem is much simpler to prove and arguably the roots of this lie in [17].

This raises the main open question from our paper: *what is the relation between the $\mathcal{F}$-PCF and the $\mathcal{F}$-PCM problem?* Clearly, the $\mathcal{F}$-PCF problem is as easy as the $\mathcal{F}$-PCM problem (set all values equal to one in the latter). But is there an $\mathcal{F}$ such that $\mathcal{F}$-PCM is "hard" while $\mathcal{F}$-PCF is "easy"? One concrete example is the corollary discussed in the last bullet point above. When $\mathcal{F}$ is a single knapsack constraint and a single matroid constraint, then the $\mathcal{F}$-PCF problem is solvable in polynomial time by minimizing a linear function over a matroid polytope and another partition matroid *base* polytope. As noted above, we don't know the complexity of the $\mathcal{F}$-PCM problem in this case.

Remark 2: Handling Approximations. If the $\mathcal{F}$-PCM problem is NP-hard, then the robust $\mathcal{F}$-center has no non-trivial approximation algorithm. However approximation algorithms for $\mathcal{F}$-PCM translate to bi-criteria approximation algorithms for the robust $\mathcal{F}$-center problem. More precisely, if we have a ρ-approximation for the $\mathcal{F}$-PCM problem ($\rho \leq 1$), then we get

Table 1 All the above results can be obtained as corollary or simple extensions to our main result. The numbers in bold indicate new results.

The constraint system $\mathcal{F}$	Without Outliers	Robust (With Outliers)
Knapsack Constraint	3 [17]	**3** (Theorem 17)
Matroid Constraint	3 [10]	3 [15]
Multiple Knapsack (poly-bounded weights)	3 [17]	**3** (Theorem 18)
Knapsack and Matroid	**3** (Theorem 24)	Open **3** in special case (Theorem 21)
Multiple Knapsacks and Matroid Constraint	No uni-criteria approximation	**3**, $(1+\varepsilon)$ violating (Theorem 27)

a $(3, \rho)$-*bi-criteria* approximation algorithm for the robust $\mathcal{F}$-center problem. That is, we return a solution $S \in \mathcal{F}$ such that the maximum distance among the closest $\rho \cdot m$ points is at most 3 times the optimum value.

There could be a different notion of approximation possible for the $\mathcal{F}$-PCM problem. Given an instance, there may be an algorithm which returns a set S whose value is at least the optimum value but $S \in \mathcal{F}^R$ for some $\mathcal{F}^R \supseteq \mathcal{F}$ which is a 'relaxation' of $\mathcal{F}$. For instance, if $\mathcal{F}$ is the intersection of multiple (constant) knapsack constraints which are not poly-bounded, then for any constant $\varepsilon > 0$ the $\mathcal{F}$-PCM problem can be solved [9, 12] returning a set with value at least the optimum but violating each constraint by multiplicative $(1+\varepsilon)$. We can use the same to get a polynomial time 3-approximation for the robust multiple knapsack-center problem if we are allowed to violate the knapsack constraints by $(1+\varepsilon)$.

Our Technique

Although our theorem statement is quite general, the proof is quite easy. Let us begin with the $\mathcal{F}$-center problem without outliers. For this, we follow the algorithmic 'partitioning' idea outlined in paper [17] by Hochbaum and Shmoys. As is standard, we guess the optimum distance which we assume to be 1 by scaling. Initially, all points are marked uncovered. Subsequently, we pick *any* uncovered point x and consider a subset B_x of points within distance 1 from it. Note that the optimum solution *must* pick at least one point from each B_x to serve x. Next, we call x "responsible" for all uncovered points within a distance 2 from it, and mark all these points covered. Observe that all the newly covered points are within distance 3 from *any* point in B_x. We continue the above procedure till all points are marked covered. Also observe that the B_x's form a sub-partition $\mathcal{P}$ of the universe where each part has a responsible point. By the above two observations, we see that the $\mathcal{F}$-PCF problem must have a feasible solution with respect to $\mathcal{P}$, and any solution to the $\mathcal{F}$-PCF problem gives a 3-approximation to the $\mathcal{F}$-center problem.

Handling outliers is a bit trickier. The above argument doesn't work since the 'responsible' point may be an outlier in the optimal solution and we can no longer assert that the optimal solution must contain a point from each part. Indeed, the nub of the problem seems to be figuring out which points should be outliers. The 3-approximation algorithm in [8] by Charikar *et al.* (see also paper [1]) cleverly chooses the partitioning via a greedy procedure, but their argument seems hard to generalize to other constraints.

A different attack used in the algorithm in [6] by Chakrabarty *et al.* and that in [15] by Harris *et al.* is by writing an LP relaxation and using the solution of the LP to recognize

the outliers. At a high level, the LP assigns each point x a variable (in this paper we call it $\mathsf{cov}(x)$) that indicates the extent to which x is served. Subsequently, the partitioning procedure described in the first paragraph is run, except the responsible points are considered in decreasing order of $\mathsf{cov}(x)$. The hope is that points assigned higher $\mathsf{cov}(x)$ in the LP solution are less likely to be outliers, and therefore the partition returned by the procedure can be used to recover a 3-approximate solution. This idea does work for the natural LP relaxation of the robust matroid center problem but fails for the natural LP relaxation of the robust knapsack center problem. Indeed, the latter has unbounded integrality gap.

Our solution is to use the round-or-cut framework that has recently been a powerful tool in designing many approximation algorithms (see [7, 20, 19, 2, 5]). We consider the following "coverage polytope" for the robust $\mathcal{F}$-center problem: the variables are $\mathsf{cov}(x)$ denoting the extent to which x is covered by a convex combination of sets $S \in \mathcal{F}$. Of course, we cannot hope to efficiently check whether a particular cov lies in this polytope. Nevertheless, we show that for any cov in the coverage polytope, the partitioning procedure when run in the decreasing order of cov, has the property that there *exists* a solution $S \in \mathcal{F}$ intersecting each part at most once which covers at least m points. We can then use the algorithm for $\mathcal{F}$-PCM to find this set. Furthermore, and more crucially, if the partitioning procedure does not have this property, then we can efficiently find a *hyperplane separating* cov from the the coverage polytope. Therefore, we can run the ellipsoid algorithm on the coverage polytope each time either obtaining a separating hyperplane, or obtaining a cov that leads to a desired partition, and therefore a 3-approximation.

2 Preliminaries

In this section we give formal definitions and statements of our results. As mentioned in a footnote in the Introduction, we focus on the supplier version of the problem.

▶ **Definition 1** ($\mathcal{F}$-Supplier Problem)**.** The input is a metric space (X, d) on a set of points $X = F \cup C$ with distance function $d : X \times X \longrightarrow \mathbb{R}_{\geq 0}$ and $\mathcal{F} \subseteq 2^F$ a down-closed family of subsets of F. The objective is to find $S \in \mathcal{F}$ such that $\max_{v \in C} d(v, S)$ is minimized.

▶ **Definition 2** (Robust $\mathcal{F}$-Supplier Problem)**.** The input is an instance of the $\mathcal{F}$-supplier problem along with an integer parameter $m \in \{0, 1, \ldots, |C|\}$. The objective is to find $S \in \mathcal{F}$ and $T \subseteq C$ for which $|T| \geq m$, and $\max_{u \in T} d(u, S)$ is minimized.

Thus an instance $\mathcal{I}$ of the robust $\mathcal{F}$-supplier problem is defined by the tuple $(F, C, d, m, \mathcal{F})$. In the definitions above, F and C are often called the set of *facilities* and *customers* respectively.

Given the set system $\mathcal{F}$ defined over F, we define the following optimization problem.

▶ **Definition 3** ($\mathcal{F}$-PCM problem)**.** The input is $\mathcal{J} = (F, \mathcal{F}, \mathcal{P}, \mathsf{val})$ where F is a finite set and $\mathcal{F} \subseteq 2^F$ is a down-closed family, $\mathcal{P} \subseteq 2^F$ is a sub-partition of F, and $\mathsf{val} : F \longrightarrow \{0, 1, 2, \cdots\}$ is an integer-valued function with maximum range $|\mathsf{val}|$ satisfying: $\forall f_1, f_2 \in A \in \mathcal{P}$, $\mathsf{val}(f_1) = \mathsf{val}(f_2)$. The objective is to find:

$$\mathsf{opt}(\mathcal{J}) = \max_{S \in \mathcal{F}} \quad \mathsf{val}(S) : \quad |S \cap A| \leq 1, \quad \forall A \in \mathcal{P}$$

The next theorem is the main result of the paper.

▶ **Theorem 4.** *Given a Robust $\mathcal{F}$-Supplier instance $\mathcal{I} = (F, C, d, m, \mathcal{F})$, Let $\mathcal{A}$ be an algorithm that solves any $\mathcal{F}$-PCM instance $\mathcal{J} = (F, \mathcal{F}, \mathcal{P}, \mathsf{val})$, with $|\mathsf{val}| \leq |C|$, in time bounded by $T_{\mathcal{A}}(\mathcal{J})$. Then, there is a 3-approximation algorithm for the Robust $\mathcal{F}$-Supplier instance that runs in time* $\mathrm{poly}(|\mathcal{I}|)T_{\mathcal{A}}(\mathcal{J})$.

The next theorem is the (easier) second part of the dichotomy theorem. We show that if $\mathcal{F}$-PCM cannot be solved, then the corresponding Robust $\mathcal{F}$-Supplier cannot be approximated.

▶ **Theorem 5.** *Given any non-trivial approximation algorithm $\mathcal{B}$ for the Robust $\mathcal{F}$-Supplier problem that runs in time $T_{\mathcal{B}}(|\mathcal{I}|)$ on instance $\mathcal{I}$, any $\mathcal{F}$-PCM instance $\mathcal{J} = (F, \mathcal{F}, \mathcal{P}, \mathsf{val})$ can be solved in time* $\mathrm{poly}(|\mathcal{J}|)\, T_{\mathcal{B}}(|\mathcal{I}|)$*, where* $|\mathcal{I}| = \mathrm{poly}(|\mathcal{J}|)$.

The proof can be found in the full version of the paper.

We end this section by setting a few notations used in the remainder of the paper. For any $u \in F \cup C$ we let $B_C(u, r)$ be the customers in a ball of radius r around u i.e. $B_C(u, r) = \{v \in C : d(u, v) \leq r\}$. Similarly, define $B_F(u, r)$ as the facilities in a ball of radius r around u i.e. for $u \in F \cup C$, $B_F(u, r) = \{f \in F : d(u, f) \leq r\}$.

3 Algorithm and Analysis : Proof of Theorem 4

We fix $\mathcal{I} = (F, C, d, \mathcal{F}, m)$ the instance of the Robust $\mathcal{F}$-Supplier problem. We use $\widehat{\mathsf{opt}}$ to denote *our guess* of the value of the optimal solution. Without loss of generality, we can always assume $\widehat{\mathsf{opt}} = 1$ because if not, we could scale d to meet this criteria. Our objective henceforth is to either find a set $S \in \mathcal{F}$ such that $|\{v \in C : d(v, S) \leq 1\}| \geq m$, or prove that $\mathsf{opt}(\mathcal{I}) > 1$.

There are two parts to our proof. The first part is a partitioning procedure which given an assignment $\mathsf{cov}(v) \in \mathbb{R}_{\geq 0}$ for every customer $v \in C$, constructs an instance $\mathcal{J}$ of $\mathcal{F}$-PCM. We call cov *valuable* if $\mathcal{J}$ has optimum value $\geq m$. Our procedure ensures that if cov is valuable, then we get a 3-approximate solution for $\mathcal{I}$. This is described in Section 3.1. The second part contains the proof of Theorem 4. In particular we show how using the round-and-cut methodology using polynomially many calls to $\mathcal{A}$ (recall this is the algorithm for $\mathcal{F}$-PCM) we can either prove $\mathsf{opt}(\mathcal{I}) > 1$, or find a valuable cov. This is described in Section 3.2.

3.1 Reduction to $\mathcal{F}$-PCM

Algorithm 1 inputs an assignment $\{\mathsf{cov}(v) \in \mathbb{R}_{\geq 0} : v \in C\}$. It returns a sub-partition $\mathcal{P}$ of F and assigns $\mathsf{val} : F \to \{0, 1, \cdots, |C|\}$ such that all the facilities in the same part of $\mathcal{P}$ get the same val. That is, it returns an $\mathcal{F}$-PCM instance $\mathcal{J} = (F, \mathcal{F}, \mathcal{P}, \mathsf{val})$ with $|\mathsf{val}| \leq |C|$.

The algorithm maintains a set of *uncovered* customers $U \subseteq C$ initialized to C (Line 1). In each iteration, it picks the customer $v \in U$ with maximum cov (Line 5) and adds it to set $\mathsf{Reps}_{\mathsf{cov}}$ (Line 6). We add the set of facilities $B_F(v, 1)$ at distance 1 from v to $\mathcal{P}$ (Line 7, 8). For each such v, we eke out the subset $\mathsf{Chld}(v) = B_C(v, 2) \cap U$ of currently uncovered clients "represented" by v (Line 9). For every facility $f \in B_F(v, 1)$ we define its *value* to be: $\mathsf{val}(f) = |\mathsf{Chld}(v)|$ (Line 10). At the end of the iteration, $\mathsf{Chld}(v)$ is removed from U (Line 11) and the loop continues till U becomes $\emptyset$. This way, the algorithm partitions C into $\{\mathsf{Chld}(v) : v \in \mathsf{Reps}_{\mathsf{cov}}\}$ (see fact(6)). Claim 8 shows that $\mathcal{P}$ is a sub-partition of F.

▶ **Fact 6.** $\{\mathsf{Chld}(v) : v \in \mathsf{Reps}_{\mathsf{cov}}\}$ *is a partition of* C.

▶ **Fact 7.** *For a* $v \in \mathsf{Reps}_{\mathsf{cov}}$ *and any* $u \in \mathsf{Chld}(v)$ *line 6 of the algorithm implies* $\mathsf{cov}(v) \geq \mathsf{cov}(u)$.

▶ **Claim 8.** *$\mathcal{P}$ constructed by Algorithm 1 is a sub-partition of F.*

Proof. By Line 11 of the algorithm, for each $u, v \in \mathsf{Reps}_{\mathsf{cov}}$ we have $d(u, v) > 2$ hence $B_F(u, 1) \cap B_F(v, 1) = \emptyset$ implying $\mathcal{P}$ is a sub-partition of F. ◀

Algorithm 1 $\mathcal{F}$-PCM instance construction.

Input: Robust $\mathcal{F}$-Supplier instance $(F, C, d, m, \mathcal{F})$ and assignment $\{\mathsf{cov}(v) \in \mathbb{R}_{\geq 0} : v \in C\}$
Output: $\mathcal{F}$-PCM instance $(F, \mathcal{F}, \mathcal{P}, \mathsf{val})$

1: $U \leftarrow C$ ▷ The set of uncovered customers
2: $\mathsf{Reps}_{\mathsf{cov}} \leftarrow \emptyset$ ▷ The set of representatives
3: $\mathcal{P} \leftarrow \emptyset$ ▷ The sub-partition of F that will be returned
4: **while** $U \neq \emptyset$ **do**
5: $v \leftarrow \arg\max_{v \in U} \mathsf{cov}(v)$ ▷ The first customer in U in non-increasing cov order
6: $\mathsf{Reps}_{\mathsf{cov}} \leftarrow \mathsf{Reps}_{\mathsf{cov}} \cup v$
7: $B_F(v, 1) \leftarrow \{f \in F : d(f, v) \leq 1\}$ ▷ Facilities that can cover v with a ball of radius 1
8: $\mathcal{P} \leftarrow \mathcal{P} \cup B_F(v, 1)$
9: $\mathsf{Chld}(v) \leftarrow \{u \in U : d(u, v) \leq 2\}$ ▷ Equals to $B_C(v, 2) \cap U$
10: $\mathsf{val}(f) \leftarrow |\mathsf{Chld}(v)| \quad \forall f \in B_F(v, 1)$
11: $U \leftarrow U \backslash \mathsf{Chld}(v)$
12: **end while**

▶ **Claim 9.** *For each $v \in \mathsf{Reps}_{\mathsf{cov}}$ and $f \in B_F(v, 1)$, $\mathsf{Chld}(v) \subseteq B_C(f, 3)$.*

Proof. For any $u \in \mathsf{Chld}(v)$, we have $d(u, v) \leq 2$ and since $d(f, v) \leq 1$, the fact that d is metric implies $d(f, u) \leq 3$. ◀

▶ **Definition 10.** For $S \subseteq F$ let $R(S) = \{v \in \mathsf{Reps}_{\mathsf{cov}} : B_F(v, 1) \cap S \neq \emptyset\}$, be the set of representative customers in $\mathsf{Reps}_{\mathsf{cov}}$ that are covered by balls of radius 1 around the facilities in S.

▶ **Claim 11.** *Let $S \in \mathcal{F}$ be any feasible solution of the $\mathcal{F}$-PCM instance constructed by Algorithm 1. Then, $\sum_{f \in S} \mathsf{val}(f) = \sum_{v \in R(S)} |\mathsf{Chld}(v)|$.*

Proof. For an $f \in S$, according to Line 10 of the algorithm, $\mathsf{val}(f) > 0$ only if $f \in B_F(v, 1)$ for some $v \in \mathsf{Reps}_{\mathsf{cov}}$. Also, by definition of the $\mathcal{F}$-PCM problem, $|B_F(v, 1) \cap S| \leq 1$ for any $v \in \mathsf{Reps}_{\mathsf{cov}}$. That is, there is exactly one $f \in B_F(v, 1) \cap S$ for each $v \in R(S)$ and again by line 10, $\mathsf{val}(f) = |\mathsf{Chld}(v)|$. Summing this equality over all $v \in R(S)$ and the corresponding $f \in B_F(v, 1) \cap S$ proves the claim. ◀

▶ **Claim 12.** *Let $\mathcal{I} = (F, C, d, m, \mathcal{F})$ be a Robust $\mathcal{F}$-Supplier instance and let $\mathsf{cov} : C \to \mathbb{R}_{\geq 0}$ be a coverage function. Let $\mathcal{J} = (F, \mathcal{F}, \mathcal{P}, \mathsf{val})$ be the $\mathcal{F}$-PCM instance returned by Algorithm 1 on input $\mathcal{I}$ and cov. Given any feasible solution S to $\mathcal{J}$, we can cover at least $\mathsf{val}(S)$ customers of C by opening radius 3-balls around each facility in S.*

Proof. By considering $R(S)$ from Definition 10, Claim 11 gives: $\sum_{v \in R(S)} |\mathsf{Chld}(v)| = \sum_{f \in S} \mathsf{val}(f)$. From Fact 6, we get that for all $u, v \in \mathsf{Reps}_{\mathsf{cov}}$, $\mathsf{Chld}(u) \cap \mathsf{Chld}(v) = \emptyset$. Thus, $|\bigcup_{v \in R(S)} \mathsf{Chld}(v)| = \sum_{v \in R(S)} |\mathsf{Chld}(v)| = \mathsf{val}(S)$. Furthermore, by Claim 9, $\{v \in C : d(v, S) \leq 3\} \supseteq \bigcup_{u \in R(S)} \mathsf{Chld}(u)$ implying the size of the former is at least $\mathsf{val}(S)$, thus proving the lemma. ◀

The above claim motivates the following definition of *valuable* cov assignments, and the subsequent lemma.

▶ **Definition 13.** An assignment $\{\mathsf{cov}(v) \in \mathbb{R}_{\geq 0} : v \in C\}$ is *valuable* with respect to a Robust $\mathcal{F}$-Supplier instance $\mathcal{I} = (F, C, d, m, \mathcal{F})$, iff $\mathsf{opt}(\mathcal{J}) \geq m$, where $\mathcal{J}$ is the $\mathcal{F}$-PCM instance returned by Algorithm 1 from $\mathcal{I}$ and cov.

▶ **Lemma 14.** *Given an instance $\mathcal{I}$ of the Robust $\mathcal{F}$-Supplier problem with $\mathsf{opt}(\mathcal{I}) = 1$, and a valuable assignment cov with respect to it, we can obtain a 3-approximate solution in time $\mathrm{poly}(|\mathcal{I}|) + T_{\mathcal{A}}(\mathcal{J})$ where $\mathcal{J}$ is the instance constructed by Algorithm 1 from $\mathcal{I}$ and cov.*

Proof. Since cov is valuable, $\mathsf{opt}(\mathcal{J}) \geq m$. We use solver $\mathcal{A}$ to return an optimal solution $S \in \mathcal{F}$ with $\mathsf{val}(S) \geq m$. Claim 12 implies that S is a 3-approximate solution to $\mathcal{I}$. ◀

3.2 The Round and Cut Approach

If the guess $\widehat{\mathsf{opt}} = 1$ for $\mathcal{I} = (F, C, d, m, \mathcal{F})$ is at least $\mathsf{opt}(\mathcal{I})$, then the following polytope must be non-empty. To see this, if $S^* \in \mathcal{F}$ is the optimal solution to $\mathcal{I}$ then set $z_{S^*} := 1$ and $z_S := 0$ for $S \in \mathcal{F} \backslash S^*$.

$$\mathcal{P}^{\mathcal{I}}_{\mathsf{cov}} = \{(\mathsf{cov}(v) : v \in C) : \quad \sum_{v \in C} \mathsf{cov}(v) \geq m \tag{$\mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}$.1}$$

$$\forall v \in C, \ \mathsf{cov}(v) - \sum_{S \in \mathcal{F}: d(v,S) \leq 1} z_S = 0 \tag{$\mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}$.2}$$

$$\sum_{S \in \mathcal{F}} z_S = 1 \tag{$\mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}$.3}$$

$$\forall S \in \mathcal{F}, \quad z_S \geq 0\} \tag{$\mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}$.4}$$

Even though $\mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}$ has exponentially many auxiliary variables (z_S for all $S \in \mathcal{F}$), its dimension is still $|C|$. The following gives a family of valid inequalities for $\mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}$ via Farkas lemma.

▶ **Lemma 15.** *Let $\lambda(v) \in \mathbb{R}$ for every $v \in C$ be such that*

$$\sum_{\substack{v \in C: \\ d(v,S) \leq 1}} \lambda(v) \leq m \quad \forall S \in \mathcal{F} \tag{V1}$$

Then any $\mathsf{cov} \in \mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}$ satisfies

$$\sum_{v \in C} \lambda(v)\mathsf{cov}(v) \leq m \tag{V2}$$

Proof. Given $\mathsf{cov} \in \mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}$, there exists $\{z_S : S \in \mathcal{F}\}$ such that together they satisfy ($\mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}$.1)-($\mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}$.4).

$$\begin{aligned}\sum_{v \in C} \lambda(v)\mathsf{cov}(v) &=_{(\mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}.2)} \sum_{v \in C} \lambda(v) \sum_{\substack{S \in \mathcal{F}: \\ d(v,S) \leq 1}} z_S = \sum_{S \in \mathcal{F}} z_S \sum_{\substack{v \in C: \\ d(v,S) \leq 1}} \lambda(v) \\ &\leq_{(\mathrm{V1}),(\mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}.4)} m \sum_{S \in \mathcal{F}} z_S =_{(\mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}.3)} m\end{aligned}$$

◀

The next lemma shows that all cov's in $\mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}$ are valuable.

▶ **Lemma 16.** *Suppose an assignment $\{\mathsf{cov}(v) \in \mathbb{R}_{\geq 0} : v \in C\}$ is not valuable with respect to $\mathcal{I} = (F, C, d, m, \mathcal{F})$. Then there is a hyper-plane separating it from $\mathcal{P}^{\mathcal{I}}_{\mathsf{cov}}$ that can be constructed in polynomial time.*

Proof. If $\sum_{v\in C} \mathsf{cov}(v) < m$, this inequality itself is a separating hyper-plane and we are done. So we may assume $\sum_{v\in C} \mathsf{cov}(v) \geq m$.

Let $\mathcal{J} = (F, \mathcal{F}, \mathcal{P}, \mathsf{val})$ be the $\mathcal{F}$-PCM instance constructed by Algorithm 1 from $\mathcal{I}$ and cov. Fix $S \in \mathcal{F}$ and recall from Definition 10 that $R(S) = \{v \in \mathsf{Reps}_{\mathsf{cov}} : B_F(v,1) \cap S \neq \emptyset\}$. Pick an arbitrary $T \subseteq S$ for which $|B_F(v,1) \cap T| = 1$, for all $v \in R(S)$. Observe that by down-closedness of $\mathcal{F}$, we have $T \in \mathcal{F}$ which implies T is a feasible solution for $\mathcal{J}$, and since cov is not valuable $\mathsf{val}(T) < m$. Furthermore, Claim 11 applied to T gives $\mathsf{val}(T) = \sum_{v\in R(T)} |\mathsf{Chld}(v)|$. Since $R(S) = R(T)$ and $|\mathsf{Chld}(v)|$ is integer-valued, we get:

$$\sum_{v\in R(S)} |\mathsf{Chld}(v)| \leq m - 1 \tag{1}$$

Let $\alpha = \frac{m}{m-0.5} > 1$. Define $\lambda(v)$ for $v \in C$ as:

$$\lambda(v) = \begin{cases} \alpha|\mathsf{Chld}(v)| & v \in \mathsf{Reps}_{\mathsf{cov}} \\ 0 & \text{for all other } v \in C \end{cases}$$

Now observe that for any $S \in \mathcal{F}$:

$$\sum_{v\in C: d(v,S)\leq 1} \lambda(v) = \sum_{v\in \mathsf{Reps}_{\mathsf{cov}}: d(v,S)\leq 1} \alpha|\mathsf{Chld}(v)| = \alpha \sum_{v\in R(S)} |\mathsf{Chld}(v)| \leq \alpha(m-1) < m$$

That is, $\lambda(v)$'s satisfy (V1). Now we prove (V2) is not satisfied thus it can be used to separate cov from $\mathcal{P}^{\mathcal{J}}_{\mathsf{cov}}$.

$$\begin{aligned} \sum_{v\in C} \lambda(v)\mathsf{cov}(v) = \quad & \alpha \sum_{v\in \mathsf{Reps}_{\mathsf{cov}}} |\mathsf{Chld}(v)|\mathsf{cov}(v) \quad = \quad \alpha \sum_{v\in \mathsf{Reps}_{\mathsf{cov}}} \sum_{u\in \mathsf{Chld}(v)} \mathsf{cov}(v) \\ \geq_{\text{Fact 7}} \quad & \alpha \sum_{v\in \mathsf{Reps}_{\mathsf{cov}}} \sum_{u\in \mathsf{Chld}(v)} \mathsf{cov}(u) \quad =_{\text{Fact 6}} \quad \alpha \sum_{v\in C} \mathsf{cov}(v) \geq \alpha m > m \end{aligned}$$

◀

Proof of Theorem 4. Given the guess $\widehat{\mathsf{opt}}$ which is scaled to 1, we use the ellipsoid algorithm to check if $\mathcal{P}^{\mathcal{J}}_{\mathsf{cov}}$ is empty or not. Whenever ellipsoid asks if a given cov is in $\mathcal{P}^{\mathcal{J}}_{\mathsf{cov}}$ or not, run Algorithm 1 for this given cov to construct the corresponding $\mathcal{F}$-PCM instance $\mathcal{J}$ and use algorithm $\mathcal{A}$, promised in the statement of Theorem 4, to solve it. If $\mathsf{opt}(\mathcal{J}) \geq m$, then Lemma 14 implies that we have a 3-approximate solution. Otherwise, cov is not valuable, and we can use Lemma 16 to find a separating hyperplane. In polynomial time, either we get a $\mathsf{cov} \in \mathcal{P}^{\mathcal{J}}_{\mathsf{cov}}$ which by Lemma 16 has to be valuable, or we prove $\mathcal{P}^{\mathcal{J}}_{\mathsf{cov}}$ is empty and we modify our $\widehat{\mathsf{opt}}$ guess. For the correct guess, the latter case won't occur and we get a 3-approximate solution. ◀

4 Applications and Extensions

In this section we elaborate on the applications and extensions stated in the Introduction. We begin with looking at specific instances of $\mathcal{F}$ which have been studied in the literature, and some which have not.

Single and Multiple Knapsack Constraints. We look at

$$\mathcal{F}_{\mathsf{KN}} := \{S \subseteq F : \text{ for } i = 1, \ldots, d, \sum_{v\in S} w_i(v) \leq k_i\}$$

where there are d weight functions over F and k_i's are upper bounds on these weights. Of special interest is the case $d = 1$ in which we get the robust knapsack supplier problem also called the weighted k-supplier problem with outliers.

The $\mathcal{F}$-PCM problem for the above $\mathcal{F}_{\mathsf{KN}}$ has the following complexity: When $d = 1$, the problem can be solved in polynomial time. Indeed, given a partition $\mathcal{P}$, since $\mathsf{val}(u) = \mathsf{val}(v)$ for all v in the same part, any solution which picks a facility from a part $A \in \mathcal{P}$ may as well pick the one with the smallest weight in that part. Thus, the problem boils down to the usual knapsack problem in which we have $|\mathcal{P}|$ items where the item corresponding to part $A \in \mathcal{P}$ has weight $\min_{v \in A} w(v)$ and value $\mathsf{val}(v)$. Since the values are poly-bounded, this problem is solvable in polynomial time. Thus, we get the following corollary to Theorem 4 resolving the open question raised in [10] and [15].

▶ **Theorem 17.** *There is a polynomial time* 3*-approximation to the robust knapsack center problem.*

When $d > 1$, then the $\mathcal{F}$-PCM problem is NP-hard even when val is poly-bounded. However, if the w_i's are also poly-bounded (actually one of them can be general), then the $\mathcal{F}$-PCM problem can be solved in polynomial time using dynamic programming. This problem was in fact studied in [17] (the conference version) and is called the *suitcase* problem there. Thus, we get the following corollary to Theorem 4 extending the result in [17].

▶ **Theorem 18.** *There is a polynomial time* 3*-approximation to the robust multiple-knapsack center problem if the number of weights is a constant and all but possibly one weight function are poly-bounded.*

Single and Multiple Matroid Constraints. We look at

$$\mathcal{F}_{\mathsf{Mat}} := \{S \subseteq F : S \in \mathcal{I}_{M_i},\ \forall i = 1, \ldots, d\}$$

When $d = 1$, we get the robust matroid center problem. The $\mathcal{F}$-PCM paper reduces to finding a maximum value set in $\mathcal{I}_M$ and a partition matroid induced by $\mathcal{P}$. This is solvable in polynomial time even when val is general and not poly-bounded, and even when $\mathcal{I}_M$ is given as an independent set oracle. Thus, we get the following corollary to Theorem 4 obtaining the result in [15].

▶ **Theorem 19.** *[Theorem 1.1 in [15]] There is a polynomial time* 3*-approximation to the robust matroid center problem even when the matroid is described as an independent set oracle.*

When there are $d > 1$ matroids, then the $\mathcal{F}$-PCM problem is NP-hard. Therefore, Theorem 5 implies that for instance, we can have *no* unicriteria approximation for the robust matroid-intersection center problem.

Single Knapsack and Single Matroid Constraint. We look at

$$\mathcal{F}_{\mathsf{KN} \cap \mathsf{Mat}} := \{S \subseteq F : \sum_{v \in S} w(v) \leq k, \quad S \in \mathcal{I}_M\}$$

which is the intersection of a single matroid and a single knapsack constraint. To the best of our knowledge, the resulting Robust $\mathcal{F}$-Supplier problem has not been studied before. One natural instantiation is when F is a collection of high-dimensional vectors with weights and the constraint on the centers is to pick a linearly independent set with total weight at most k.

The corresponding $\mathcal{F}$-PCM problem asks us, given a partition $\mathcal{P}$ and poly-bounded values val, to find a set $S \in \mathcal{I}_{\mathcal{M}} \cap \mathcal{I}_{\mathcal{P}}$ of maximum value such that $w(S) \leq k$, where $\mathcal{I}_{\mathcal{P}}$ is the partition matroid induced by $\mathcal{P}$. We don't know if this problem can be solved in polynomial time, even in the case when M is another partition matroid.

However, the above problem is related to the *exact matroid intersection* problem. In this problem, we are given two matroids $\mathcal{M}$ and $\mathcal{P}$, and a weight function $\mathbf{w}$ on each ground element and a budget $\mathbf{W}$. The objective is to decide whether or not there is a set $S \in \mathcal{I}_{\mathcal{M}} \cap \mathcal{I}_P$ such that $\mathbf{w}(S) = \mathbf{W}$. Understanding the complexity of this problem is a long standing challenge [4, 21, 22]. When the matroids are representable over the same field, then [4] gives a randomized pseudopolynomial time algorithm for the problem. The following claim shows the relation between $\mathcal{F}$-PCM and the exact matroid intersection problem; this claim is essentially present in [3] and the reader can refer to the full version of our paper for the proof.

▶ **Claim 20.** *Given an algorithm for the exact matroid intersection problem, one can solve the $\mathcal{F}$-PCM problem in polynomial time when the weights w are poly-bounded.*

Armed with the non-trivial result about exact matroid intersection from [4], we get the following.

▶ **Theorem 21.** *Given a linear matroid $\mathcal{M}$ and a poly-bounded weight function, there is a randomized polynomial time 3-approximation to the robust knapsack-and-matroid center problem.*

4.1 The Case of No Outliers

The $\mathcal{F}$-supplier problem, that is the case of $m = |C|$, may be of special interest. In this case the problem is easier and the complexity is defined by the complexity of the following decision problem.

▶ **Definition 22** ($\mathcal{F}$-PCF problem)**.** The input is $\mathcal{J} = (F, \mathcal{F}, \mathcal{P})$ where F is a finite set, $\mathcal{F} \subseteq 2^F$ is a down-closed family and $\mathcal{P} \subseteq 2^F$ is an arbitrary sub-partition of F. The objective is to decide whether there *exists* a set $S \in \mathcal{F}$ such that $|S \cap A| = 1, \quad \forall A \in \mathcal{P}$.

▶ **Theorem 23.** *If the $\mathcal{F}$-PCF problem can be solved efficiently for any partition $\mathcal{P}$, then the $\mathcal{F}$-supplier problem has a polynomial time 3-approximation. Otherwise, there is no non-trivial approximation possible for the $\mathcal{F}$-supplier problem.*

Sketch. Run Algorithm 1 with an arbitrary assignment cov (and ignore the val's). Let $\mathcal{J} = (F, \mathcal{F}, \mathcal{P})$ be the resulting $\mathcal{F}$-PCF instance. If the guess $\widehat{\mathsf{opt}} = 1$ is correct, then note that the optimum solution S^* must satisfy $S^* \cap A \neq \emptyset$ for all $A \in \mathcal{P}$; if not, then the corresponding $v \in \mathsf{Reps}_{\mathsf{cov}}$ can't be served. Conversely, any S satisfying $S \cap A \neq \emptyset$ for all $A \in \mathcal{P}$ implies a 3-approximate solution. Therefore, an algorithm for $\mathcal{F}$-PCF can either give a 3-approximate solution or prove the guess $\widehat{\mathsf{opt}}$ is too low. ◀

Theorem 4 and Theorem 23 raise the question: is there any set of constraints for which the problem without outliers is significantly easier than the problem with outliers? We don't know the answer to this question, although we guess the answer is yes. For this, it suffices to design a set system for which $\mathcal{F}$-PCF is easy but $\mathcal{F}$-PCM is hard (perhaps NP-hard). To see the difference between these problems consider the $\mathcal{F}_{\mathsf{KN} \cap \mathsf{Mat}}$ family described in the previous subsection. We don't know if $\mathcal{F}$-PCM is easy or hard, but $\mathcal{F}$-PCF is easy: this amounts to minimizing $w(S)$ over $S \in \mathcal{I}_{\mathcal{M}} \cap \mathcal{B}_{\mathcal{P}}$ where $\mathcal{B}_{\mathcal{P}}$ is the base polytope induced by $\mathcal{P}$. This can be done in polynomial time, and therefore we get the following corollary.

▶ **Theorem 24.** *There is a polynomial time* 3*-approximation to the knapsack-and-matroid center problem.*

4.2 Handling Approximation

The technique used to prove Theorem 4 is robust enough to translate approximation algorithms for the $\mathcal{F}$-PCM problem to *bi-criteria* approximation algorithms for the Robust $\mathcal{F}$-Supplier problem. There are two notions of approximation algorithms for the $\mathcal{F}$-PCM problem and they lead to two notions of bi-criteria approximation.

The first is the standard notion: a ρ-approximation (for $\rho \leq 1$) algorithm that takes instance $\mathcal{J}$ of $\mathcal{F}$-PCM, returns a solution $S \in \mathcal{F}$ of value $\mathsf{val}(S) \geq \rho \cdot \mathsf{opt}(\mathcal{J})$. The corresponding bi-criteria approximation notion for the Robust $\mathcal{F}$-Supplier problem is the following: an (α, β)-approximation algorithm for instance $\mathcal{I}$ of Robust $\mathcal{F}$-Supplier returns a solution which opens centers at $S \in \mathcal{F}$ and the distance of at least βm customers to S is $\leq \alpha \cdot \mathsf{opt}(\mathcal{I})$. The proof of Theorem 4 in fact implies the following.

▶ **Theorem 25.** *Let $\mathcal{A}$ be a polynomial time ρ-approximate algorithm for the $\mathcal{F}$-PCM problem. Then there is a polynomial time $(3, \rho)$-bi-criteria approximation algorithm for the Robust $\mathcal{F}$-Supplier problem.*

The second notion of approximation for the $\mathcal{F}$-PCM problem is one which satisfies the constraints approximately. This notion is more problem dependent and makes sense only if there is a notion of an approximate relaxation $\mathcal{F}^R$ for the set $\mathcal{F}$. For example, an $(1+\varepsilon)$-relaxation for $\mathcal{F}_{\mathsf{KN}}$ could be the subsets S with $w_i(S) \leq (1+\varepsilon) \cdot k_i$ for all i. A ρ-violating algorithm for an instance $\mathcal{J}$ of $\mathcal{F}$-PCM would then return a set S with $\mathsf{val}(S) \geq \mathsf{opt}(\mathcal{J})$ but $S \in \mathcal{F}^R$ which is an ρ-relaxation for $\mathcal{F}$. This defines a different bi-criteria approximation notion for the Robust $\mathcal{F}$-Supplier problem. An α-approximate β-violating algorithm for the Robust $\mathcal{F}$-Supplier problem takes an instance $\mathcal{I}$ and returns a solution $S \in \mathcal{F}^R$ which is a β-relaxation for $\mathcal{F}$ such that at least m customers in C are at distance at most $\alpha \cdot \mathsf{opt}(\mathcal{I})$ to S.

▶ **Theorem 26.** *Let $\mathcal{A}$ be a polynomial time ρ-violating algorithm for the $\mathcal{F}$-PCM problem. Then there is a polynomial time* 3*-approximate-ρ-violating algorithm for the Robust $\mathcal{F}$-Supplier problem.*

When $\mathcal{F}$ is described by constant d knapsack constraints (with general weights) and a single matroid constraint, for any constant $\varepsilon > 0$ Chekuri *et al.* give an $(1+\varepsilon)$-approximation algorithm for the $\mathcal{F}$-PCM in [9]. Without the matroid constraint, Grandoni *et al.* give an $(1+\varepsilon)$-violating algorithm in [12]. Together, we get the following corollary. The latter recovers a result from [10].

▶ **Theorem 27.** *Fix any constant $\varepsilon > 0$. There is a polynomial time $(3, (1+\varepsilon))$-bi-criteria approximation algorithm for the robust supplier problem with constant many knapsack constraints and one matroid constraint. There is a polynomial time* 3*-approximate $(1+\varepsilon)$-violating algorithm for the robust supplier problem with constant many knapsack constraints.*

References

1 Gagan Aggarwal, Rina Panigrahy, Tomás Feder, Dilys Thomas, Krishnaram Kenthapadi, Samir Khuller, and An Zhu. Achieving Anonymity via Clustering. *ACM Trans. Algorithms*, 6(3):49:1–49:19, 2010. doi:10.1145/1798596.1798602.

2 Hyung-Chan An, Mohit Singh, and Ola Svensson. LP-based algorithms for capacitated facility location. In *Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, FOCS '14, pages 256–265, Washington, DC, USA, 2014. IEEE Computer Society. doi:10.1109/FOCS.2014.35.

3 André Berger, Vincenzo Bonifaci, Fabrizio Grandoni, and Guido Schäfer. Budgeted Matching and Budgeted Matroid Intersection via the Gasoline Puzzle. *Mathematical Programming*, 128(1):355–372, 2011. doi:10.1007/s10107-009-0307-4.

4 Paolo M. Camerini, Giulia Galbiati, and Francesco Maffioli. Random Pseudo-Polynomial Algorithms for Exact Matroid Problems. *J. Algorithms*, 13(2):258–273, 1992. doi:10.1016/0196-6774(92)90018-8.

5 Robert D. Carr, Lisa K. Fleischer, Vitus J. Leung, and Cynthia A. Phillips. Strengthening Integrality Gaps for Capacitated Network Design and Covering Problems. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 106–115, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=338219.338241.

6 Deeparnab Chakrabarty, Prachi Goyal, and Ravishankar Krishnaswamy. The Non-Uniform k-Center Problem. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 67:1–67:15, 2016. doi:10.4230/LIPIcs.ICALP.2016.67.

7 Deeparnab Chakrabarty, Ravishankar Krishnaswamy, and Amit Kumar. The Heterogeneous Capacitated k-Center Problem. In *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 123–135, 2017. doi:10.1007/978-3-319-59250-3_11.

8 Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. Algorithms for Facility Location Problems with Outliers. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 642–651, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

9 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Multi-budgeted Matchings and Matroid Intersection via Dependent Rounding. *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1080–1097, 2011. doi:10.1137/1.9781611973082.82.

10 Danny Z. Chen, Jian Li, Hongyu Liang, and Haitao Wang. Matroid and Knapsack Center Problems. In *Integer Programming and Combinatorial Optimization (IPCO)*, pages 110–122, 2013.

11 Teofilo F. Gonzalez. Clustering to Minimize the Maximum Intercluster Distance. *Theoretical Computer Science*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.

12 Fabrizio Grandoni, R. Ravi, Mohit Singh, and Rico Zenklusen. New Approaches to Multi-Objective Optimization. *Math. Program.*, 146(1-2):525–554, 2014. doi:10.1007/s10107-013-0703-7.

13 Seifollah L. Hakimi. Optimum Locations of Switching Centers and the Absolute Centers and Medians of a Graph. *Ops Res.INFORMS*, 12:450–459, 1964. doi:10.1287/opre.12.3.450.

14 Seifollah L. Hakimi. Optimum Distribution of Switching Centers in a Communication Network and Some Related Graph Theoretic Problems. *Ops Res.INFORMS*, 13:462–475, 1965. doi:10.1287/opre.13.3.462.

15 David G. Harris, Thomas Pensyl, Aravind Srinivasan, and Khoa Trinh. A Lottery Model for Center-Type Problems with Outliers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, pages 10:1–10:19, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.10.

16 Dorit S. Hochbaum and David B. Shmoys. A Best Possible Heuristic for the k-Center Problem. *Math. Oper. Res.*, 10(2):180–184, 1985. doi:10.1287/moor.10.2.180.

17 Dorit S. Hochbaum and David B. Shmoys. A Unified Approach to Approximation Algorithms for Bottleneck Problems. *J. ACM*, 33(3):533–550, 1986. doi:10.1145/5925.5933.

18 Wen-Lian Hsu and George L. Nemhauser. Easy and Hard Bottleneck Location Problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979. doi:10.1016/0166-218X(79)90044-1.

19 Shi Li. On Uniform Capacitated k-median Beyond the Natural LP Relaxation. In *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 696–707, Philadelphia, PA, USA, 2015. Society for Industrial and Applied Mathematics.

20 Shi Li. Approximating Capacitated k-median with $(1+\varepsilon)k$ Open Facilities. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 786–796, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=2884435.2884491.

21 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is As Easy As Matrix Inversion. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 345–354, New York, NY, USA, 1987. ACM. doi:10.1145/28395.383347.

22 Christos H. Papadimitriou and Mihalis Yannakakis. The Complexity of Restricted Spanning Tree Problems. *J. ACM*, 29(2):285–309, 1982. doi:10.1145/322307.322309.

Orthogonal Point Location and Rectangle Stabbing Queries in 3-d

Timothy M. Chan
Dept. of Computer Science, University of Illinois at Urbana-Champaign, USA
tmc@illinois.edu

Yakov Nekrich
Cheriton School of Computer Science, University of Waterloo, Canada
yakov.nekrich@googlemail.com

Saladi Rahul
Dept. of Computer Science, University of Illinois at Urbana-Champaign, USA
saladi@uiuc.edu

Konstantinos Tsakalidis[1]
Dept. of Computer and Information Science, Tandon School of Engineering, New York University, USA
kt79@nyu.edu

Abstract

In this work, we present a collection of new results on two fundamental problems in geometric data structures: *orthogonal point location* and *rectangle stabbing*.

- **Orthogonal point location.** We give the first linear-space data structure that supports 3-d point location queries on n disjoint axis-aligned boxes with *optimal* $O(\log n)$ query time in the (arithmetic) pointer machine model. This improves the previous $O\left(\log^{3/2} n\right)$ bound of Rahul [SODA 2015]. We similarly obtain the first linear-space data structure in the I/O model with optimal query cost, and also the first linear-space data structure in the word RAM model with sub-logarithmic query time.
- **Rectangle stabbing.** We give the first linear-space data structure that supports 3-d 4-sided and 5-sided rectangle stabbing queries in *optimal* $O(\log_w n + k)$ time in the word RAM model. We similarly obtain the first optimal data structure for the closely related problem of 2-d top-k rectangle stabbing in the word RAM model, and also improved results for 3-d 6-sided rectangle stabbing.

For point location, our solution is simpler than previous methods, and is based on an interesting variant of the van Emde Boas recursion, applied in a round-robin fashion over the dimensions, combined with bit-packing techniques. For rectangle stabbing, our solution is a variant of Alstrup, Brodal, and Rauhe's grid-based recursive technique (FOCS 2000), combined with a number of new ideas.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases geometric data structures, orthogonal point location, rectangle stabbing, pointer machines, I/O model, word RAM model

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.31

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.08602`.

[1] Partially supported by NSF grants CCF-1319648 and CCF-1533564

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 31; pp. 31:1–31:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In this work we present a plethora of new results on two fundamental problems in geometric data structures: (a) *orthogonal point location* (where the input rectangle or boxes are non-overlapping), and (b) *rectangle stabbing* (where the input rectangles or boxes are overlapping).

1.1 Orthogonal point location

Point location is among the most central problems in the field of computational geometry, which is covered in textbooks and has countless applications. In this paper we study the *orthogonal point location* problem. Formally, we want to preprocess a set of n *disjoint* axis-aligned boxes (hyperrectangles) in $\mathbb{R}^d$ into a data structure, so that the box in the set containing a given query point (if any) can be reported efficiently. There are two natural versions of this problem, for (a) *arbitrary disjoint boxes* where the input boxes need not fill the entire space, and (b) a *subdivision* where the input boxes fill the entire space.

Arbitrary disjoint boxes. Historically, the point location problem has been studied in the pointer machine model and the main question has been the following:

"Is there a linear-space structure with $O(\log n)$ query time?"

In 2-d this question has been successfully resolved: there exists a linear-space structure with $O(\log n)$ query time [19, 18, 14, 27, 30] (actually this result holds for nonorthogonal point location). In 3-d there has been work on this problem [15, 17, 2, 23], but the question has not yet been resolved. The currently best known result on the pointer machine model is a linear-space structure with $O(\log^{3/2} n)$ query time by Rahul [23]. In this paper,

- we obtain the first linear-space structure with $O(\log n)$ query time for 3-d orthogonal point location for arbitrary disjoint boxes. The structure works in the (arithmetic) pointer machine model and is *optimal* in this model.

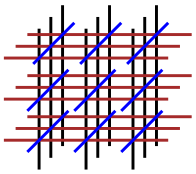
The orthogonal point location problem has been studied in the I/O-model and the word RAM as well (please see the full version for a brief description of these models). In the I/O model, an optimal solution is known in 2-d [16, 6]: a linear-space structure with $O(\log_B n)$ query time, where B is the block size (this result holds for nonorthogonal point location). However, in 3-d the best known result is a linear-space structure with a query cost of $O(\log_B^2 n)$ I/Os by Nekrich [20] (for orthogonal point location for disjoint boxes).

- In the I/O model, we obtain the first linear-space structure with $O(\log_B n)$ query cost for 3-d orthogonal point location for arbitrary disjoint boxes. This result is *optimal*.

In the word RAM model, an optimal solution in 2-d was given by Chan [10] with a query time of $O(\log\log U)$, assuming that input coordinates are in $[U] = \{0, 1, \ldots, U-1\}$. However, in 3-d the best known result for arbitrary disjoint boxes is a linear-space structure with $O(\log n \log\log n)$ query time: this result was not stated explicitly before but can obtained by an interval tree augmented with Chan's 2-d orthogonal point location structure [10] at each node. Our above new result with logarithmic query time is already an improvement even in the word RAM, but we can do slightly better still:

- In the w-bit word RAM model, we obtain the first linear-space structure with *sub-logarithmic* query time for 3-d orthogonal point location for arbitrary disjoint boxes. The time bound is $O(\log_w n)$. (We do not know whether this result is optimal, however.)

Subdivisions. In the plane, the two versions of the problem are equivalent in the sense that any arbitrary set of n disjoint rectangles can be converted into a subdivision of $\Theta(n)$ rectangles via the vertical decomposition. In 3-d, the two versions are no longer equivalent, since there exist sets of n disjoint boxes that need $\Omega\left(n^{3/2}\right)$ boxes to fill the entire space. See figure on the right.



In 3-d the special case of a subdivision is potentially easier than the arbitrary disjoint boxes setting, as the former allows for a fast $O(\log^2 \log U)$ query time in the word RAM model with $O(n \log \log U)$ space, as shown by de Berg, van Kreveld, and Snoeyink [13] (with an improvement by Chan [10]).

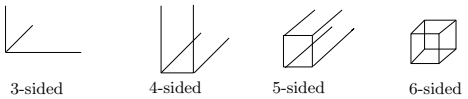
- In the word RAM model, we further improve de Berg, van Kreveld, and Snoeyink's method to achieve a *linear*-space structure with $O(\log^2 \log U)$ query time for 3-d orthogonal point location on subdivisions.

1.2 Rectangle stabbing

Rectangle stabbing is a classical problem in geometric data structures [1, 3, 7, 12, 23], which is as old, and as equally natural, as orthogonal range searching—in fact, it can be viewed as an "inverse" of orthogonal range searching, where the input objects are boxes and query objects are points, instead of vice versa. Formally, we want to preprocess a set S of n axis-aligned boxes (possibly overlapping) in $\mathbb{R}^d$ into a data structure, so that the boxes in S containing a given query point q can be reported efficiently. (As one of many possible applications, imagine a dating website, where each lady is interested in gentlemen whose salary is in a range $[S_1,S_2]$ and age is in a range $[A_1,A_2]$; suppose that a gentleman with salary x_q and age y_q wants to identify all ladies who might be potentially interested in him.)

Throughout this paper, we will assume that the endpoints of the rectangles lie on the grid $[2n]^3$ (this can be achieved via a simple rank-space reduction). In the word RAM model, Pătraşcu [21] gave a lower bound of $\Omega(\log_w n)$ query time for any data structure which occupies at most $n \log^{O(1)} n$ space to answer the 2-d rectangle stabbing query. Shi and Jaja [29] presented an optimal solution in 2-d which occupies linear space with $O(\log_w n + k)$ query time, where k is the number of rectangles reported.

We introduce some notation to define various types of rectangles in 3-d. (We will use the terms "rectangle" and "box" interchangably throughout the paper.) A rectangle in 3-d is called $(3+t)$-*sided* if it is bounded in t out of the 3 dimensions and unbounded (on one side) in the remaining $3-t$ dimensions.



In the word RAM model, an optimal solution in 3-d is known *only* for the 3-sided rectangle stabbing query: a linear-space structure with $O(\log \log_w n + k)$ query time (by combining the work of Afshani [1] and Chan [10]; this is optimal due to the lower bound of Pătraşcu and Thorup [22]). Finding an optimal solution for 4-, 5-, and 6-sided rectangle stabbing has remained open.

3-d 4- and 5-sided rectangle stabbing. Currently, the best-known result for 4-sided and 5-sided rectangle stabbing queries by Rahul [23] occupies $O(n \log^* n)$ space with $O(\log n + k)$ and $O(\log n \log \log n + k)$ query time, respectively. This result holds in the pointer machine model. For 4-sided rectangle stabbing, adapting Rahul's solution to the word RAM model does not lead to any improvement in the query time (the bottleneck is in answering $\log n$

3-d dominance reporting queries). For 5-sided rectangle stabbing, even if we assume the existence of an optimal 4-sided rectangle stabbing structure, plugging it into Rahul's solution can improve the query time to only $O(\log n + k)$, which is still suboptimal. In this paper,

- we obtain the first *optimal* solution for 3-d 4-sided and 5-sided rectangle stabbing in the word RAM model: a linear-space structure with $O(\log_w n + k)$ query time.

2-d top-k rectangle stabbing. Recently, there has been a lot of interest in top-k range searching [4, 8, 9, 24, 25, 26, 28, 31]. Specifically, in the 2-d top-k rectangle stabbing problem, we want to preprocess a set of *weighted* axis-aligned rectangles in 2-d, so that given a query point q and an integer k, the goal is to report the k largest-weight rectangles containing (or stabbed by) q. This problem is closely related to the 5-sided rectangle stabbing problem (by treating the weight as a third dimension, a rectangle r with weight $w(r)$ can be mapped to a 5-sided rectangle $r \times (-\infty, w(r)]$).

- By extending the solution for 3-d 5-sided rectangle stabbing problem, we obtain the first *optimal* solution for the 2-d top-k rectangle stabbing problem: a linear-space structure with $O(\log_w n + k)$ query time.

3-d 6-sided rectangle stabbing. Our new solution to 3-d 5-sided rectangle stabbing, combined with standard interval trees, immediately implies a solution to 3-d 6-sided rectangle stabbing with a query time of $O(\log_w n \cdot \log n + k)$, which is already new. But we can do slightly better still:

- We obtain a linear-space structure with $O(\log_w^2 n + k)$ query time for 3-d 6-sided rectangle stabbing problem in the word RAM model. We conjecture this to be optimal (the analogy is the lower bound of $\Omega(\log^2 n + k)$ query time for linear-space pointer machine structures [3]).

Back to orthogonal point location. Our solution for orthogonal point location uses rectangle stabbing as a subroutine: if there is an $S(n)$-space data structure with $Q(n) + O(k)$ query time to answer the rectangle stabbing problem in $\Re^d$, then one can obtain a data structure for orthogonal point location in $\Re^{d+1}$ with $O(S(n))$-space and $O(Q(n))$ time. By plugging in our new results for 3-d 6-sided rectangle stabbing, we obtain a linear-space word RAM structure which can answer any orthogonal point location query in 4-d in $O(\log_w^2 n)$ time, improving the previously known $O(\log^2 n \log\log n)$ bound [10].

1.3 Our techniques

Our results are obtained using a number of new ideas (in addition to existing data structuring techniques), which we feel are as interesting as the results themselves.

3-d orthogonal point location. To better appreciate our new 3-d orthogonal point location method, we first recall that the current best word-RAM method had $O(\log n \log\log n)$ query time, and was obtained by building an interval tree over the x-coordinates, and at each node of the tree, storing Chan's 2-d point location data structure on the yz-projection of the rectangles. Interval trees caused the query time to increase by a logarithmic factor, while Chan's 2-d structures achieved $O(\log\log n)$ query time via a complicated van-Emde-Boas-like

recursion. We can thus summarize this approach loosely by the following recurrence for the query time (superscripts refer to the dimension):

$$Q^{(3)}(n) = O(Q^{(2)}(n)\log n) \text{ and } Q^{(2)}(n) = Q^{(2)}(\sqrt{n}) + O(1)$$
$$\implies Q^{(3)}(n) = O(\log n \log\log n).$$

(Note that naively increasing the fan-out of the interval tree could reduce the query time but would blow up the space usage.)

In the pointer machine model, the current best data structure by Rahul [23], with $O(\log^{3/2} n)$ query time, required an even more complicated combination of interval trees, Clarkson and Shor's random sampling technique, 3-d rectangle stabbing, and 2-d orthogonal point location.

To avoid the extra $\log\log n$ factor, we cannot afford to use Chan's 2-d orthogonal point location structure as a subroutine; and we cannot work with just yz-projections, which intuitively cause loss of efficiency. Instead, we propose a more direct solution based on a new van-Emde-Boas-like recursion, aiming for a new recurrence of the form

$$Q^{(3)}(n) = Q^{(3)}(\sqrt{n}) + O(\log n).$$

The $O(\log n)$ term arises from the need to solve 2-d rectangle stabbing subproblems, on projections along all three directions (the yz-, xz-, and xy-plane), applied in a *round-robin* fashion. The new recurrence then solves to $O(\log n)$—notice how $\log\log$ disappears, unlike the usual van Emde Boas recursion! In the word RAM model, we can even use known sub-logarithmic solutions to 2-d rectangle stabbing to get $O(\log_w n)$ query time.

We emphasize that our new method is much *simpler* than the previous, slower methods, and is essentially self-contained except for the use of a known data structure for 2-d rectangle stabbing emptiness (which reduces to standard 2-d orthogonal range counting).

One remaining issue is space. In our new method, a rectangle is stored $O(\log\log n)$ times, due to the depth of the recursion. To achieve linear space, we need another idea, namely, *bit-packing* tricks, to compress the data structure. Because of the rapid reduction of the universe size in the round-robin van-Emde-Boas recursion, the amortized space in words per input box satisfies a recurrence of the form

$$s(n) = s(\sqrt{n}) + O\left(\frac{\log n}{w}\right) \implies s(n) = O\left(\frac{\log n}{w}\right) = O(1).$$

Our new result on the subdivision case is obtained by a similar space-reduction trick.

3-d 5-sided rectangle stabbing. For 3-d rectangle stabbing, the previous solution by Rahul [23] was based on a grid-based, $\sqrt{n}$-way recursive approach of Alstrup, Brodal, and Rauhe [5], originally designed for 2-d orthogonal range searching. The fact that the approach can be adapted here is nontrivial and interesting, since our input objects are now more complicated (rectangles instead of points) and the target query time is quite different (near logarithmic rather than $\log\log$). More specifically, Rahul first solved the 4-sided case via a complicated data structure, and then applied Alstrup et al.'s technique to reduce 5-sided rectangles to 4-sided rectangles, which led to a query-time recurrence similar to the following (subscripts denote the number of sides, and output cost related to k is ignored):

$$Q_4(n) = O(\log n) \text{ and } Q_5(n) = 2Q_5(\sqrt{n}) + O(Q_4(n)) \implies Q_5(n) = O(\log n \log\log n).$$

Intuitively, the reduction from the 5-sided to the 4-sided case causes loss of efficiency. To avoid the extra $\log\log n$ factor, we propose a new method that is also based on Alstrup *et*

al.'s recursive technique, but reduces 5-sided rectangles directly to 3-sided rectangles, aiming for a new recurrence of the form

$$Q_3(n) = O(\log\log_w n) \text{ and } Q_5(n) = 2Q_5(\sqrt{n}) + O(Q_3(n)).$$

During recursion, we do not put 4-sided rectangles in separate structures (which would slow down querying), but instead use a common tree for both 4-sided and 5-sided rectangles. The new recurrence then solves to $Q_5(n) = O(\log_w n)$ with an appropriate base case—notice how $\log\log$ again disappears, and notice how this gives a new result even for the 4-sided case!

One remaining issue is space. Again, we can compress the data structure by incorporating bit-packing tricks (which was also used in Alstrup *et al.*'s original method). For 4- and 5-sided rectangle stabbing, the space recurrence then solves to linear.

However, with space compression, a new issue arises. The cost of reporting each output rectangle in a query increases to $O(\log\log n)$ (the depth of the recursion), because of the need to *decode* the coordinates of a compressed rectangle. In other words, the query cost becomes $O(\log_w n + k\log\log n)$ instead of $O(\log_w n + k)$. This extra decoding overhead also occurred in previous work on 2-d orthogonal range searching by Alstrup *et al.* [5] and Chan *et al.* [11], and it is open how to avoid the overhead for that problem without sacrificing space (this is related to the so-called *ball inheritance problem* [11]).

We observe that for the 4- and 5-sided rectangle stabbing problem, a surprisingly simple idea suffices to avoid the overhead: instead of keeping pointers between consecutive levels of the recursion tree, we just keep pointers directly from each level to the *leaf* level.

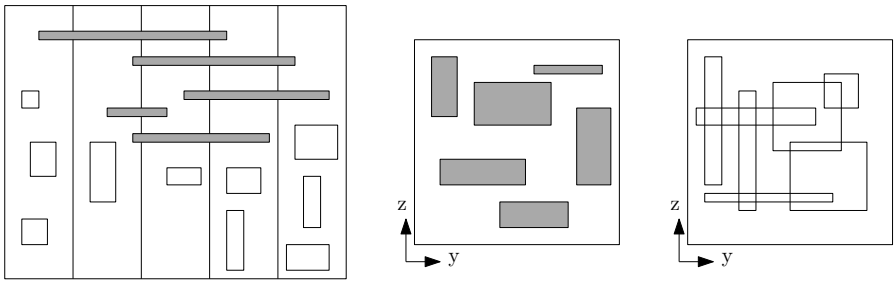
3-d 6-sided rectangle stabbing. We can solve 6-sided rectangle stabbing by using our result for 5-sided rectangle stabbing as a subroutine. However, the naive reduction via interval trees increases the query time by a $\log n$ factor instead of $\log_w n$. To speed up querying, the standard idea is to use a tree with a larger fan-out w^ε. This leads to various colored generalizations of 2-d rectangle stabbing with a small number w^ε of colors. Much of our ideas can be extended to solve these colored subproblems in a straightforward way, but a key subproblem, of answering colored 2-d dominance searching queries in $O(\log\log_w n + k)$ time with linear space, is nontrivial. We solve this key subproblem via a clever use of 2-d shallow cuttings, combined with a grouping trick, which may be of independent interest.

2 Orthogonal Point Location in 3-d

Preliminaries. Our solution to 3-d orthogonal point location will require known data structures for *2-d orthogonal point location* and *2-d rectangle stabbing emptiness*. The proofs are presented in the full version.

▶ **Lemma 1.** *Given n disjoint axis-aligned rectangles in $[U]^2$ ($n \le U \le 2^w$), there are data structures for point location with $O\left(\frac{n\log U}{w}\right)$ words of space and $O(\log n)$ query time in the pointer machine model, $O(\log_B n)$ query cost in the I/O model, and $O(\min\{\log\log U, \log_w n\})$ query time in the word RAM model.*

▶ **Lemma 2.** *Given n (possibly overlapping) axis-aligned rectangles in $[U]^2$ ($n \le U \le 2^w$), there are data structures for rectangle stabbing emptiness with $O\left(\frac{n\log U}{w}\right)$ words of space and $O(\log n)$ query time in the pointer machine model, $O(\log_B n)$ query cost in the I/O model, and $O(\log_w n)$ query time in the word RAM model.*



$\sqrt{U}$ slabs along x-axis (a) Left/right boxes of a slab (b) Short boxes of a slab (c)

Figure 1 Boxes obtained after partitioning along the x-direction.

Data structure. We are now ready to describe our data structure for 3-d orthogonal point location. We focus on the pointer machine model first. At the beginning, we apply a rank space reduction (replacing input coordinates by their ranks) so that all coordinates are in $[2n]^3$, where n is the global number of input boxes. Given a query point, we can initially find the ranks of its coordinates by three predecessor searches (costing $O(\log n)$ time in the pointer machine model).

We describe our preprocessing algorithm recursively. The input to the preprocessing algorithm is a set of n disjoint boxes that are assumed to be aligned to the $[U_x] \times [U_y] \times [U_z]$ grid. (At the beginning, $U_x = U_y = U_z = 2n$.)

Without loss of generality, assume that $U_x \geq U_y, U_z$. We partition the $[U_x] \times [U_y] \times [U_z]$ grid into $\sqrt{U_x}$ equal-sized vertical slabs perpendicular to the x-direction. See Figure 1. (In the symmetric case $U_y \geq U_x, U_z$ or $U_z \geq U_x, U_y$, we partition along the y- or z-direction instead.) We classify the boxes into two categories:

- *Short boxes.* For each slab, define its short boxes to be those that lie completely inside the slab.
- *Long boxes.* Long boxes intersect the boundary (vertical plane) of at least one slab. Each long box $\mathcal{B}$ is broken into three disjoint boxes:
  - *Left box.* Let s_L be the slab containing the left endpoint (with respect to the x-axis) of $\mathcal{B}$. The left box is defined as $\mathcal{B} \cap s_L$.
  - *Right box.* Let s_R be the slab containing the right endpoint of $\mathcal{B}$. The right box is defined as $\mathcal{B} \cap s_R$.
  - *Middle box.* The remaining portion of box $\mathcal{B}$ after removing its left and right box, i.e. $\mathcal{B} \setminus ((\mathcal{B} \cap s_L) \cup (\mathcal{B} \cap s_R))$.

We build our data structure as follows:

1. *Planar point location structure.* For each slab, we project its left boxes onto the yz-plane. The projected boxes remain disjoint, since they intersect a common boundary. We store them in a data structure for 2-d orthogonal point location by Lemma 1. We do this for the slab's right boxes as well.
2. *Rectangle stabbing structure.* For each slab, we project its short boxes onto the yz-plane. The short boxes are not necessarily disjoint. We store them in a data structure for 2-d rectangle stabbing emptiness by Lemma 2.
3. *Recursive middle structure.* We recursively build a *middle structure* on all the middle boxes.
4. *Recursive short structures.* For each slab, we recursively build a *short structure* on all the short boxes inside the slab.

By translation or scaling, these recursive short structures or middle structure can be made aligned to the $\left[\sqrt{U_x}\right] \times [U_y] \times [U_z]$ grid. In addition, we store the mapping from left/right/middle boxes to their original boxes, as a list of pairs (sorted lexicographically) packed in $O\left(\frac{n \log(U_x U_y U_z)}{w}\right)$ words.

Query algorithm. The following lemma is crucial for deciding whether to query recursively the middle or the short structure.

▶ **Lemma 3.** *Given a query point (q_x, q_y, q_z), if the query with (q_y, q_z) on the rectangle stabbing emptiness structure of the slab that contains q_x returns*

- NON-EMPTY*, then the query point cannot lie inside a box stored in the middle structure, or*
- EMPTY*, then the query point cannot lie inside a box stored in the slab's short structure.*

Proof. If NON-EMPTY is returned, then the query point is stabbed by the extension (along the x-direction) of a box in the slab's short structure and cannot be stabbed by any box stored in the middle structure, because of disjointness of the input boxes. If EMPTY is returned, then obviously the query point cannot lie inside a box stored in the short structure. ◀

To answer a query for a given point (q_x, q_y, q_z), we proceed as follows:

1. Find the slab that contains q_x by predecessor search over the slab boundaries.
2. Query with (q_y, q_z) the planar point location structures at this slab. If a left or a right box returned by the query contains the query point, then we are done.
3. Query with (q_y, q_z) the rectangle stabbing emptiness structure at this slab. If it returns NON-EMPTY, query recursively the slab's short structure, else query recursively the middle structure (after appropriate translation/scaling of the query point).

In step 3, to decode the coordinates of the output box, we need to map from a left/right/middle box to its original box; this can be done naively by another predecessor search in the list of pairs we have stored.

Query time analysis. Let $Q(U_x, U_y, U_z)$ denote the query time for our data structure in the $[U_x] \times [U_y] \times [U_z]$ grid. Observe that the number of boxes n is trivially upper-bounded by $U_x U_y U_z$ because of disjointness. The predecessor search in step 1, the 2-d point location query in step 2, and the 2-d rectangle stabbing query in step 3 all take $O(\log n) = O(\log(U_x U_y U_z))$ time by Lemmata 1 and 2. We thus obtain the following recurrence, assuming that $U_x \geq U_y, U_z$:

$$Q(U_x, U_y, U_z) = Q\left(\sqrt{U_x}, U_y, U_z\right) + O\left(\log\left(U_x U_y U_z\right)\right).$$

If $U_x = U_y = U_z = U$, then three rounds of recursion will partition along the x-, y-, and z-directions and decrease U_x, U_y, and U_z in a round-robin fashion, yielding

$$Q(U, U, U) = Q\left(\sqrt{U}, \sqrt{U}, \sqrt{U}\right) + O(\log U),$$

which solves to $Q(U, U, U) = O(\log U)$. As $U = 2n$ initially, we get $O(\log n)$ query time.

Space analysis. Let $s(U_x, U_y, U_z)$ denote the *amortized* number of words of space needed per input box for our data structure in the $[U_x] \times [U_y] \times [U_z]$ grid. The amortized number of words per input box for the 2-d point location and rectangle stabbing structures is $O\left(\frac{\log(U_x U_y U_z)}{w}\right)$ by Lemmata 1 and 2. We thus obtain the following recurrence, assuming that $U_x \geq U_y, U_z$:

$$s(U_x, U_y, U_z) = s\left(\sqrt{U_x}, U_y, U_z\right) + O\left(\frac{\log(U_x U_y U_z)}{w}\right).$$

Three rounds of recursion yield

$$s(U, U, U) = s\left(\sqrt{U}, \sqrt{U}, \sqrt{U}\right) + O\left(\frac{\log U}{w}\right),$$

which solves to $s(U, U, U) = O\left(\frac{\log U}{w}\right)$. As $U = 2n$ initially, the total space in words is $O\left(n\frac{\log n}{w}\right) \leq O(n)$. Note that the above analysis ignores an overhead of $O(1)$ words of space per node of the recursion tree, but by shortcutting degree-1 nodes, we can bound the number of nodes in the recursion tree by $O(n)$. To summarize, we claim the following results:

▶ **Theorem 4.** *Given n disjoint axis-aligned boxes in 3-d, there are data structures for point location with $O(n)$ words of space and $O(\log n)$ query time in the pointer machine model, $O(\log_B n)$ query cost in the I/O model, and $O(\log_w n)$ query time in the word RAM model.*

Proof. The proof for the I/O model and the word RAM model can be found in the full version. ◀

Further applications of this framework to subdivisions, 4-d and higher dimensions can be found in the full version.

3 Rectangle Stabbing

3.1 Preliminaries

▶ **Lemma 5.** *(Rahul [23]) There is a data structure of size $O(n)$ words which can answer a 5-sided 3-d rectangle stabbing query in $O(\log^2 n \cdot \log\log n + k)$ time.*

▶ **Lemma 6.** *(Leaf structure.) For a set of size $O(w^{1/4})$, there is a data structure of size $O(w^{1/4})$ words which can answer a 5-sided 3-d rectangle stabbing query in $O(1 + k)$ time.*

3.2 3-d 5-sided rectangle stabbing

Skeleton of the structure. Consider the projection of the rectangles of S on to the xy-plane and impose an orthogonal $\left[\!\left[2\sqrt{\frac{n}{\log^4 n}}\right]\!\right] \times \left[\!\left[2\sqrt{\frac{n}{\log^4 n}}\right]\!\right]$ grid such that each horizontal and vertical slab contains the projections of $\sqrt{n \log^4 n}$ sides of S. This grid is the root node of our tree $\mathcal{T}$. For each vertical and horizontal slab, we recurse on the rectangles of S which are *sent* to that slab. At each node of the recursion tree, if we have m rectangles in the subproblem, the grid size changes to $\left[\!\left[2\sqrt{\frac{m}{\log^4 m}}\right]\!\right] \times \left[\!\left[2\sqrt{\frac{m}{\log^4 m}}\right]\!\right]$. We stop the recursion when a node has less than $w^{1/4}$ rectangles.

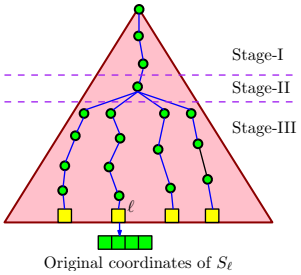


Figure 2

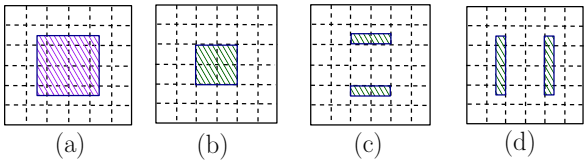


Figure 3

Breaking the rectangles. The solution of Rahul [23] *breaks* only one side to reduce 5-sided rectangles to 4-sided rectangles, and then uses the solution for 4-sided rectangle stabbing as a black box. Unlike the approach of Rahul [23], we will break two sides of each 5-sided rectangle to obtain $O(\log \log n)$ 3-sided rectangles.

For a node in the tree, the intersection of every pair of horizontal and vertical grid line defines a *grid point*. A rectangle $r \in S$ is associated with four root-to-leaf paths (as shown in Figure 2). Any node (say, v) on these four paths is classified w.r.t. r into one of the three stages as follows:

Stage-I. The xy-projection of r intersects none of the grid points. Then r is not stored at v, and sent to the child corresponding to the row or column r lies in.

Stage-II. The xy-projection of r intersects at least one of the grid points. Then r is broken into at most five disjoint pieces. The first piece is a *grid rectangle*, which is the bounding box of all the grid points lying inside r, as shown in Figure 3(b). The remaining four pieces are two *column rectangles* and two *row rectangles* as shown in Figure 3(c) and (d), respectively. The grid rectangle is stored at v. Note that each column rectangle (resp., row rectangle) is now a 4-sided rectangle in $\mathbb{R}^3$ w.r.t. its column (resp., row), and is sent to its corresponding child node.

Stage-III. The xy-projection of a 4-sided piece of r intersects at least one of the grid points. Without loss of generality, assume that the 4-sided rectangle r is unbounded along the negative x-axis. Then the rectangle is broken into at most four disjoint pieces: a *grid rectangle*, two *row rectangles*, and a *column rectangle*, as shown in Figure 4(b), (c) and (d), respectively. The grid rectangle and the two row rectangles are stored at v, and the column rectangle is sent to its corresponding child node. Note that the two row rectangles are now 3-sided rectangles in $\mathbb{R}^3$ w.r.t. their corresponding rows (unbounded in one direction along x-, y- and z-axis).

Encoding structures. Let S_v be the set of rectangles stored at a node v in the tree. We apply a rank space reduction (replacing input coordinates by their ranks) so that the coordinates of all the endpoints are in $[2|S_v|]^3$. If v is a leaf node, then we build an instance of Lemma 6. Otherwise, the following three structures will be built using S_v:

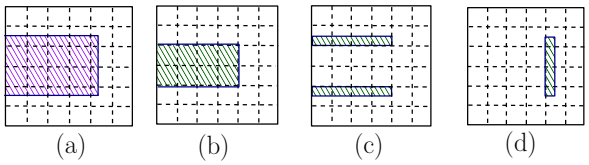


Figure 4

(A) Slow structure. An instance of Lemma 5 is built on S_v to answer the 3-d 5-sided rectangle stabbing query when the output size is "large".
(B) Grid structure. For each cell c of the grid, among the rectangles which completely cover c, pick the $\log^3 |S_v|$ rectangles with the largest span along the z-direction. Store them in a list $Top(c)$ in decreasing order of their span.
(C) 3-d dominance structure. For a given row or column in the grid, based on the 3-sided rectangles stored in it, a linear-space 3-d dominance reporting structure [1, 10] is built. This structure is built for each row and column slab.

Where are the original coordinates stored? Unlike the previous approaches for indexing points [5, 11], we use a somewhat unusual approach for storing the original coordinates of each rectangle. In the process of breaking each 5-sided rectangle described above, there will be four leaf nodes where portions of the rectangle will get stored. We will choose these leaf nodes to store the original coordinates of the rectangle (see Figure 2). The benefit is that each 3-sided rectangle (stored at a node v) has to maintain a decoding pointer of length merely $O(\log |S_v|)$ to point to its original coordinates stored in its subtree.

Query algorithm and analysis. Given a query point q, we start at the root node and perform the following steps: First, query the dominance structure corresponding to the horizontal and the vertical slab containing q. Next, for the grid structure, locate the cell c on the grid containing q. Scan the list $Top(c)$ to keep reporting till (a) all the rectangles have been exhausted, or (b) a rectangle not containing q is found. If case (a) happens and $|Top(c)| = \log^3 |S_v|$, then we discard the rectangles reported till now, and query the slow structure. The decoding pointers will be used to report the original coordinates of the rectangles. Finally, we recurse on the horizontal and the vertical slab containing q. If we visit a leaf node, then we query the leaf structure (Lemma 6).

First, we analyze the space. Let $s(|S_v|)$ be the *amortized* number of bits needed per input 5-sided rectangle in the subtree of a node v. The amortized number of bits needed per rectangle for the encoding structures and the pointers to the original coordinates is $O(\log |S_v|)$. This leads to the following recurrence:

$$s(n) = s(\sqrt{n \log^4 n}) + O(\log n)$$

which solves to $s(n) = O(\log n)$ bits. Therefore, the overall space is bounded by $O(n)$ words.

Next, we analyze the query time. To simplify the analysis, we will exclude the output size term while mentioning the query time. At the root, the time taken to query the grid and the dominance structure is $O(\log \log_w n)$. This leads to the following recurrence:

$$Q(n) = 2Q(\sqrt{n \log^4 n}) + O(\log \log_w n)$$

with a base case of $Q(w^{1/4}) = O(1)$. This solves to $Q(n) = O(\log_w n - \log \log_w n) = O(\log_w n)$. For each reported rectangle it takes constant time to recover its original coordinates. The

time taken to query the slow structure is dominated by the output size. Therefore, the overall query time is $O(\log_w n + k)$.

▶ **Theorem 7.** *There is a data structure of size $O(n)$ words which can answer any 3-d 5-sided rectangle stabbing query in $O(\log_w n + k)$ time. This is optimal in the word RAM model.*

Our solution for 2-d top-k rectangle stabbing can be found in the full version.

3.3 3-d 6-sided rectangle stabbing

The complete discussion on 6-sided rectangle stabbing can be found in the full version. Here we will only highlight the key result.

▶ **Lemma 8.** *There exists an optimal linear-space data structure that answers z-restricted 3-d 4-sided rectangle stabbing queries in $O(\log \log_w n + k)$ time. A z-restricted 4-sided rectangle is of the form $(-\infty, x] \times (-\infty, y] \times [i, j]$, where integers $i, j \in [w^\varepsilon]$ and $\varepsilon = 0.1$.*

Proof. We can safely assume that $n > w^{2\varepsilon} \cdot \log w \log\log n$, because the case of $n < w^{2\varepsilon} \cdot \log w \log\log n = O(w^{1/4})$ can be handled in $O(1+k)$ time by using the structure of Lemma 6. To keep the discussion short, we will assume that $k < \log w \cdot \log\log n$ (handling small values of k is typically more challenging).

Shallow cuttings. A point p_1 is said to *dominate* point p_2 if it has a larger x-coordinate and a larger y-coordinate value. Our main tool to handle this case are *shallow cuttings* which have the following three properties: (a) A t-shallow cutting for a set P of 2-d points is a union of $O(n/t)$ cells where every cell is of the form $[a, +\infty) \times [b, +\infty)$, (b) every point that is dominated by at most t points from P will lie within some cell(s), and (c) each cell contains at most $O(t)$ points of P. A cell $[a, +\infty) \times [b, +\infty)$ can be identified by its corner (a, b). We denote by $Dom(c)$ the set of points that dominate the corner c.

Data structure. We classify rectangles according to their z-projections. The set S_{ij} contains all rectangles of the form $r = (-\infty, x_f] \times (-\infty, y_f] \times [i, j]$. Since $1 \le i \le j \le w^\varepsilon$, there are $O(w^{2\varepsilon})$ sets S_{ij}. Every rectangle r in S_{ij} is associated with a point $p(r) = (x_f, y_f)$. We construct a t-shallow cutting L_{ij} with $t = \log w \cdot \log\log n$ for the set of points $p(r)$, such that $r \in S_{ij}$. A rectangle $r = (-\infty, x_f] \times (-\infty, y_f] \times [i, j]$ is stabbed by a query point $q = (q_x, q_y, q_z)$ if and only if $p(r) \in S_{ij}$ and the point $p(r)$ dominates the 2-d point (q_x, q_y). We can find points of a set S_{ij} that dominate q using the shallow cutting L_{ij}. However, to answer the stabbing query we must simultaneously answer a dominance query on $O(w^{2\varepsilon})$ different sets of points.

We address this problem by *grouping* corners of different shallow cuttings into one structure. Let $\mathcal{C}_{ij}$ denote the set of corners in a shallow cutting L_{ij} and let $\mathcal{C} = \bigcup_{\forall i,j \in [w^\varepsilon]} \mathcal{C}_{ij}$. The set $\mathcal{C}$ is divided into disjoint groups, so that every group G_α consists of $w^{2\varepsilon}$ consecutive corners (with respect to their x-coordinates): for any $c \in G_\alpha$ and $c' \in G_{\alpha+1}$, $c.x < c'.x$. We say that a corner $c \in \mathcal{C}_{ij}$ is immediately to the left of G_α if it is the rightmost corner in $\mathcal{C}_{ij}$ such that $c_x \le c'_x$ for any corner $c' = (c'_x, c'_y)$ in G_α. The set of corners $\overline{G}_\alpha$ contains (1) all corners from G_α, and (2) for every pair i, j such that $1 \le i \le j \le w^\varepsilon$, the corner $c \in \mathcal{C}_{ij}$ immediately to the left of G_α. The set R_α contains all rectangles r such that $p(r) \in Dom(c)$ for each corner $c \in \overline{G}_\alpha$. Since R_α contains $O(w^{2\varepsilon} \log w \cdot \log\log n) = O(w^{1/6})$ rectangles, we can perform a rank-space reduction and answer queries on R_α in $O(k+1)$ time by using Lemma 6.

Next, we will show that the space occupied by this structure is $O(n)$. The crucial observation is that the number of corners in G_α is $w^{2\varepsilon}$ and the number of "immediately left" corners added to each $\overline{G}_\alpha$ is also bounded by $w^{2\varepsilon}$. The number of corners in set $\mathcal{C}$ is bounded by $\sum_{\forall L_{i,j}} O\left(1 + \frac{|S_{ij}|}{t}\right) = O(n/t)$, since $n/t > w^{2\varepsilon}$. Therefore, the number of groups will be $O\left(\frac{n}{tw^{2\varepsilon}}\right)$. Each set R_α contains $O(w^{2\varepsilon}t)$ rectangles. Therefore, the total space occupied by this structure is $\sum_{\forall\alpha} |R_\alpha| = O(\frac{n}{tw^{2\varepsilon}} \cdot w^{2\varepsilon}t) = O(n)$.

Query algorithm. Given a query point $q = (q_x, q_y, q_z)$, we find the set G_α that "contains" q_x. Then we report all the rectangles in R_α that are stabbed by q by using Lemma 6. We need $O(\log\log_w n)$ time to find the group G_α [22] and, then $O(1 + k)$ time to report $R_\alpha \cap q$. ◀

References

1. Peyman Afshani. On dominance reporting in 3D. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 41–51, 2008.
2. Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 240–246, 2010.
3. Peyman Afshani, Lars Arge, and Kasper Green Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 323–332, 2012.
4. Peyman Afshani, Gerth Stølting Brodal, and Norbert Zeh. Ordered and unordered top-k range reporting in large data sets. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 390–400, 2011.
5. Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 198–207, 2000.
6. Lars Arge, Andrew Danner, and Sha-Mayn Teh. I/O-efficient point location using persistent B-trees. *ACM Journal of Experimental Algorithmics*, 8, 2003.
7. J.L. Bentley. Solutions to Klee's rectangle problems. *Technical Report, Carnegie-Mellon University, Pittsburgh, PA*, 1977.
8. Gerth Stølting Brodal. External memory three-sided range reporting and top-k queries with sublogarithmic updates. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 47, pages 23:1–23:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
9. Gerth Stølting Brodal, Rolf Fagerberg, Mark Greve, and Alejandro Lopez-Ortiz. Online sorted range reporting. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 173–182, 2009.
10. Timothy M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. *ACM Transactions on Algorithms*, 9(3):22, 2013.
11. Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 1–10, 2011.
12. Bernard Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal of Computing*, 15(3):703–724, 1986.
13. Mark de Berg, Marc J. van Kreveld, and Jack Snoeyink. Two- and three-dimensional point location in rectangular subdivisions. *Journal of Algorithms*, 18(2):256–277, 1995.
14. Herbert Edelsbrunner, Leonidas J. Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal of Computing*, 15(2):317–340, 1986.

15 Herbert Edelsbrunner, G. Haring, and D. Hilbert. Rectangular point location in d dimensions with applications. *Comput. J.*, 29(1):76–82, 1986.

16 Michael T. Goodrich, Jyh-Jong Tsay, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory computational geometry. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 714–723, 1993.

17 John Iacono and Stefan Langerman. Dynamic point location in fat hyperrectangles with integer coordinates. In *Proceedings of the Canadian Conference on Computational Geometry (CCCG)*, 2000.

18 David G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal of Computing*, 12(1):28–35, 1983.

19 Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM Journal of Computing*, 9(3):615–627, 1980.

20 Yakov Nekrich. I/O-efficient point location in a set of rectangles. In *Latin American Symposium on Theoretical Informatics (LATIN)*, pages 687–698, 2008.

21 Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM Journal of Computing*, 40(3):827–847, 2011.

22 Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 232–240, 2006.

23 Saladi Rahul. Improved bounds for orthogonal point enclosure query and point location in orthogonal subdivisions in $\mathbb{R}^3$. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 200–211, 2015.

24 Saladi Rahul and Ravi Janardan. A general technique for top-k geometric intersection query problems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 26(12):2859–2871, 2014.

25 Saladi Rahul and Yufei Tao. On top-k range reporting in 2d space. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 265–275, 2015.

26 Saladi Rahul and Yufei Tao. Efficient top-k indexing via general reductions. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, 2016.

27 Neil Sarnak and Robert Endre Tarjan. Planar point location using persistent search trees. *Communications of the ACM (CACM)*, 29(7):669–679, 1986.

28 Cheng Sheng and Yufei Tao. Dynamic top-k range reporting in external memory. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, 2012.

29 Qingmin Shi and Joseph JáJá. Novel transformation techniques using q-heaps with applications to computational geometry. *SIAM Journal of Computing*, 34(6):1474–1492, 2005.

30 Jack Snoeyink. Point location. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 767–787. CRC Press, 2nd edition, 2004.

31 Yufei Tao. A dynamic I/O-efficient structure for one-dimensional top-k range reporting. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 256–265, 2014.

Spanning Tree Congestion and Computation of Generalized Győri-Lovász Partition

L. Sunil Chandran[1]

Department of Computer Science and Automation, Indian Institute of Science, India
sunil@csa.iisc.ernet.in

Yun Kuen Cheung[2]

Max Planck Institute for Informatics, Saarland Informatics Campus, Germany
ycheung@mpi-inf.mpg.de
https://orcid.org/0000-0002-9280-0149

Davis Issac

Max Planck Institute for Informatics, Saarland Informatics Campus, Germany
dissac@mpi-inf.mpg.de
https://orcid.org/0000-0001-5559-7471

Abstract

We study a natural problem in graph sparsification, the Spanning Tree Congestion (STC) problem. Informally, it seeks a spanning tree with no tree-edge *routing* too many of the original edges.

For any general connected graph with n vertices and m edges, we show that its STC is at most $\mathcal{O}(\sqrt{mn})$, which is asymptotically optimal since we also demonstrate graphs with STC at least $\Omega(\sqrt{mn})$. We present a polynomial-time algorithm which computes a spanning tree with congestion $\mathcal{O}(\sqrt{mn} \cdot \log n)$. We also present another algorithm for computing a spanning tree with congestion $\mathcal{O}(\sqrt{mn})$; this algorithm runs in sub-exponential time when $m = \omega(n \log^2 n)$.

For achieving the above results, an important intermediate theorem is *generalized Győri-Lovász theorem*. Chen et al. [8] gave a non-constructive proof. We give the first elementary and constructive proof with a local search algorithm of running time $\mathcal{O}^*(4^n)$. We discuss some consequences of the theorem concerning graph partitioning, which might be of independent interest.

We also show that for any graph which satisfies certain *expanding properties*, its STC is at most $\mathcal{O}(n)$, and a corresponding spanning tree can be computed in polynomial time. We then use this to show that a random graph has STC $\Theta(n)$ with high probability.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

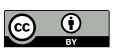
Keywords and phrases Spanning Tree Congestion, Graph Sparsification, Graph Partitioning, Min-Max Graph Partitioning, k-Vertex-Connected Graphs, Győri-Lovász Theorem

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.32

Related Version A full version of this paper is available at `https://arxiv.org/abs/1802.07632`.

[1] This work was done while this author was visiting Max Planck Institute for Informatics, Saarbrücken, Germany, supported by Alexander von Humboldt Fellowship.

[2] Part of the work done while this author was a visitor at the Courant Institute, NYU. The visit was funded in part by New York University.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 32; pp. 32:1–32:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Graph Sparsification/Compression generally describes a transformation of a *large* input graph into a *smaller/sparser* graph that preserves certain feature (e.g., distance, cut, congestion, flow) either exactly or approximately. The algorithmic value is clear, since the smaller graph might be used as a preprocessed input to an algorithm, so as to reduce subsequent running time and memory requirement. In this paper, we study a natural problem in graph sparsification, the Spanning Tree Congestion (STC) problem. Informally, the STC problem seeks a spanning tree with no tree-edge *routing* too many of the original edges. The problem is well-motivated by network design applications, where designers aim to build sparse networks that meet traffic demands, while ensuring no connection (edge) is too congested. Indeed, the root of this problem dates back to at least 30 years ago under the name of "load factor" [5, 25], with natural motivations from parallel computing and circuit design applications. The STC problem was formally defined by Ostrovskii [21] in 2004, and since then a number of results have been presented.

Two canonical goals for graph sparsification problems are to understand the trade-off between the sparsity of the output graph(s) and how well the feature is preserved, and to devise (efficient) algorithms for computing the sparser graph(s). These are also our goals for the STC problem. We focus on two scenarios: (A) general connected graphs with n vertices and m edges, and (B) graphs which exhibit certain *expanding properties*.

For (A), we show that the spanning tree congestion (STC) is at most $\mathcal{O}(\sqrt{mn})$, which is a factor of $\Omega(\sqrt{m/n})$ better than the trivial bound of m. We present a polynomial-time algorithm which computes a spanning tree with congestion $\mathcal{O}(\sqrt{mn} \cdot \log n)$. We also present another algorithm for computing a spanning tree with congestion $\mathcal{O}(\sqrt{mn})$; this algorithm runs in sub-exponential time when $m = \omega(n \log^2 n)$. For almost all ranges of average degree $2m/n$, we also demonstrate graphs with STC at least $\Omega(\sqrt{mn})$. For (B), we show that the expanding properties permit us to devise polynomial-time algorithm which computes a spanning tree with congestion $\mathcal{O}(n)$. Using this result, together with a separate lower-bound argument, we show that a random graph has $\Theta(n)$ STC with high probability.

For achieving the results for (A), an important intermediate theorem is *generalized Győri-Lovász theorem*, which was first proved by Chen et al. [8]. Their proof uses advanced techniques in topology and homology theory, and is non-constructive. For brevity, we will say "k-connected" for "k-vertex-connected" henceforth.

▶ **Definition 1.** In a graph $G = (V, E)$, a k-connected-partition is a k-partition of V into $\cup_{j=1}^{k} V_j$, such that for each $j \in [k]$, $G[V_j]$ is connected.

▶ **Theorem 2** ([8, Theorems 25, 26])**.** *Let $G = (V, E)$ be a k-connected graph. Let w be a weight function $w : V \to \mathbb{R}^+$. For any $U \subset V$, let $w(U) := \sum_{v \in U} w(v)$. Given any k distinct terminal vertices $t_1, \cdots, t_k$, and k positive integers $T_1, \cdots, T_k$ such that for each $j \in [k]$, $T_j \geq w(t_j)$ and $\sum_{i=1}^{k} T_i = w(V)$, there exists a k-connected-partition of V into $\cup_{j=1}^{k} V_j$, such that for each $j \in [k]$, $t_j \in V_j$ and $w(V_j) \leq T_j + \max_{v \in V} w(v) - 1$.*

One of our main contributions is to give the first elementary and constructive proof by providing a *local search* algorithm with running time $\mathcal{O}^*(4^n)$.

▶ **Theorem 3.** *(a) There is an algorithm which given a k-connected graph, computes a k-connected-partition satisfying the conditions stated in Theorem 2 in time $\mathcal{O}^*(4^n)$.*
(b) If we need a $(\lfloor k/2 \rfloor + 1)$-partition instead of k-partition (the input graph remains assumed to be k-connected), the algorithm's running time improves to $\mathcal{O}^*(2^{\mathcal{O}((n/k)\log k)})$.

We make three remarks. First, the $\mathcal{O}^*(2^{\mathcal{O}((n/k)\log k)})$-time algorithm is a key ingredient of our algorithm for computing a spanning tree with congestion $\mathcal{O}(\sqrt{mn})$. Second, since Theorem 2 guarantees the existence of such a partition, the problem of computing such a partition is not a *decision problem* but a *search problem*. Our algorithm shows that this problem is in the complexity class PLS [12]; we raise its completeness in PLS as an open problem. Third, the running times do not depend on the weights.

The STC Problem, Related Problems and Our Results. Given a connected graph $G = (V, E)$, let T be a spanning tree. For an edge $e = (u, v) \in E$, its detour with respect to T is the unique path from u to v in T; let $\mathsf{DT}(e, T)$ denote the set of edges in this detour. The stretch of e with respect to T is $|\mathsf{DT}(e, T)|$, the length of its detour. The dilation of T is $\max_{e\in E} |\mathsf{DT}(e, T)|$. The edge-congestion of an edge $e \in T$ is $\mathsf{ec}(e, T) := |\{f \in E : e \in \mathsf{DT}(f, T)\}|$, i.e., the number of edges in E whose detours contain e. The congestion of T is $\mathsf{cong}(T) := \max_{e\in T} \mathsf{ec}(e, T)$. The spanning tree congestion (STC) of the graph G is $\mathsf{STC}(G) := \min_T \mathsf{cong}(T)$, where T runs over all spanning trees of G.

We note that there is an equivalent cut-based definition for edge-congestion, which we will use in our proofs. Given a connected graph $G = (V, E)$, an edge set $F \subseteq E$ and two disjoint vertex subsets $V_1, V_2 \subset V$, we let $F(V_1, V_2) := \{ e = \{v_1, v_2\} \in F \mid v_1 \in V_1 \text{ and } v_2 \in V_2 \}$. For each tree-edge $e \in T$, removing e from T results in two connected components; let U_e denote one of the components. Then $\mathsf{ec}(e, T) := |E(U_e, V \setminus U_e)|$.

Various types of congestion, stretch and dilation problems are studied in computer science and discrete mathematics; see the survey [24] for more details. In these problems, one typically seeks a spanning tree (or some other structure) with minimum congestion, stretch or dilation. Among them, the most famous one is the Low Stretch Spanning Tree (LSST) problem, which seeks a spanning tree which minimizes the total stretch of all the edges of G [3]. It is easy to see that minimizing the total stretch is equivalent to minimizing the total edge-congestion of the selected spanning tree. Several strong results were published about the LSST problem. Alon et al. [3] had shown a lower bound of $\Omega(\max\{n \log n, m\})$. Upper bounds have been derived and many efficient algorithms have been devised; the current best upper bound is $\tilde{\mathcal{O}}(m \log n)$. [3, 9, 1, 14, 2] Since total stretch is identical to total edge-congestion, the best upper bound for the LSST problem automatically implies an $\tilde{\mathcal{O}}(\frac{m}{n} \log n)$ upper bound on the *average* edge-congestion. But in the STC problem, we concern the *maximum* edge-congestion.

In comparison, there were not many strong and general results for the STC Problem, though it was studied extensively in the past 13 years. The problem was formally proposed by Ostrovskii [21] in 2004. Prior to this, Simonson [25] had studied the same parameter under a different name to approximate the cut width of outer-planar graph. A number of graph-theoretic results were presented on this topic [22, 17, 16, 15, 7]. Some complexity results were also presented recently [20, 6], but most of these results concern special classes of graphs. The most general result regarding STC of general graphs is an $\mathcal{O}(n\sqrt{n})$ upper bound by Löwenstein, Rautenbach and Regen in 2009 [19], and a matching lower bound by Ostrovskii in 2004 [21]. Note that the above upper bound is not interesting when the graph is sparse, since there is also a trivial upper bound of m. In this paper we come up with a strong improvement to these bounds after 8 years:

Theorem (informal): *For a connected graph G with n vertices and m edges, its spanning tree congestion is at most $\mathcal{O}(\sqrt{mn})$. In terms of average degree $d_{avg} = 2m/n$, we can state this upper bound as $\mathcal{O}(n\sqrt{d_{avg}})$. There is a matching lower bound.*

Our proof for achieving the $\mathcal{O}(\sqrt{mn})$ upper bound is constructive. It runs in exponential time in general; for graphs with $m = \omega(n \log^2 n)$ edges, it runs in sub-exponential time. By using an algorithm of Chen et al. [8] for computing *single-commodity confluent flow* from single-commodity splittable flow, we improve the running time to polynomial, but with a slightly worse upper bound guarantee of $\mathcal{O}(\sqrt{mn} \cdot \log n)$.

Motivated by an open problem raised by Ostrovskii [23] concerning STC of random graphs, we formulate a set of *expanding properties*, and prove that for any graph satisfying these properties, its STC is at most $\mathcal{O}(n)$. We devise a polynomial time algorithm for computing a spanning tree with congestion $\mathcal{O}(n)$ for such graphs. This result, together with a separate lower-bound argument, permit us to show that for random graph $\mathcal{G}(n,p)$ with $1 \geq p \geq \frac{c \log n}{n}$ for some small constant $c > 1$, its STC is $\Theta(n)$ with high probability, thus resolving the open problem raised by Ostrovskii completely.

Min-Max Graph Partitioning and the Generalized Győri-Lovász Theorem. It looks clear that the powerful Theorem 2 can make an impact on graph partitioning. We discuss a number of its consequences which might be of wider interest.

Graph partitioning/clustering is a prominent topic in graph theory/algorithms, and has a wide range of applications. A popular goal is to partition the vertices into sets such that the number of edges across different sets is *small*. While the *min-sum* objective, i.e., minimizing the total number of edges across different sets, is more widely studied, in various applications, the more natural objective is the *min-max* objective, i.e., minimizing the maximum number of edges leaving each set. The min-max objective is our focus here.

Depending on applications, there are additional constraints on the sets in the partition. Two natural constraints are (i) balancedness: the sets are (approximately) balanced in sizes, and (ii) induced-connectivity: each set induces a connected subgraph. The balancedness constraint appears in the application of *domain decomposition* in parallel computing, while the induced-connectivity constraint is motivated by divide-and-conquer algorithms for spanning tree construction. Imposing both constraints simultaneously is not feasible for every graph; for instance, consider the star graph with more than 6 vertices and one wants a 3-partition. Thus, it is natural to ask, for which graphs do partitions satisfying both constraints exist. Theorem 2 implies a simple sufficient condition for existence of such partitions.

By setting the weight of each vertex in G to be its degree, and using the fact that (the maximum degree) $\leq n \leq \frac{2m}{k}$ for any k-connected graph on n vertices and m edges, we have

▶ **Proposition 4.** *If G is a k-connected graph with m edges, then there exists a k-connected-partition, such that the total degree of vertices in each part is at most $4m/k$. Consequently, the min-max objective is also at most $4m/k$.*

Due to expander graphs, this bound is optimal up to a small constant factor. This proposition (together with Lemma 9) implies the following crucial lemma for achieving some of our results.

▶ **Lemma 5.** *Let G be a k-connected graph with m edges. Then $\mathsf{STC}(G) \leq 4m/k$.*

Proposition 4 can be generalized to include approximate balancedness in terms of number of vertices. By setting the weight of each vertex to be cm/n plus its degree in G, we have

▶ **Proposition 6.** *Given any fixed $c > 0$, if G is a k-connected graph with m edges and n vertices, then there exists a k-connected-partition such that the total degree of vertices in each part is at most $(2c+4)m/k$, and the number of vertices in each part is at most $\frac{2c+4}{c} \cdot \frac{n}{k}$.*

Further Related Work. Concerning STC problem, Okamoto et al. [20] gave an $\mathcal{O}^*(2^n)$ algorithm for computing the exact STC of a graph. For graph partitioning, Kiwi, Spielman and Teng [13] formulated the min-max k-partitioning problem and gave bounds for classes of graphs with *small separators*, which were then improved by Steurer [26]. On the algorithmic side, the min-sum objective has been extensively studied; the min-max objective, while striking as the more natural objective in some applications, has received much less attention. The only algorithmic work on this objective (and its variants) are Svitkina and Tardos [28] and Bansal et al. [4]. None of the above work addresses the induced-connectivity constraint.

The classical version of Győri-Lovász Theorem (i.e., the vertex weights are uniform) was proved independently by Győri [10] and Lovász [18]. Lovász used homology theory and is non-constructive. Győri's proof is elementary and is constructive implicitly, but he did not analyze the running time. Polynomial time algorithms for constructing the k-partition were devised for $k = 2, 3$ [27, 29], but no non-trivial finite-time algorithm was known for general graphs with $k \geq 4$. Recently, Hoyer and Thomas [11] provided a clean presentation of Győri's proof by introducing their own terminology, which we use for our constructive proof.

In the full version, we discuss some other related work and some interesting open problems.

2 Technical Overview

To prove the generalized Győri-Lovász theorem, we follow the same framework of Győri's proof [10], and we borrow terminology from the recent presentation by Hoyer and Thomas [11]. But it should be emphasized that proving our generalized theorem is not straight-forward, since in Győri's proof, at each stage a single vertex is moved from one set to other to make progress, while making sure that the former set remains connected. In our setting, in addition to this we also have to ensure that the weights in the partitions do not exceed the specified limit; and hence any vertex that can be moved from one set to another need *not* be candidate for being transferred. The proof of the theorem is presented in Section 3.

As discussed, a crucial ingredient for our upper bound results is Lemma 5, which is a direct corollary of the generalized Győri-Lovász theorem. The lemma takes care of the highly-connected cases; for other cases we provide a recursive way to construct a low congestion spanning tree. See Section 4 for details. For showing our lower bound for general graphs, the challenge is to maintain high congestion while keeping density small. To achieve this, we combine three expander graphs with *little* overlapping between them, and we further make those overlapped vertices of very high degree. This will force a tree-edge adjacent to the centroid of the tree to have high congestion. See Section 5 for details.

We formulate a set of expanding properties which permit constructing a spanning tree of better congestion guarantee in polynomial time. The basic idea is simple: start with a vertex v of high degree as the root, then try to grow the tree by keep attaching new vertices to it, while keeping the invariant that the subtrees rooted at each of the neighbours of v are roughly balanced in size; each such subtree is called a *branch*. But when trying to grow the tree in a balanced way, we will soon realize that as the tree grows, all the remaining vertices might be adjacent only to a few "heavy" branches. To help the balanced growth, our algorithm identifies a *transferable* vertex in a "heavy" branch, and it and its descendants in the tree can be transferred to a "lighter" branch. Another technique is to use multiple rounds of matching between vertices in the tree and the remaining vertices to attach new vertices to the tree. This will tend to make sure that all subtrees do not grow uncontrolled. By showing that random graph satisfies the expanding properties with appropriate parameters, we show that its STC is $\Theta(n)$ with high probability. These results are formally presented in Section 6; see the full version for the complete algorithm and its analysis.

3 Generalized Győri-Lovász Theorem

Let $G(V, E)$ be a k-connected graph on n vertices and m edges, and $w : V \to \mathbb{R}^+$ be a weight function. For any subset $U \subseteq V$, we write $w(U) := \sum_{u \in U} w(u)$. Let $w_{max} := \max_{v \in V} w(v)$. We prove Theorem 3 in this section. Observe that the classical Győri-Lovász Theorem follows from Theorem 3 by taking $w(v) = 1$ for all $v \in V$ and $T_j = n_j$ for all $j \in [k]$. We note that a perfect generalization where one requires that $w(V_j) = T_j$ is not possible – think when all vertex weights are even integers, while some T_j is odd.

3.1 Key Combinatorial Notions

We first highlight the key combinatorial notions used for proving Theorem 3.

Fitted Partial Partition. First, we introduce the notion of *fitted partial partition* (FPP). An FPP A is a tuple of k subsets of V, $(A_1, \ldots, A_k)$, such that the k subsets are pairwise disjoint, and for each $j \in [k]$:

1. $t_j \in A_j$ and $G[A_j]$ is connected, and
2. $w(A_j) \leq T_j + w_{max} - 1$ (we say the set is *fitted* for satisfying this inequality).

We say an FPP is a *Strict Fitted Partial Partition* (SFPP) if $A_1 \cup \cdots \cup A_k$ is a proper subset of V. We say the set A_j is *light* if $w(A_j) < T_j$, and we say it is *heavy* otherwise. Note that there exists at least one light set in any SFPP, for otherwise $w(A_1 \cup \cdots \cup A_k) \geq \sum_{j=1}^{k} T_j = w(V)$, which means $A_1 \cup \cdots \cup A_k = V$. Also note that by taking $A_j = \{t_j\}$, we have an FPP, and hence at least one FPP exists.

Configuration. For a set A_j in an FPP A and a vertex $v \in A_j \setminus \{t_j\}$, we define the reservoir of v with respect to A, denoted by $R_A(v)$, as the vertices in the same connected component as t_j in $G[A_j] \setminus \{v\}$. Note that $v \notin R_A(v)$.

For a *heavy* set A_j, a sequence of vertices $(z_1, \ldots, z_p)$ for some $p \geq 0$ is called a cascade of A_j if $z_1 \in A_j \setminus \{t_j\}$ and $z_{i+1} \in A_j \setminus R_A(z_i)$ for all $1 \leq i < p$. The cascade is called a null cascade if $p = 0$, i.e., if the cascade is empty. Note that for light set, we do *not* need to define its cascade since we do not use it in the proof.

A configuration $\mathcal{C}_A$ is defined as a pair (A, D), where $A = (A_1, \cdots, A_k)$ is an FPP, and D is a set of cascades, which consists of exactly one cascade (possibly, a null cascade) for each heavy set in A. A vertex in some cascade of the configuration is called a cascade vertex.

Given a configuration, we define rank and level inductively as follows. Any vertex in a light set is said to have level 0. For $i \geq 0$, a cascade vertex is said to have rank $i + 1$ if it has an edge to a level-i vertex but does not have an edge to any level-i' vertex for $i' < i$. A vertex u is said to have level i, for $i \geq 1$, if $u \in R_A(v)$ for some rank-i cascade vertex v, but $u \notin R_A(w)$ for any cascade vertex w such that rank of w is less than i. A vertex that is not in $R_A(v)$ for any cascade vertex v is said to have level ∞.

A configuration is called a valid configuration if for each heavy set A_j, rank is defined for each of its cascade vertices and the rank is strictly increasing in the cascade, i.e., if $\{z_1, \ldots, z_p\}$ is the cascade, then $\mathsf{rank}(z_1) < \cdots < \mathsf{rank}(z_p)$. Note that by taking $A_j = \{t_j\}$ and taking the null cascade for each heavy set (in this case A_j is heavy if $w(t_j) = T_j$), we get a valid configuration.

Configuration Vectors and Their Total Ordering. For any vertex, we define its neighborhood level as the smallest level of any vertex adjacent to it. A vertex v of level ℓ is said to satisfy maximality property if each vertex adjacent on it is either a rank-$(\ell+1)$ cascade vertex, has a level of at most $\ell+1$, or is one of the terminals t_j for some j. For any $\ell \geq 0$, a valid configuration is called an ℓ-maximal configuration if all vertices having level at most $\ell-1$ satisfy the maximality property. Note that by definition, any valid configuration is a 0-maximal configuration.

For a configuration $\mathcal{C}_A = ((A_1, \ldots, A_k), D)$, we define $S_A := V \setminus (A_1 \cup \cdots \cup A_k)$. An edge uv is said to be a bridge in $\mathcal{C}_A$ if $u \in S_A$, $v \in A_j$ for some $j \in [k]$, and $level(v) \neq \infty$.

A valid configuration $\mathcal{C}_A$ is said to be ℓ-good if the highest rank of a cascade vertex in $\mathcal{C}_A$ is exactly ℓ (if there are no cascade vertices, then we take the highest rank as 0), $\mathcal{C}_A$ is ℓ-maximal, and all bridges uv in $\mathcal{C}_A$ (if any) are such that $u \in S_A$ and $level(v) = \ell$. Note that taking $A_j = \{t_j\}$ and taking the null cascade for each heavy set gives a 0-good configuration.

For each configuration $\mathcal{C}_A = (A, D)$, we define a configuration vector as below:

$$(L_A \ , \ N_A^0 \ , \ N_A^1 \ , \ N_A^2 \ , \ \ldots \ , \ N_A^n \ ),$$

where L_A is the number of light sets in A, and N_A^ℓ is the number of all level-ℓ vertices in $\mathcal{C}_A$.

Next, we define ordering on configuration vectors. Let $\mathcal{C}_A$ and $\mathcal{C}_B$ be configurations. We say $\mathcal{C}_A >_0 \mathcal{C}_B$ if

- $L_A < L_B$, or
- $L_A = L_B$, and $N_A^0 > N_B^0$.

We say $\mathcal{C}_A =_0 \mathcal{C}_B$ if $L_A = L_B$ and $N_A^0 = N_B^0$. We say $\mathcal{C}_A \geq_0 \mathcal{C}_B$ if $\mathcal{C}_A =_0 \mathcal{C}_B$ or $\mathcal{C}_A >_0 \mathcal{C}_B$. We say $\mathcal{C}_A =_\ell \mathcal{C}_B$ if $L_A = L_B$, and $N_A^{\ell'} = N_B^{\ell'}$ for all $\ell' \leq \ell$.

For $1 \leq \ell \leq n$, we say $\mathcal{C}_A >_\ell \mathcal{C}_B$ if

- $\mathcal{C}_A >_{\ell-1} \mathcal{C}_B$, or
- $\mathcal{C}_A =_{\ell-1} \mathcal{C}_B$, and $N_A^\ell > N_B^\ell$.

We say $\mathcal{C}_A \geq_\ell \mathcal{C}_B$ if $\mathcal{C}_A =_\ell \mathcal{C}_B$ or $\mathcal{C}_A >_\ell \mathcal{C}_B$. We say $\mathcal{C}_A > \mathcal{C}_B$ ($\mathcal{C}_A$ is *strictly better* than $\mathcal{C}_B$) if $\mathcal{C}_A >_n \mathcal{C}_B$.

3.2 Proof of Theorem 3(a)

We use two technical lemmas about configuration vectors and their orderings to prove Theorem 3(a). The proof of Theorem 3(b) follows closely with the proof of Theorem 3(a), but makes use of an observation to give an improved bound on the number of configuration vectors navigated by the local search algorithm.

▶ **Lemma 7.** *Given any ℓ-good configuration $\mathcal{C}_A = (A = (A_1, \ldots, A_k), D_A))$ that does not have a bridge, we can find an $(\ell+1)$-good configuration $\mathcal{C}_B = (B = (B_1, B_2, \ldots, B_k), D_B)$ in polynomial time such that $\mathcal{C}_B > \mathcal{C}_A$.*

▶ **Lemma 8.** *Given an ℓ-good configuration $\mathcal{C}_A = (A = (A_1, \ldots, A_k), D_A)$ having a bridge, we can find in polynomial time a valid configuration $\mathcal{C}_B = (B = (B_1, \ldots, B_k), D_B)$ such that one of the following holds:*

- *$\mathcal{C}_B >_\ell \mathcal{C}_A$, and $\mathcal{C}_B$ is an ℓ-good configuration, or*
- *$\mathcal{C}_B \geq_{\ell-1} \mathcal{C}_A$, there is a bridge $u'v'$ in $\mathcal{C}_B$ such that $u' \in S_B$ and $level(v') \leq \ell - 1$, and $\mathcal{C}_B$ is an $(\ell-1)$-good configuration.*

Proof of Theorem 3(a): We always maintain a configuration $\mathcal{C}_A = (A, D_A)$ that is ℓ-good for some $\ell \geq 0$. If the FPP A is not an SFPP at any point, then we are done. So assume A is an SFPP.

We start with the 0-good configuration where $A_j = \{t_j\}$ and the cascades of all heavy sets are null cascades. If our current configuration $\mathcal{C}_A$ is an ℓ-good configuration that has no bridge, then we use Lemma 7 to get a configuration $\mathcal{C}_B$ such that $\mathcal{C}_B > \mathcal{C}_A$ and B is $(\ell+1)$-good. We take $\mathcal{C}_B$ as the new current configuration $\mathcal{C}_A$. If our current configuration $\mathcal{C}_A$ is an ℓ-good configuration with a bridge, then we get an ℓ'-good configuration $\mathcal{C}_B$ for some $\ell' \geq 0$ such that $\mathcal{C}_B > \mathcal{C}_A$ by repeatedly applying Lemma 8 at most ℓ times. So in either case, we get a strictly better configuration that is ℓ'-good for some $\ell' \geq 0$ in polynomial time. We call this an iteration of our algorithm.

Notice that the number of iterations possible is at most the number of distinct configuration vectors possible. It is easy to see that the number of distinct configuration vectors with highest rank at most r is at most $\binom{n+r-1}{n}$. Since rank of any point is at most n, the number of iterations of our algorithm is at most $(k+1) \cdot \binom{2n}{n} \leq n \cdot 4^n$. Since each iteration runs in polynomial time as guaranteed by the two lemmas, the required running time is $\mathcal{O}^*(4^n)$.

When the algorithm terminates, the FPP given by the current configuration is not an SFPP and this gives the required partition.

Proof of Lemma 7: Since $\mathcal{C}_A$ is ℓ-maximal, any vertex that is at level $\ell' < \ell$ satisfies maximality property. So, for satisfying $(\ell+1)$-maximality, we only need to worry about the vertices that are at level ℓ. Let X_j be the set of all vertices $x \in A_j$ such that x is adjacent to a level-ℓ vertex, $level(x) \geq \ell + 1$ (i.e., $level(x) = \infty$ as the highest rank of any cascade vertex is ℓ), $x \neq t_j$, and x is not a cascade vertex of rank ℓ.

We claim that there exists at least one j for which X_j is not empty. If that is not the case, then we exhibit a cut set of size at most $k-1$. For each j such that A_j is a heavy set with a non-null cascade, let y_j be the highest ranked cascade vertex in A_j. For each j such that A_j is a heavy set with a null cascade, let y_j be t_j. Let Y be the set of all y_j such that A_j is a heavy set. Note that $|Y| \leq k-1$ as A is an SFPP and hence has at least one light set. Let Z_∞ be the set of all vertices in $V \setminus Y$ that have level ∞ and Z be the remaining vertices in $V \setminus Y$. Since A is an SFPP, $S_A \neq \emptyset$, and since all vertices in S_A have level ∞, we have that $Z_\infty \neq \emptyset$. Z is not empty because there exists at least one light set in A and the vertices in a light set have level 0. We show that there is no edge between Z_∞ and Z in G. Suppose there exists an edge uv such that $u \in Z_\infty$ and $v \in Z$. If $u \in S_A$, then uv is a bridge which is a contradiction by our assumption that $\mathcal{C}_A$ does not have a bridge. Hence $u \in A_j$ for some $j \in [k]$. Note that A_j has to be a heavy set, otherwise u has level 0. We have that u is not a cascade vertex (as all cascade vertices with level ∞ are in Y) and $u \neq t_j$ (as all t_j such that $level(t_j) = \infty$ are in Y). Also, v is not of level ℓ as otherwise, $u \in X_j$ but we assumed X_j is empty. But then, v has level at most $\ell - 1$, u has level ∞, and there is an edge uv. This means that $\mathcal{C}_A$ was not ℓ-maximal, which is a contradiction. Thus, there exists at least one j for which X_j is not empty.

For any j such that $X_j \neq \emptyset$, there is at least one vertex x_j such that $X_j \setminus \{x_j\} \subseteq R_A(x_j)$. Now we give the configuration $\mathcal{C}_B$ as follows. We set $B_j = A_j$ for all $j \in [k]$. For each heavy set A_j such that $X_j \neq \emptyset$, we take the cascade of B_j as the cascade of A_j appended with x_j. For each heavy set A_j such that $X_j = \emptyset$, we take the cascade of B_j as the cascade of A_j. It is easy to see that $\mathcal{C}_B$ is $(\ell+1)$-maximal as each vertex that had an edge to level-ℓ vertices in $\mathcal{C}_A$ is now either a rank $\ell+1$ cascade vertex or a level-$(\ell+1)$ vertex or is t_j for some j.

Also, notice that all the new cascade vertices that we introduce (i.e., the x_j's) have their rank as $\ell+1$ and there is at least one rank $\ell+1$ cascade vertex as X_j is not empty for some j. Since there were no bridges in $\mathcal{C}_A$, all bridges in $\mathcal{C}_B$ has to be from S_B to a vertex having level $\ell+1$. Hence, $\mathcal{C}_B$ is $(\ell+1)$-good. All vertices that had level at most ℓ in $\mathcal{C}_A$ retained their levels in $\mathcal{C}_B$. And, at least one level-∞ vertex of $\mathcal{C}_A$ became a level-$(\ell+1)$ vertex in $\mathcal{C}_B$ because the cascade vertex that was at rank ℓ becomes level-$(\ell+1)$ vertex now in at least one set. Since $\mathcal{C}_A$ had no level-$(\ell+1)$ vertices, this means that $\mathcal{C}_B > \mathcal{C}_A$.

Proof of Lemma 8: Let uv be a bridge where $u \in S_A$. Let A_{j^*} be the set containing v. Note that $level(v) = \ell$ because $\mathcal{C}_A$ is ℓ-good. We keep $B_j = A_j$ for all $j \neq j^*$. But we modify A_{j^*} to get B_{j^*} as described below. We maintain that if A_j is a heavy set then B_j is also a heavy set for all j, and hence maintain that $L_B \leq L_A$.

Case 1: A_{j^*} is a light set (i.e., when $\ell = 0$). We take $B_{j^*} = A_{j^*} \cup \{u\}$. For all j such that B_j is a heavy set, cascade of B_j is taken as the null cascade. We have $w(A_{j^*}) \leq T_j - 1$ because A_{j^*} is a light set. So, $w(B_{j^*}) = w(A_{j^*}) + w(u) \leq (T_j - 1) + w_{max}$, and hence B_{j^*} is fitted. Also, $G[B_{j^*}]$ is connected and hence $(B_1, \ldots, B_k)$ is an FPP. We have $\mathcal{C}_B >_0 \mathcal{C}_A$ because either B_{j^*} became a heavy set in which case $L_B < L_A$, or it is a light set in which case $L_B = L_A$ and $N^0_B > N^0_A$. It is easy to see that $\mathcal{C}_B$ is 0-good.

Case 2: A_{j^*} is a heavy set i.e., when $\ell \geq 1$.

Case 2.1: $w(A_{j^*} \cup \{u\}) \leq T_j + w_{max} - 1$. We take $B_{j^*} = A_{j^*} \cup \{u\}$. For each j such that B_j is a heavy set (A_j is also heavy set for such j), the cascade of B_j is taken as the cascade of A_j. $G[B_{j^*}]$ is clearly connected and B_{j^*} is fitted by assumption of the case that we are in. Hence B is indeed an FPP. Observe that all vertices that had level $\ell' \leq \ell$ in $\mathcal{C}_A$ still has level ℓ' in $\mathcal{C}_B$. Since $level(v)$ was ℓ in $\mathcal{C}_A$ by ℓ-goodness of $\mathcal{C}_A$, u also has level ℓ in $\mathcal{C}_B$; and u had level ∞ in $\mathcal{C}_A$. Hence, $\mathcal{C}_B >_\ell \mathcal{C}_A$. It is also easy to see that $\mathcal{C}_B$ remains ℓ-good.

Case 2.2: $w(A_{j^*} \cup \{u\}) \geq T_j + w_{max}$. Let z be the cascade vertex of rank ℓ in A_{j^*}. Note that A_{j^*} should have such a cascade vertex as $v \in A_{j^*}$ has level ℓ. Let $\bar{R}$ be $A_{j^*} \setminus (R_A(z) \cup z)$, i.e., $\bar{R}$ is the set of all vertices in $A_{j^*} \setminus \{z\}$ with level ∞. We initialize $B_{j^*} := A_{j^*} \cup \{u\}$. Now, we delete vertices one by one from B_{j^*} in a specific order until B_{j^*} becomes fitted. We choose the order of deleting vertices such that $G[B_{j^*}]$ remains connected. Consider a spanning tree τ of $G[\bar{R} \cup \{z\}]$. τ has at least one leaf, which is not z. We delete this leaf from B_{j^*} and τ. We repeat this process until τ is just the single vertex z or B_{j^*} becomes fitted. If B_{j^*} is not fitted even when τ is the single vertex z, then delete z from B_{j^*}. If B_{j^*} is still not fitted then delete u from B_{j^*}. Note that at this point $B_{j^*} \subset A_{j^*}$ and hence is fitted. Also, note that $G[B_{j^*}]$ remains connected. Hence $(B_1, \ldots, B_k)$ is an FPP. B_{j^*} does not become a light set because B_j became fitted when the last vertex was deleted from it. Before this vertex was deleted, it was not fitted and hence had weight at least $T_{j^*} + w_{max}$ before this deletion. Since the last vertex deleted has weight at most w_{max}, B_{j^*} has weight at least T_{j^*} and hence is a heavy set. Now we branch into two subcases for defining the cascades.

Case 2.2.1: $z \in B_{j*}$ (i.e, z was not deleted from B_{j^*} in the process above). For each j such that B_j is a heavy set, the cascade of B_j is taken as the cascade of A_j. Since a new ℓ level vertex u is added and all vertices that had level at most ℓ retain their level, we have that $\mathcal{C}_B >_\ell \mathcal{C}_A$. It is also easy to see that $\mathcal{C}_B$ remains ℓ-good.

Case 2.2.2: $z \notin B_{j^*}$ (i.e, z was deleted from B_{j^*}). For each j such that B_j is a heavy set, the cascade of B_j is taken as the cascade of A_j but with the rank ℓ cascade vertex (if it has any) deleted from it. $\mathcal{C}_B \geq_{\ell-1} \mathcal{C}_A$ because all vertices that were at a level of $\ell' = \ell - 1$ or

smaller, retain their levels. Observe that there are no bridges in $\mathcal{C}_B$ to vertices that are at a level at most $\ell-2$, all vertices at a level at most $\ell-2$ still maintain the maximality property, and we did not introduce any cascade vertices. Hence, $\mathcal{C}_B$ is $(\ell-1)$-good. It only remains to prove that there is a bridge $u'v'$ in $\mathcal{C}_B$ such that $level(v') \leq \ell-1$. We know $z \in S_B$. Since z was a rank ℓ cascade vertex in $\mathcal{C}_A$, z had an edge to z' such that z' had level $\ell-1$ in $\mathcal{C}_A$. Observe that level of z' is at most $\ell-1$ in $\mathcal{C}_B$ as well. Hence, taking $u'v' = zz'$ completes the proof.

4 Upper Bounds for Spanning Tree Congestion

We first state the following easy lemma, which together with Proposition 4, implies Lemma 5.

▶ **Lemma 9.** *In a graph $G=(V,E)$, let t_1 be a vertex, and let $t_2,\cdots,t_\ell$ be any $(\ell-1)$ neighbours of t_1. Suppose that there exists a ℓ-connected-partition $\cup_{j=1}^{\ell} V_\ell$ such that for all $j \in \ell$, $t_j \in V_j$, and the sum of degree of vertices in each V_j is at most D. Let τ_j be an arbitrary spanning tree of $G[V_j]$. Let e_j denote the edge $\{t_1,t_j\}$. Let τ be the spanning tree of G defined as $\tau := \left(\cup_{j=1}^{\ell}\ \tau_j\right) \bigcup \left(\cup_{j=2}^{\ell}\ e_j\right)$. Then τ has congestion at most D.*

▶ **Theorem 10.** *For any connected graph $G=(V,E)$, there is an algorithm which computes a spanning tree with congestion at most $8\sqrt{mn}$ in time $\mathcal{O}^*\left(2^{\mathcal{O}\left(n\log n/\sqrt{m/n}\right)}\right)$.*

▶ **Theorem 11.** *For any connected graph $G=(V,E)$, there is a polynomial time algorithm which computes a spanning tree with congestion at most $16\sqrt{mn}\log n$.*

The two algorithms follows the same framework, depicted in Algorithm 1. It is a recursive algorithm; the parameter $\hat{m}$ is a global parameter, which is the number of edges in the input graph G in the first level of the recursion; let $\hat{n}$ denote the number of vertices in this graph.

The only difference between the two algorithms is in Line 15 on how this step is executed, with trade-off between the running time of the step $T(\hat{m}, n_{_H}, m_{_H})$, and the guarantee $D(\hat{m}, n_{_H}, m_{_H})$. For proving Theorem 10, we use Theorem 3(b), Proposition 4 and Lemma 9, yielding $D(\hat{m}, n_{_H}, m_{_H}) \leq 8m_{_H}\sqrt{n_{_H}/\hat{m}}$ and $T(\hat{m}, n_{_H}, m_{_H}) = \mathcal{O}^*\left(2^{\mathcal{O}(n_{_H}\log n_{_H}/\sqrt{\hat{m}/n_{_H}})}\right)$. For proving Theorem 11, we make use of an algorithm in Chen et al. [8], which yields $D(\hat{m}, n_{_H}, m_{_H}) \leq 16m_{_H}\sqrt{n_{_H}/\hat{m}}\log n_{_H}$ and $T(\hat{m}, n_{_H}, m_{_H}) = \mathsf{poly}(n_{_H}, m_{_H})$. Next, we discuss the algorithm in Chen et al., then we prove Theorem 11.

Single-Commodity Confluent Flow. In a *single-commodity confluent flow* problem, the input includes a graph $G=(V,E)$, a *demand* function $w: V \to \mathbb{R}^+$ and ℓ sinks $t_1,\cdots,t_\ell \in V$. For each $v \in V$, a flow of amount $w(v)$ is routed from v to one of the sinks. But there is a restriction: at every vertex $u \in V$, the outgoing flow must leave u on at most 1 edge, i.e., the outgoing flow from u is unsplittable. The problem is to seek a flow satisfying the demands which minimizes the *node congestion*, i.e., the maximum incoming flow among all vertices. Since the incoming flow is maximum at one of the sinks, it is equivalent to minimize the maximum flow received among all sinks. (We assume that no flow entering a sink will leave.) *Single-commodity splittable flow* problem is almost identical to single-commodity confluent flow problem, except that the above restriction is dropped, i.e., now the outgoing flow at u can split along multiple edges. Note that here, the maximum incoming flow might not be at a sink. It is known that single-commodity splittable flow can be solved in polynomial time. For brevity, we drop the phrase "single-commodity" from now on. Corollary 13 below follows from Theorem 12 and Proposition 4.

Algorithm 1: `FindLCST`$(H, \hat{m})$

Input : A connected graph $H = (V_H, E_H)$ on n_H vertices and m_H edges
Output : A spanning tree τ of H

1 **if** $m_H \leq 8\sqrt{\hat{m} n_H}$ **then**
2 | **return** an arbitrary spanning tree of H
3 **end**
4 $k \leftarrow \left\lceil \sqrt{\hat{m}/n_H} \right\rceil$
5 $Y \leftarrow$ a global minimum vertex cut of H
6 **if** $|Y| < k$ **then**
7 | $X \leftarrow$ the smallest connected component in $H[V_H \setminus Y]$
8 | $Z \leftarrow V_H \setminus (X \cup Y)$
9 | $\tau_1 \leftarrow$ `FindLCST`$(H[X], \hat{m})$
10 | $\tau_2 \leftarrow$ `FindLCST`$(H[Y \cup Z], \hat{m})$; ($H[Y \cup Z]$ is connected as Y is a global min cut)
11 | **return** $\tau_1 \cup \tau_2 \cup$ (an arbitrary edge between X and Y)
12 **else**
13 | $t_1 \leftarrow$ an arbitrary vertex in V_H
14 | Pick $\lfloor k/2 \rfloor$ neighbors of t_1 in the graph H; denote them by $t_2, t_3, \cdots, t_{\lfloor k/2 \rfloor + 1}$. Let e_j denote edge $t_1 t_j$ for $2 \leq j \leq \lfloor k/2 \rfloor + 1$.
15 | Compute a $(\lfloor k/2 \rfloor + 1)$-connected-partition of H, denoted by $\cup_{j=1}^{\lfloor k/2 \rfloor + 1} V_j$, such that for each $j \in [\lfloor k/2 \rfloor + 1]$, $t_j \in V_j$, and the total degree (w.r.t. graph H) of vertices in each V_j is at most $D(\hat{m}, n_H, m_H)$. Let the time needed be $T(\hat{m}, n_H, m_H)$.
16 | For each $j \in [\lfloor k/2 \rfloor + 1]$, $\tau_j \leftarrow$ an arbitrary spanning tree of $G[V_j]$
17 | **return** $\left(\cup_{j=1}^{\lfloor k/2 \rfloor + 1} \tau_j\right) \bigcup \left(\cup_{j=2}^{\lfloor k/2 \rfloor + 1} e_j\right)$
18 **end**

▶ **Theorem 12** ([8, Section 4]). *Suppose that given graph G, demand w and ℓ sinks, there is a splittable flow with node congestion q. Then there exists a polynomial time algorithm which computes a confluent flow with node congestion at most $(1 + \ln \ell)q$ for the same input.*

▶ **Corollary 13.** *Let G be a k-connected graph with m edges. Then for any $\ell \leq k$ and for any ℓ vertices $t_1, \cdots, t_\ell \in V$, there exists a polynomial time algorithm which computes an ℓ-connected-partition $\cup_{j=1}^{\ell} V_\ell$ such that for all $j \in \ell$, $t_j \in V_j$, and the total degrees of vertices in each V_j is at most $4(1 + \ln \ell)m/\ell$.*

Congestion Analysis. We view the whole recursion process as a recursion tree. There is no endless loop, since down every path in the recursion tree, the number of vertices in the input graphs are *strictly* decreasing. On the other hand, note that the leaf of the recursion tree is resulted by either (i) when the input graph H to that call satisfies $m_H \leq 8\sqrt{\hat{m} n_H}$, or (ii) when Lines 13–17 are executed. An internal node appears only when the vertex-connectivity of the input graph H is *low*, and it makes two recursion calls.

We prove the following statement by induction from bottom-up: for each graph which is the input to some call in the recursion tree, the returned spanning tree of that call has congestion at most $16\sqrt{\hat{m} n_H} \log n_H$.

We first handle the two basis cases (i) and (ii). In case (i), `FindLCST` returns an arbitrary spanning tree, and the congestion is bounded by $m_H \leq 8\sqrt{\hat{m}n_H}$. In case (ii), by Corollary 13 and Lemma 9, `FindLCST` returns a tree with congestion at most $16m_H\sqrt{n_H/\hat{m}}\log n_H \leq 16\sqrt{\hat{m}n_H}\log n_H$.

Next, let H be the input graph to a call which is represented by an internal node of the recursion tree. Recall the definitions of X, Y, Z, τ_1, τ_2 in the algorithm.

Let $|X| = x$. Note that $1 \leq x \leq n_H/2$. Then by induction hypothesis, the congestion of the returned spanning tree is at most

$$\begin{aligned}&\max\{\text{ congestion of } \tau_1 \text{ in } H[X]\ ,\ \text{congestion of } \tau_2 \text{ in } H[Y \cup Z]\ \} \ +\ |X|\cdot|Y| \\ &\leq 16\sqrt{\hat{m}(n_H - x)}\log(n_H - x) \ +\ \left(\sqrt{\hat{m}/n_H} + 1\right)\cdot x.\end{aligned}$$

Viewing x as a real variable, by taking derivative, it is easy to see that the above expression is maximized at $x = 1$. Thus the congestion is at most $16\sqrt{\hat{m}(n_H - 1)}\log(n_H - 1) + \sqrt{\hat{m}/n_H} + 1 \ \leq\ 16\sqrt{\hat{m}n_H}\log n_H$. In the full version, we do the running time analysis.

5 Lower Bound for Spanning Tree Congestion

Here, we give a lower bound on spanning tree congestion which matches our upper bound.

▶ **Theorem 14.** *For any sufficiently large n, and for any m satisfying $n^2/2 \geq m \geq \max\{16n\log n, 100n\}$, there exists a connected graph with $N = (3 - o(1))n$ vertices and $M \in [m, 7m]$ edges, for which the spanning tree congestion is at least $\Omega(\sqrt{mn})$.*

By some standard random graph arguments, we show that when $n^2/2 \geq m \geq 16n\log n$, there exists a connected graph $H(n, m)$ with n vertices and $[m/2, 2m]$ edges, such that for each subset of vertices S with $|S| \leq n/2$, the number of edges leaving S is $\Omega((m/n)\cdot|S|)$.

We discuss our construction for Theorem 14. The full proof is in the full version. The vertex set V is the union of three vertex subsets V_1, V_2, V_3, such that $|V_1| = |V_2| = |V_3| = n$, $|V_1 \cap V_2| = |V_2 \cap V_3| = \sqrt{m/n}$, and V_1, V_3 are disjoint. In each of V_1, V_2 and V_3, we embed $H(n, m)$. Up to this point, the construction is similar to that of Ostrovskii [21], except that we use $H(n, m)$ instead of a complete graph. The new component in our construction is adding the following edges. For each vertex $v \in V_1 \cap V_2$, add an edge between v and every vertex in $(V_1 \cup V_2) \setminus \{v\}$. Similarly, for each vertex $v \in V_3 \cap V_2$, add an edge between v and every vertex in $(V_3 \cup V_2) \setminus \{v\}$. This new component is crucial: without it, we could only prove a lower bound of $\Omega(m/\sqrt{n}) = \Omega(\sqrt{mn}\cdot\frac{\sqrt{m}}{n})$.

6 Graphs with Expanding Properties

For any vertex subset $U, W \subset V$, let $N_W(U)$ denote the set of vertices in W which are adjacent to a vertex in U. Let $N(U) := N_{V\setminus U}(U)$.

▶ **Definition 15.** A graph $G = (V, E)$ on n vertices is an (n, s, d_1, d_2, d_3, t)-expanding graph if the following four conditions are satisfied:

(1) for each vertex subset S with $|S| = s$, $|N(S)| \geq d_1 n$;
(2) for each vertex subset S with $|S| \leq s$, $|N(S)| \geq d_2|S|$;
(3) for each vertex subset S with $|S| \leq n/2$ and for any subset $S' \subset S$, $|N_{V\setminus S}(S')| \ \geq\ |S'| - t$.
(4) For each vertex subset S, $|E(S, V \setminus S)| \leq d_3|S|$.

▶ **Theorem 16.** *For any connected graph G which is an (n, s, d_1, d_2, d_3, t)-expanding graph, there is a polynomial time algorithm which computes a spanning tree with congestion at most*

$$d_3 \cdot \left[4 \cdot \max \left\{ s+1 \ , \ \left\lceil \frac{3d_1 n}{d_2} \right\rceil \right\} \cdot \left(\frac{1}{2d_1} \right)^{\log_{(2-\delta)} 2} + t \right], \quad \textit{where } \delta = \frac{t}{d_1 n}.$$

In the full version, we show that for random graph $\mathcal{G}(n,p)$ with $p \geq 64 \log n/n$. with high probability the graph is an (n, s, d_1, d_2, d_3, t)-expanding graph with $s = \Theta(1/p)$, $d_1 = \Theta(1)$, $d_2 = \Theta(np)$, $d_3 = \Theta(np)$, $t = \Theta(1/p)$ (and hence $\delta = o(1)$). Applying the above theorem, together with a separate lower bound argument, we show:

▶ **Theorem 17.** *If $G \in \mathcal{G}(n,p)$ where $p \geq 64 \log n/n$, then with probability at least $1 - \mathcal{O}(1/n)$, its STC is $\Theta(n)$.*

References

1. Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. In *FOCS 2008*, pages 781–790, 2008.
2. Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. In *STOC 2012*, pages 395–406, 2012.
3. Noga Alon, Richard M. Karp, David Peleg, and Douglas B. West. A graph-theoretic game and its application to the k-server problem. *SIAM J. Comput.*, 24(1):78–100, 1995.
4. Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph Naor, and Roy Schwartz. Min-max graph partitioning and small set expansion. *SIAM J. Comput.*, 43(2):872–904, 2014.
5. Sandeep N. Bhatt, Fan R. K. Chung, Frank Thomson Leighton, and Arnold L. Rosenberg. Optimal simulations of tree machines (preliminary version). In *FOCS 1986*, pages 274–282, 1986.
6. Hans L. Bodlaender, Fedor V. Fomin, Petr A. Golovach, Yota Otachi, and Erik Jan van Leeuwen. Parameterized complexity of the spanning tree congestion problem. *Algorithmica*, 64(1):85–111, 2012.
7. Hans L. Bodlaender, Kyohei Kozawa, Takayoshi Matsushima, and Yota Otachi. Spanning tree congestion of k-outerplanar graphs. *Discrete Mathematics*, 311(12):1040–1045, 2011.
8. Jiangzhuo Chen, Robert D. Kleinberg, László Lovász, Rajmohan Rajaraman, Ravi Sundaram, and Adrian Vetta. (almost) tight bounds and existence theorems for single-commodity confluent flows. *J. ACM*, 54(4):16, 2007.
9. Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM J. Comput.*, 38(2):608–628, 2008.
10. E. Győri. On division of graphs to connected subgraphs. *Colloq. Math. Soc. Janos Bolyai*, 18:485–494, 1976.
11. Alexander Hoyer and Robin Thomas. The Győri-Lovász theorem. *arXiv*, abs/1605.01474, 2016. URL: `http://arxiv.org/abs/1706.08115`.
12. David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.
13. Marcos A. Kiwi, Daniel A. Spielman, and Shang-Hua Teng. Min-max-boundary domain decomposition. *Theor. Comput. Sci.*, 261(2):253–266, 2001.
14. Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly $O(m \log n)$ time solver for SDD linear systems. In *FOCS 2011*, pages 590–598, 2011.
15. Kyohei Kozawa and Yota Otachi. Spanning tree congestion of rook's graphs. *Discussiones Mathematicae Graph Theory*, 31(4):753–761, 2011.

16 Kyohei Kozawa, Yota Otachi, and Koichi Yamazaki. On spanning tree congestion of graphs. *Discrete Mathematics*, 309(13):4215–4224, 2009.

17 Hiu Fai Law, Siu Lam Leung, and Mikhail I. Ostrovskii. Spanning tree congestions of planar graphs. *Involve*, 7(2):205–226, 2014.

18 László Lovász. A homology theory for spanning trees of a graph. *Acta Math. Acad. Sci. Hungaricae*, 30(3–4):241–251, 1977.

19 Christian Löwenstein, Dieter Rautenbach, and Friedrich Regen. On spanning tree congestion. *Discrete Math.*, 309(13):4653–4655, 2009.

20 Yoshio Okamoto, Yota Otachi, Ryuhei Uehara, and Takeaki Uno. Hardness results and an exact exponential algorithm for the spanning tree congestion problem. *J. Graph Algorithms Appl.*, 15(6):727–751, 2011.

21 M. I. Ostrovskii. Minimal congestion trees. *Discrete Math.*, 285:219–226, 2004.

22 M. I. Ostrovskii. Minimum congestion spanning trees in planar graphs. *Discrete Math.*, 310:1204–1209, 2010.

23 M. I. Ostrovskii. Minimum congestion spanning trees in bipartite and random graphs. *Acta Mathematica Scientia*, 31(2):634–640, 2011.

24 André Raspaud, Ondrej Sýkora, and Imrich Vrto. Congestion and dilation, similarities and differences: A survey. In *SIROCCO 2000*, pages 269–280, 2000.

25 Shai Simonson. A variation on the min cut linear arrangement problem. *Mathematical Systems Theory*, 20(4):235–252, 1987.

26 David Steurer. Tight bounds for the min-max boundary decomposition cost of weighted graphs. In *SPAA 2006*, pages 197–206, 2006.

27 Hitoshi Suzuki, Naomi Takahashi, and Takao Nishizeki. A linear algorithm for bipartition of biconnected graphs. *Inf. Process. Lett.*, 33(5):227–231, 1990.

28 Zoya Svitkina and Éva Tardos. Min-max multiway cut. In *APPROX-RANDOM 2004*, pages 207–218, 2004.

29 Koichi Wada and Kimio Kawaguchi. Efficient algorithms for tripartitioning triconnected graphs and 3-edge-connected graphs. In *Graph-Theoretic Concepts in Computer Science, 19th International Workshop, WG '93, Utrecht, The Netherlands, June 16-18, 1993, Proceedings*, pages 132–143, 1993.

Fully Dynamic Almost-Maximal Matching: Breaking the Polynomial Worst-Case Time Barrier

Moses Charikar[1]
Computer Science Department, Stanford University, Stanford, California, USA

Shay Solomon[2]
IBM Research, T. J. Watson Research Center, Yorktown Heights, New York, USA

Abstract

The state-of-the-art algorithm for maintaining an approximate maximum matching in fully dynamic graphs has a polynomial *worst-case* update time, even for poor approximation guarantees. Bhattacharya, Henzinger and Nanongkai showed how to maintain a constant approximation to the minimum vertex cover, and thus also a constant-factor estimate of the maximum matching size, with polylogarithmic worst-case update time. Later (in SODA'17 Proc.) they improved the approximation to $2+\epsilon$. Nevertheless, the fundamental problem of *maintaining* an approximate matching with sub-polynomial worst-case time bounds remained open.

We present a randomized algorithm for maintaining an *almost-maximal* matching in fully dynamic graphs with polylogarithmic worst-case update time. Such a matching provides $(2+\epsilon)$-approximations for both maximum matching and minimum vertex cover, for any $\epsilon > 0$. The worst-case update time of our algorithm, $O(\mathsf{poly}(\log n, \epsilon^{-1}))$, holds deterministically, while the almost-maximality guarantee holds with high probability. Our result was done independently of the $(2+\epsilon)$-approximation result of Bhattacharya et al., thus settling the aforementioned problem on dynamic matchings and providing essentially the best possible approximation guarantee for dynamic vertex cover (assuming the unique games conjecture).

To prove this result, we exploit a connection between the standard oblivious adversarial model, which can be viewed as inherently "online", and an "offline" model where some (limited) information on the future can be revealed efficiently upon demand. Our randomized algorithm is derived from a deterministic algorithm in this offline model. This approach gives an elegant way to analyze randomized dynamic algorithms, and is of independent interest.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Theory of computation → Graph algorithms analysis, Theory of computation → Dynamic graph algorithms

Keywords and phrases dynamic graph algorithms, maximum matching, worst-case bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.33

Related Version A full version of the paper can be found at [9], `https://arxiv.org/abs/1711.06883`.

Acknowledgements The second author is grateful to Danupon Nanongkai and Uri Zwick for their suggestion to study dynamic matchings in the offline model.

[1] Supported by NSF grant CCF-1617577 and a Simons Investigator Award.
[2] Supported by the IBM Herman Goldstine Postdoctoral Fellowship.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 33; pp. 33:1–33:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Consider a *fully dynamic* setting where we start from an initially empty graph on n fixed vertices G_0, and at each time step i a single edge (u, v) is either inserted in the graph G_{i-1} or deleted from it, resulting in graph G_i. The problem of maintaining a large matching or a small vertex cover in such graphs has attracted a lot of research attention in recent years. In general, one would like to devise an algorithm for maintaining a "good" matching and/or vertex cover with $\mathsf{polylog}(n)$ *update time* (via a data structure that answers queries of whether an edge is matched or not in constant time), where "good" means a good approximation to the maximum matching and/or the minimum vertex cover, and the update time is the time required by the algorithm to update the matching/vertex cover at each step.

One may try to optimize the *amortized* (i.e., average) update time of the algorithm or its *worst-case* (i.e., maximum) update time, over a worst-case sequence of graphs. There is a strong separation between the state-of-the-art amortized bounds and the worst-case bounds. A similar separation exists for various other dynamic graph problems, such as spanning tree, minimum spanning tree and two-edge connectivity. Next, we provide a brief literature survey on dynamic matchings. (See [17, 2, 18, 19] for a detailed survey.)

In FOCS'11, [2] devised an algorithm for maintaining a *maximal matching* with an expected amortized update time of $O(\log n)$ under the oblivious adversarial model.[3] Building on [2], [19] devised a different randomized algorithm with constant amortized update time. Note that a maximal matching provides a 2-approximation for both the maximum matching and the minimum vertex cover, while a better-than-2 approximate vertex cover cannot be efficiently computed under the unique games conjecture (UGC) [14]. In SODA'15 [5] (respectively, STOC'16 [6]) devised a deterministic algorithm for maintaining $(2+\epsilon)$-approximate vertex cover (resp., matching) with amortized update time $O(\log n/\epsilon^2)$ (resp., $O(\mathsf{poly}(\log n, \epsilon^{-1}))$). All these time bounds are amortized.

All the known algorithms for maintaining a better-than-2 approximate matching (for general graphs) require polynomial update time. In FOCS'13 [12] devised a deterministic algorithm for maintaining $(1+\epsilon)$-approximate matching with a worst-case update time $O(\sqrt{m}/\epsilon^2)$, improving over the 3/2-approximation result of [16]. [4] maintained $(3/2+\epsilon)$-approximate matching with an *amortized* update time $O(m^{1/4}/\epsilon^{2.5})$, generalizing their earlier work [3] for bipartite graphs, but the time bound in [3] is worst-case not amortized.

There are two main open questions in this area. The 1st is if one can maintain a *better-than-2* approximate matching in *amortized* polylogarithmic update time. The 2nd is:

▶ **Question 1.** *Can one maintain a "good" (close to 2) approximate matching and/or vertex cover with* worst-case *polylogarithmic update time?*

In a recent breakthrough, Bhattacharya, Henzinger and Nanongkai devised a deterministic algorithm that maintains a *constant* approximation to the minimum vertex cover, and thus also a constant-factor estimate of the maximum matching size, with polylogarithmic worst-case update time. While this result makes significant progress towards Question 1, this fundamental question remained open.[4] In particular, no algorithm for maintaining a matching with sub-polynomial worst-case update time was known, even if a polylogarithmic approximation guarantee on the matching size is allowed!

[3] In the standard *oblivious adversarial model* (cf. [8], [13]), the adversary knows all the edges in the graph and their arrival order, but is not aware of the random bits used by the algorithm.

[4] Later (in SODA'17 Proc. [7]) Bhattacharya et al. significantly improved the approximation to $2+\epsilon$. However, our result was done independently to [7]. Moreover, [7] solves Question 1 only for vertex cover. Independently of us, Arar et al. [1] solves Question 1 for matching by building on [7].

In this paper we devise a *randomized* algorithm that maintains an *almost-maximal matching (AMM)* with a polylogarithmic update time. We say that a matching for G is *almost-maximal* w.r.t. some slack parameter ϵ, or $(1-\epsilon)$-*maximal* in short, if it is maximal w.r.t. any graph obtained from G after removing $\epsilon \cdot |\mathcal{M}^*|$ arbitrary vertices, where $\mathcal{M}^*$ is a maximum matching for G. Just as a maximal matching provides a 2-approximation for matching and vertex cover, an AMM provides a $(2+\epsilon)$-approximation. We show:

▶ **Theorem 2.** *For any $\epsilon > 0$, one can maintain an AMM with worst-case update time $O(\mathsf{poly}(\log n, \epsilon^{-1}))$, where the $(1-\epsilon)$-maximality guarantee holds with high probability.*

Our update time is $O(\max\{\log^7 n/\epsilon, \log^5 n/\epsilon^4\})$; reducing this bound towards constant lies outside the scope of this paper; see Sec. 9 in the full version [9] (shortly, "i.t.f.v.").

The algorithm's worst-case guarantee can be strengthened, using [20], to bound the number of changes (replacements) to the matching. Optimizing this measure is important in various practical applications; refer to [20] for a motivation of this measure.

Our result resolves Question 1 in the affirmative, up to the ϵ dependency. In particular, it is essentially the best result possible under the UGC for the dynamic vertex cover problem; it started to circulate in Nov. 2016, and is independent of the $(2+\epsilon)$-vertex cover result of [7].

On the way to this result, we devise a *deterministic* algorithm that maintains an AMM with a polylogarithmic update time in a natural *offline model* that is described next. This deterministic algorithm may be of independent interest, as the offline setting seems important in its own right; see p. 3 i.t.f.v. for further details.

1.1 A Technical Overview

The offline model. Suppose the entire update sequence is known in advance, and is stored in some data structure. Suppose further that for any i, accessing the ith edge update via the data structure is efficient, taking $\mathsf{polylog}(n)$ or even $O(1)$ time. A natural question is whether one can exploit this knowledge of the future to obtain better algorithms for maintaining a good matching and/or vertex cover. Consider in particular the *maximal matching* problem, and a deletion of a matched edge (u, v) from the graph (which is the problematic part). If u has a free neighbor, we need to match them, and similarly for v. The algorithm may naively scan the neighbors of u and v, which may require $O(n)$ time. Surprisingly, this naive $O(n)$ bound is the state-of-the-art for general (dense) graphs, unless one allows both randomization and amortization [2, 19]. Can one do better in the offline setting?

We argue that a dynamic maximal matching can be maintained in the offline setting *deterministically* with constant *amortized* update time. To this end, we make the following observation: The machinery of [2, 19] extends *seamlessly* to the offline setting. More specifically, in contrast to the algorithms [2, 19], which choose the matched edge of v *uniformly at random* among a subset of its adjacent edges E_v that is computed carefully by those algorithms (details below), in the offline setting we choose the matched edge to be the one that will be deleted last among E_v. (We do not choose the edge that will be deleted last among all adjacent edges of v, as this is doomed; see the technical overview i.t.f.v). It is readily verified that the analysis of [2, 19] carries over to the offline setting directly.

The resulting deterministic algorithm for the offline setting is inherently *amortized*, whereas our focus is on *worst-case* bounds. To obtain good worst-case bounds, we build on the machinery of [2, 19]. The price of translating the amortized bounds of [2, 19] into a similar worst-case bound is that the maintained matching is no longer maximal, but rather

almost-maximal.[5] This translation is highly non-trivial, and is carried out in two stages. First, we consider the offline setting, and devise a deterministic algorithm there. Coping with the offline setting is easier than with the standard setting, as it allows us to ignore intricate probabilistic considerations, and to handle them separately. Second, we convert the results for the offline setting to the standard setting. The algorithm itself remains essentially the same. (Instead of choosing the edge that will be deleted last, choose a random edge.) On the other hand, showing that the maintained matching remains almost-maximal requires more work. This two-stage approach thus provides an elegant way to analyze randomized dynamic algorithms, and we believe it would be useful in other dynamic graph problems as well.

The framework of [2, 19]. We next provide a rough description of the *amortized* framework of [2, 19]. (The approach of [19] builds on the framework of [2] and extends it; for clarity, we won't distinguish between [2] and [19].)

Matched edges will be chosen randomly. If an edge $e = (u, v)$ is chosen to the matching uniformly at random among k adjacent edges of either u or v, w.l.o.g. u, we say that its *potential* is k. Under the oblivious adversarial model, the expected number of edges incident on u that are deleted from the graph before deleting edge (u, v) is $k/2$. Thus, following a deletion of a matched edge (u, v) with potential k from the graph, we have time $\tilde{O}(k)$ to handle u and v in the amortized sense.

Each vertex v dynamically maintains a *level* ℓ_v; informally, v's level will be logarithmic in the potential value of the matched edge adjacent to v. Free vertices are at level -1; matched vertices are at levels between 0 and $O(\log n)$. Based on vertices' levels, a dynamic edge orientation is maintained, where each edge is oriented towards the lower level endpoint.

When a vertex u becomes free, the algorithm (usually) chooses a mate for it randomly. If this mate w is already matched, say to w', the algorithm has to delete edge (w, w') from the matching in order to match u with w. However, we will be able to compensate for the loss in potential value (caused by deleting edge (w, w') from the matching) if it is significantly smaller than the potential of the newly created matched edge. Since vertices' levels are logarithmic in their potential, all neighbors of u with lower level should have potential value at most half the potential value of the new matched edge on u. In other words, for each of those neighbors, we can afford to break their old matched edge. Hence, the mate w of u will be chosen uniformly at random among u's neighbors with lower level.

A central obstacle is to distinguish between neighbors of u with level ℓ_u and those with lower level. Indeed, it is possible that most of u's neighbors have level ℓ_u, and none of them can be chosen as a mate for u. Roughly speaking, the execution of the algorithm splits into two cases. If the current out-degree of u is not (much) larger than its out-degree at the time its old matched edge got created, then we should be able to afford to scan all of them, due to sufficiently many adversarial edge deletions that are expected to occur. Notice that in this case the charging argument is based on *past* edge deletions.

The second case is when the out-degree of u is (much) larger than what it was when the old matched edge got created. The time needed for distinguishing u's neighbors at level ℓ_u from those at lower levels could be significantly larger than the "money" we got from

[5] The amortized update time analysis of the algorithm from [2] (both the FOCS'11 and subsequent journal SICOMP'15 versions) was erroneous, but was corrected in a subsequent erratum by the same authors. (The amortized update time analysis of the algorithm from [19] is different than the one used in [2], and does not have that mistake.) Although our algorithm builds on the machinery of [2, 19], the mistake in [2] does not affect the current paper, as we provide an independent analysis for a different algorithm, which bounds the worst-case update time of our algorithm rather than the amortized update time.

past edge deletions. In this case the algorithm *raises* u to a possibly much higher level ℓ^*, where there are not too many neighbors for u at that level as compared to the number of neighbors at lower levels. Having raised u to that level, we can perform the random sampling of its mate among all neighbors of level lower than ℓ^*. Notice that in this case the charging argument is not based on the past, but rather on *future* edge deletions.

Our approach. Note that the framework of [2, 19] is inherently *amortized*: Every once in a while there are "expensive" operations, which are charged to "cheap" operations that occurred in the past or will occur in the future. To obtain a low worst-case update time, we should be *cheap in any time interval*, thus we can rely neither on the past nor the future. Consider a matched edge (u, v) deleted by the adversary. We expect the adversary to make many edge deletions on at least one of these endpoints before deleting this edge. Alas, it is possible that all these edge deletions occurred a long time ago, which is useless for a worst-case algorithm. Consider the offline setting, and let $e_1, \ldots, e_\eta$ be η arbitrary matched edges with potential value k. For each such edge e_i, let $S(e_i)$ be its *sample*, i.e., the set of edges from which e_i was sampled to the matching. In the offline setting e_i will be deleted only after all $k-1$ other edges from its sample have been deleted. However, it is possible that the adversary first deletes the first $k-1$ edges from the samples of each of the matched edges, and only then turn to deleting the matched edges. Assuming k is large, it takes a long time for the adversary to delete the first $\eta(k-1)$ edges from all η samples, but the amortized algorithms of [2, 19] remain *idle* during all this time. An algorithm with a low worst-case update time must be active in this time interval, as immediately afterwards the adversary can remove the η matched edges from the graph, much faster than the algorithm can add edges to the matching in their place, leading to a poor approximation guarantee. Hence, at any point in time, the algorithm needs to be proactive and protect itself from such a situation happening in the future.

Generally, while in an amortized algorithm invariants may sometimes be violated and then restored via expensive operations, an algorithm with a low worst-case update time should persistently "clean" the graph, making sure that it is never close to violating any invariant. Naturally, we will need to maintain additional invariants to those maintained by the amortized algorithms [2, 19]. To this end we employ four different data structures that we call *schedulers*, each for a different purpose. Each of those schedulers consists of $O(\log n)$ sub-schedulers, a single sub-scheduler per level. Next we fix some level $\ell \approx \log k$, where k is the potential of the matched edges on that level, and focus on it.

The scheduler `unmatch-schedule` periodically removes edges from the matching, one after another, by always picking a matched edge whose *remaining sample* (i.e., the set of edges from the sample that have not been deleted yet from the graph) is smallest. As strange as it might seem, this strategy enables us to guarantee that only few matched edges will ever be deleted by the adversary. Note that removing a matched edge from the matching is not a cheap operation, as we need to find new mates for the two endpoints of the edge. Thus, the execution of the scheduler must be *simulated* over sufficiently many adversarial updates, which may include more deletions. But, as we control the rate at which the scheduler is working, we can make sure that it works sufficiently faster than the adversary. Therefore, in this "game" between the scheduler and the adversary, the scheduler will always win.

The role of `unmatch-schedule` is to make sure that all the samples are pretty full. Intuitively, this provides the counter-measure of relying on past adversarial edge deletions, as done in the amortized argument. The next scheduler `rise-schedule` provides the counter-measure of relying on future adversarial edge deletions. Recall that future edge deletions

are used in the amortized argument only in the case that a vertex had to rise to a higher level, which occurs only if its out-degree became too large for its current level. The role of `rise-schedule` is to make sure that vertices' out-degrees are always commensurate with their level. This scheduler periodically raises vertices to the level ℓ of which it is in charge, one after another, by always choosing to raise a vertex with the largest number of neighbors at level lower than ℓ. Although the two schedulers are based on the same principle, the game that we play here is not between the scheduler and the adversary, because here the algorithm itself may change the level of vertices and their out-degree, so `rise-schedule` has to compete against both the adversary and the algorithm. In contrast to the other scheduler, speeding up the rate at which `rise-schedule` works will not help winning the game. Instead, we manage to bound the speed of this scheduler with respect to that of the (adversary + algorithm), which enables us to show that the out-degree of all vertices is always in check.

For the offline model, these two schedulers suffice. However, in the oblivious adversarial model, the adversary will manage to delete some matched edges from time to time. The scheduler `free-schedule` periodically handles all the vertices that become free due to the adversary, one after another. Using the property that all samples are always pretty full, we manage to prove that only an ϵ-fraction of the matched edges get destroyed by the adversary at any time interval. Note that this bound is probabilistic – to make sure that it indeed occurs with high probability, we also use another scheduler `shuffle-schedule`, which periodically removes a random edge from the matching. For technical reasons, it is vital that `shuffle-schedule` would work sufficiently faster than some of the other schedulers.

2 The Update Algorithm

2.1 Invariants and schedulers. Our algorithm builds on the amortized algorithms by [2, 19], which maintain for each vertex v a *level* ℓ_v, with $-1 \leq \ell_v \leq \log_\gamma(n-1)$, where $\gamma = \Theta(\log n)$. (We use logarithms in base $\gamma = \Theta(\log n)$, whereas [2] and [19] use base 2 and 5, respectively.) Based on the levels of vertices, an edge orientation is maintained, with the vertex out-degree serving as an important parameter. The amortized algorithms of [2, 19] maintain the following invariants (Invariants 3(a)-3(d)) *at all times*. i.e., these invariants hold at the *end of the execution* of the update algorithms (and before the next update operation occurs). These invariants may become violated *throughout the execution* of the update algorithms. Also, the runtime of the update algorithms of [2, 19] may be $\Omega(n)$ in the worst case, thus it may take them a lot of time to restore the validity of these invariants, once violated. We added a comment to the right of each of these invariants, either /* maintained */ or /* partially maintained */, to indicate jf the respective invariant is maintained fully or only partially by our new algorithm. Our algorithm will maintain Invariants 3(a) and 3(b) at all times, as in the amortized algorithms [2, 19], where Invariants 3(c) and 3(d) are maintained only partially. Next, we make this statement precise.

▶ **Invariant 3.**

(a) *Any matched vertex has level at least 0. /* maintained */*

(b) *The endpoints of any matched edge are of the same level, and this level remains unchanged until the edge is deleted from the matching. (We henceforth define the level of a matched edge, which is at least 0 by item (a), as the level of its endpoints.) /* maintained */*

(c) *Any free vertex has level -1 and out-degree 0. (The matching is maximal.) /* partially maintained */*

(d) *An edge (u, v) with $\ell_u > \ell_v$ is* oriented *by the algorithm as $u \to v$. (If $\ell_u = \ell_v$, the orientation of (u, v) is determined suitably by the algorithm.) /* partially maintained */*

Once a matched vertex becomes free, its level will exceed -1 until the update algorithm handles it. We say that such a vertex is *temporarily free* (shortly, *TF*), meaning that it is not matched to any vertex yet, but its level and out-degree remain temporarily as before. From now on, we distinguish between *free* and *TF* vertices: Free vertices are unmatched and their level is -1, while TF vertices are unmatched and their level exceeds -1. By making this distinction, Invariant 3(c) holds true as stated. Combining it with Invariant 3(a), we obtain:

▶ **Invariant 4.** *Any vertex of level -1 is unmatched and has out-degree 0.*

Invariants 3(c) and 4 do not apply to TF vertices; thus there may be edges between TF (thus unmatched) vertices, hence the matching is not maximal. The challenge is to guarantee that the number of TF vertices is small w.r.t. the number of matched vertices, yielding an AMM.

TF vertices are handled via data structures that we call *schedulers*. We distinguish between vertices that become TF due to the adversary and those due to the update algorithm itself. For each level ℓ, we maintain a queue Q_ℓ of level-ℓ vertices that become TF due to the adversary, and the vertices in Q_ℓ will be handled, one after another, via appropriate schedulers. We will need to make sure that the total number of vertices over the queues of all levels is in check at all times. The various schedulers need to work together, without conflicting each other; the exact way in which they work is described soon.

A TF vertex that is being handled by some scheduler is called *active*, and the process of handling it may be simulated over multiple update operations. Hence, there might be *inconsistencies* in the data structures throughout this process concerning the active vertices. To account for those inconsistencies, we hold a list of active vertices, denoted *Active*, and we will make sure that this list is of size $O(\log_\gamma n) = O(\log n)$ at any point in time. By bounding the number of active vertices, we can *authenticate* the up-to-date information concerning active vertices efficiently; this *authentication process* is described i.t.f.v in Section 2.3. Our algorithm maintains the following relaxation of Invariant 3(d).

▶ **Invariant 5.** *Any edge (u, v), with $\ell_u > \ell_v$ and $u, v \notin$ Active, is* oriented *as $u \to v$.*

For each vertex v, we maintain its neighbors and outgoing neighbors in linked lists $\mathcal{N}_v$ and $\mathcal{O}_v$, and its incoming neighbors via a more detailed structure $\mathcal{I}_v$; see p. 8 i.t.f.v.

Our algorithm employs four different schedulers, each of which consists of $O(\log_\gamma n) = O(\log n)$ sub-schedulers, a single sub-scheduler per level $\ell = 0, 1, \ldots, \log_\gamma(n-1)$. It is instructive to think of each sub-scheduler as running threads of execution, and of its scheduler as synchronizing $O(\log n)$ threads, one per level. Each thread executed by a level-ℓ sub-scheduler, hereafter *level-ℓ thread*, will run in the same amount of time $T_\ell = \gamma^\ell \cdot \Theta(\log^4 n)$, by "sleeping" if finishing the execution prematurely. To achieve a low worst-case update time, the execution of any such thread is not carried out at once, but is rather carried out (or *simulated*) over multiple update operations, simulating a fixed number of computation steps per update operation; we refer to that number as a *simulation parameter*, and we'll use two of them, $\Delta := \Theta(\log^5 n/\epsilon)$ and $\Delta' = \Delta \cdot \gamma = \Delta \cdot \Theta(\log n)$. The schedulers `free-schedule`, `rise-schedule` and `shuffle-schedule` use a simulation parameter of Δ', whereas `unmatch-schedule` uses a simulation parameter of Δ, and is thus "slower" than the others by a factor of $\gamma = \Theta(\log n)$. The simulation parameters, Δ or Δ', determine the number of update operations required to finish the execution of the thread, T_ℓ/Δ or T_ℓ/Δ', respectively. We refer to this number as the *(level ℓ) simulation time*; unlike the simulation parameters, which do not change with the level, the simulation times grow with each level by a factor of γ.

1st scheduler. The first scheduler `free-schedule` handles all vertices that become TF due to the adversary; for each level $\ell = 0, 1, \ldots, \log_\gamma(n-1)$, the corresponding sub-scheduler `free-schedule`$_\ell$ handles all vertices of Q_ℓ, one after another. The exact procedure for handling a TF vertex v, `handle-free`(v), is described in Section 2.4. Procedure `handle-free`(v) will be executed by a single level-ℓ thread corresponding to v that runs in an overall time of T_ℓ, simulating Δ' steps of this procedure following each update operation. The $\log_\gamma(n-1)+1$ execution threads (over all levels) executed by `free-schedule` following every update operation are handled *sequentially*, by decreasing order of simulation times, and thus by decreasing order of levels, i.e., the $\log_\gamma(n-1)$-level thread is handled first, then the $\log_\gamma(n-1)-1$-level thread, etc., until the 0-level thread. Note that these threads execute different calls of Procedure `handle-free`, which handle vertices at different levels. [[S: moved this sentence]] Following each update operation, the $\log_\gamma(n-1)$-level thread simulates Δ' steps of its own call of Procedure `handle-free`, the $\log_\gamma(n-1)-1$-level thread simulates Δ' steps of its own call, and so on, hence the total time spent by `free-schedule` following a single update operation is $\Delta' \cdot (\log_\gamma(n-1)+1) = O(\log^7 n/\epsilon)$.

By the same principle, the total time spent by `rise-schedule` and `shuffle-schedule` following a single update operation will be bounded by $\Delta' \cdot (\log_\gamma(n-1)+1) = O(\log^7 n/\epsilon)$. On the other hand, `unmatch-schedule` has a simulation parameter of Δ rather than Δ', so the total time spent by this scheduler following a single update will be bounded by $\Delta \cdot (\log_\gamma(n-1)+1) = O(\log^6 n/\epsilon)$. This scheme gives rise to a worst-case update time of $O(\log^7 n/\epsilon)$, and this bound holds *deterministically*.

2nd scheduler. The second scheduler `unmatch-schedule` removes matched edges from the matching; for each ℓ, the corresponding sub-scheduler `unmatch-schedule`$_\ell$ removes level-ℓ edges from the matching, one after another, as follows. Each level-ℓ matched edge $e = (u, v)$ is sampled uniformly at random from between $(1-\epsilon)\cdot\gamma^\ell$ and γ^ℓ edges. (In the offline setting, we choose the edge that will be deleted last among those.) We refer to this edge set, denoted by $S(e)$, as the *sample space* (or *sample*) of e. As time progresses, some edges of $S(e)$ may be deleted from the graph; denote by $S^*(e)$ the *original* sample of e, with $(1-\epsilon)\cdot\gamma^\ell \le |S^*(e)| \le \gamma^\ell$, and by $S_t(e) = S(e)$ its sample *remaining* at time t. The goal of `unmatch-schedule`$_\ell$ is to guarantee that the samples of all level-ℓ matched edges never reach $(1-2\epsilon)\cdot\gamma^\ell$; more accurately, `unmatch-schedule`$_\ell$ maintains the following invariant:

▶ **Invariant 6.** *For any level-ℓ matched edges e with $T_\ell/\Delta \ge 1$ and any t, $|S_t(e)| > (1-2\epsilon)\cdot\gamma^\ell$.*

To maintain this invariant, `unmatch-schedule`$_\ell$ will always remove a matched edge of smallest remaining sample (can be easily carried out in $O(1)$ time). For each level-ℓ matched edge $e = (u, v)$ that is removed by `unmatch-schedule`$_\ell$, its two endpoints u and v become TF, and they are handled by appropriate calls to Procedure `handle-free`. More specifically, we execute Procedure `handle-free`(u) and then `handle-free`(v) by running a level-ℓ thread, which runs in an overall time of T_ℓ, simulating Δ steps of execution following each update operation. The intuition as to why `unmatch-schedule`$_\ell$ can maintain Invariant 6 is the following. (See Section 4.2 i.t.f.v. for the formal argument.) Since $T_\ell = \gamma^\ell \cdot \Theta(\log^4 n)$ and $\Delta = \Theta(\log^5 n/\epsilon)$, the simulation time T_ℓ/Δ of a thread run by `unmatch-schedule`$_\ell$ (which designates the number of update operations needed for simulating its entire execution) is $\Theta(\epsilon(\gamma^\ell/\log n))$. In other words, `unmatch-schedule`$_\ell$ can remove a level-ℓ matched edge within $T_\ell/\Delta = \Theta(\epsilon(\gamma^\ell/\log n))$ adversarial update operations. On the other hand, the expected number of adversarial edge deletions needed to turn a "full" level-ℓ matched edge e (with sample $|S^*(e)| \ge (1-\epsilon)\cdot\gamma^\ell$) into an "under full" edge (with sample $\le (1-2\epsilon)\cdot\gamma^\ell$) is

$\Omega(\epsilon \cdot \gamma^\ell)$. Thus `unmatch-schedule`$_\ell$ is "faster" than the adversary by at least a logarithmic factor, assuming $T_\ell/\Delta \geq 1$ (which holds when $\gamma^\ell = \Omega(\log n/\epsilon)$), a property that suffices for showing that no edge is ever under full, i.e., the samples of all level-ℓ matched edges are always in check. This is the idea behind maintaining the validity of Invariant 6 in levels ℓ for which the simulation time satisfies $T_\ell/\Delta \geq 1$. This invariant, in turn, guarantees that the adversary is unlikely to delete any particular edge from the matching, using which we show (in Section 6 i.t.f.v.) that the maintained matching is always almost-maximal with high probability. The complementary regime of levels, namely, levels ℓ for which $T_\ell/\Delta < 1$, is trivial and does not rely on Invariant 6, as then the adversary does not make any edge deletion within the time required by a level-ℓ thread to complete its entire execution.

3rd scheduler. Let $N_v(\ell)$ denote the set of neighbors of v with level strictly lower than ℓ, and write $\phi_v(\ell) = |N_v(\ell)|$. For each vertex v, we will maintain the $\phi_v(\ell)$ values for all levels ℓ greater than the current level ℓ_v of v. For any level $\ell \leq \ell_v$, the corresponding value $\phi_v(\ell)$ will not be maintained, and the algorithm will have to compute it "on the fly", if needed. The algorithm of [2] maintains the invariant that $\phi_v(\ell) < \gamma^\ell$, for any v and $\ell > \ell_v$. (Recall that γ is taken to be constant in [2], whereas here we take γ to be $\Theta(\log n)$.) The scheduler `rise-schedule` maintains the following relaxation of the invariant from [2], and it does so by raising vertices to higher levels in a specific order, as described next.

▶ **Invariant 7.** *For any vertex v and any level $\ell > \ell_v$, $\phi_v(\ell) \leq \gamma^\ell \cdot O(\log^2 n)$.*

For each level ℓ, the corresponding sub-scheduler `rise-schedule`$_\ell$ is responsible for maintaining the invariant w.r.t. that level. Whenever a new level-ℓ thread is initiated by `rise-schedule`$_\ell$, it starts by *authenticating* the $\phi_v(\ell)$ values over all vertices v using the *Active* list. (The authentication process takes time $O(\log^2 n)$ to guarantee that all $\phi_v(\ell)$ values are up to date, and is described in Section 2.3 i.t.f.v.) Then the thread picks a vertex v whose $\phi_v(\ell)$ value is highest among all vertices with level lower than ℓ (can be easily carried out in $O(1)$ time). These steps take time $O(\log^2 n)$, and are thus carried out by the thread "instantly", i.e., without simulating their execution over subsequent update operations. The same execution thread continues to removing v's old matched edge (v, w) (if exists) from the matching, and raises v to level ℓ by executing Procedure `set-level`(v, ℓ), whose description is in Section 2.2. The runtime of this procedure is high, so its execution is simulated over multiple update operations, simulating Δ' execution steps following each update operation. Then the same execution thread handles the two TF vertices v and w using Procedure `handle-free`, i.e., it continues to executing the call to `handle-free`(v) and then to `handle-free`(w), simulating Δ' execution steps following each update operation.

4th scheduler. The fourth scheduler `shuffle-schedule` removes matched edges from the matching uniformly at random. By working faster than some other schedulers (`unmatch-schedule` in particular), it forms a dominant part of the algorithm, using which we show (Section 6.2 i.t.f.v.) that it provides a near-uniform random shuffling of the matched edges. This random shuffling facilitates the proof of the assertion that the adversary is unlikely to delete any particular edge from the matching. For each ℓ, the sub-scheduler `shuffle-schedule`$_\ell$ always picks a matched edge uniformly at random among all remaining level-ℓ edges, and then removes it from the matching. As with `unmatch-schedule`$_\ell$, for each level-ℓ matched edge $e = (u, v)$ that is removed by `shuffle-schedule`$_\ell$, its two endpoints u and v become TF, and they are handled by calls to Procedure `handle-free`. We execute these calls (to `handle-free`(u) and `handle-free`(v)) by running a level-ℓ thread, which runs

in an overall time of T_ℓ, and we simulate Δ' (rather than Δ as with `unmatch-schedule`$_\ell$) execution steps following each update operation, which ensures that `shuffle-schedule` is faster than `unmatch-schedule` by a logarithmic factor. We only need to apply the shuffling in levels ℓ for which the simulation time satisfies $T_\ell/\Delta \geq 1$, as in the complementary regime ($T_\ell/\Delta < 1$) the adversary does not make any edge deletion within the time required by a level-ℓ thread to complete its entire execution, and then a random shuffling is redundant.

2.2 Procedure `set-level`(v, ℓ). (This procedure is described in detail in Section 3.1 i.t.f.v.) Whenever the update algorithm examines a vertex v, it may need to re-evaluate its level. After the new level ℓ is determined (outside this procedure, details below), the algorithm calls Procedure `set-level`(v, ℓ). Although setting the level of v to ℓ can be done instantly, the task of Procedure `set-level`(v, ℓ) is to update the relevant data structures as a result of this level change. This process involves updating the sets of outgoing and incoming neighbors of v and some of its neighbors (or "flipping" the respective edges) so as to maintain Invariant 5, and also updating the ϕ values of v and its relevant neighbors. We refer to this (possibly long) process as the *falling* (if $\ell < \ell_v$) or *rising* of v (if $\ell > \ell_v$); the thread executing this procedure simulates multiple execution steps following each update operation. We will need to make sure that any call to `set-level`(v, ℓ) is executed by a level-$\hat{\ell}$ thread, where $\hat{\ell} \geq \tilde{\ell} := \max\{\ell_v, \ell\}$. We show (see Lemma 3.2 i.t.f.v.) that the runtime of this procedure is bounded by $O((\phi_v(\tilde{\ell}+1) + \log n) \cdot \log n)$, where $\phi_v(\tilde{\ell}+1)$ is the number of v's neighbors of level $< \tilde{\ell}+1$ at the beginning of this procedure's execution.

2.3 Procedures `handle-insertion`(u, v) and `handle-deletion`(u, v). (These procedures are described in detail in Section 3.2 i.t.f.v.) An edge insertion (u, v) is handled (via `handle-insertion`(u, v)) in the obvious way in time $O(\log^4 n)$, which is within the time reserved for a single update operation. An edge deletion (u, v) is handled (via `handle-deletion`(u, v)) similarly, unless (u, v) is matched, in which case both u and v become TF, and they are inserted to the queue Q_{ℓ_u} (by Invariant 3(b) $\ell_u = \ell_v$). As described above, `free-schedule`$_{\ell_u}$ will handle u and v (by making the calls to `handle-free`(u) and `handle-free`(v)), one after another, after handling all preceding vertices in Q_{ℓ_u}.

2.4 Procedure `handle-free`(v). (This procedure is described in detail in Section 3.3 i.t.f.v.) This procedure handles a TF vertex v, and is first invoked by the schedulers as described above, but then also recursively. It starts by computing the highest level $\ell, 0 \leq \ell \leq \ell_v$, where $\phi_v(\ell) \geq \gamma^\ell$, and the corresponding vertex set $N_\ell(v)$ of v, in order to sample a neighbor w of level $< \ell$ as v's new mate. The sampling is done from the set $N'_\ell(v)$ of non-active vertices in $N_\ell(v)$. To match v with w, we first delete the old matched edge (w, w') on w (if exists), thus rendering w' TF. Second, we let v and w fall and rise to the same level ℓ, respectively, by calling to `set-level`(v, ℓ) and `set-level`(w, ℓ). We then match v with w, thus creating a new level-ℓ matched edge (satisfying Invariant 3(b)). Finally, assuming w was previously matched to w', we handle w' recursively by calling to `handle-free`(w'). (In the degenerate case that no level ℓ as above exists, we have $\phi_v(0) = 0$, i.e., v does not have any neighbor at level -1, thus we call `set-level`$(v, -1)$ and v becomes free.)

Our update algorithm guarantees that this procedure is executed by a level-ℓ_v thread, where ℓ_v is v's level at the outset of the procedure's execution. The same thread is used also for all subsequent recursive calls. We show (Lemma 3.1 i.t.f.v.) that the runtime of Procedure `handle-free`(v) is bounded by $O(\gamma^{\ell_v} \cdot \log^4 n)$.

3 Analysis

3.1 Schedulers. The principle that governs the operation of `unmatch-schedule` and `rise-schedule` can be described via a balls and bins game by [11, 15, 10] between two players. Initially there are N empty bins. In each round Player I removes a bin of largest size, then Player II may add up to $b \geq 1$ balls to bins. The game ends when no bin is left or when the size of any bin reaches some parameter k. Player I wins (respectively, loses) in the former (resp., latter) case. As follows from [11, 15, 10], Player I wins if $b < \frac{k}{(\ln N+1)}$.

To prove that `unmatch-schedule` and `rise-schedule` maintain Invariants 4 and 5, respectively, we carefully build on this principle in several steps; see Sections 4.2 and 4.3 i.t.f.v. The analysis of `rise-schedule` is more intricate than that of `unmatch-schedule`, as our update algorithm affects both players in the underlying balls and bins game; we henceforth focus on `rise-schedule`, highlighting some insights behind our analysis. Consider the variant of the game where the bins are not empty initially, but rather contain at most $k' \ll k$ balls each. Using the same argument, Player 1 wins if $b < \frac{k-k'}{(\ln N+1)}$. We show that Invariant 7 is maintained by translating this variant of the game appropriately.

Fix any level $\ell \geq 0$. Invariant 7 requires that the $\phi_v(\ell)$ values are always $< \gamma^\ell \cdot O(\log^2 n)$, for all v with $\ell_v < \ell$. In the balls and bins game, the bins represent the respective vertex sets $N_v(\ell)$ (of v's neighbors of level $\leq \ell - 1$), for $\ell_v < \ell$. (Our algorithm does not maintain these sets, only the ϕ values.) The sub-scheduler $\texttt{rise-schedule}_\ell$ is Player I in the game; it always picks a vertex v whose $\phi_v(\ell)$ value is highest, and raises it to level ℓ. Following this rise, $\ell_v = \ell$, hence the invariant for v and level ℓ holds vacuously. Thus, the analog of removing a bin by Player I is to raise a vertex to level ℓ.

At the beginning the graph is empty, so all vertex levels are -1 and all vertex sets $N_v(\ell)$ are empty. Thus initially we have an empty bin for every vertex. As time progresses some of these bins are being removed due to vertex rising. When a vertex rises to level ℓ, all its bins up to level ℓ are removed instantly. Bins are also created due to vertices falling, by Procedure `handle-free`. When a vertex v starts falling from level ℓ_v to level ℓ, it is as if the corresponding vertex sets $N_v(j)$ in all levels $j \in \{\ell + 1, \ldots, \ell_v\}$ are created instantly; the level of v is viewed as its destination level ℓ from the moment its falling to level ℓ starts. Although the same vertex set $N_v(j)$ may be removed and created multiple times, we view any such newly created set as a different bin that was there from the game's outset. To comply with the initial bound of $\leq k'$ balls in any bins, we set $k' = k'_\ell$ as γ^ℓ, and prove (Lemma 4.1 i.t.f.v.) that any newly created level-ℓ bin contains $\leq k' = \gamma^\ell$ balls.

The level-ℓ vertex sets $N_v(\ell)$ and values $\phi_v(\ell)$ may grow either due to edge insertions (by adversary) or due to falling vertices (by update algorithm). In other words, Player II in the game is (adversary + update algorithm). Letting Player I ($\texttt{rise-schedule}_\ell$) work faster than the other sub-schedulers is problematic: While this would lead to more vertices rising, which helps Player I win, each vertex rising may trigger the fall of another vertex, which has the opposite effect. Instead, we prove (Lemma 4.2 i.t.f.v.) that the number of balls $b = b_\ell$ added to the bins by Player II while Player I removes a bin is $O(\gamma^\ell \cdot \log n)$. It is easy to verify that the number N of level-ℓ bins is polynomially bounded, so $\ln N = \Theta(\log n)$. Taking $k = O(\gamma^\ell \cdot \log^2 n)$ completes the translation of the balls and bins game. By setting the constant appropriately, we obtain $b < \frac{k-k'}{\ln N}$, hence Player I wins the game. Consequently, we showed that Invariant 7 is maintained.

3.2 Proof of (Almost-)Maximality. To prove almost-maximality, we show that the number of TF vertices is always an ϵ-fraction of the number of matched edges. We only consider here TF vertices due to the adversary of levels ℓ with $T_\ell/\Delta \geq 1$, as the complementary case

is easy. Any matched edge is created by the algorithm by first determining its level, and only then performing the sampling. If the edge is matched at level ℓ, it is chosen uniformly at random from between $(1-\epsilon)\cdot\gamma^\ell$ and γ^ℓ edges. We thus fix some level ℓ with $T_\ell/\Delta \geq 1$, and focus on the matched edges at that level. Invariant 6 holds for level ℓ, thus the samples of all level-ℓ edges always contain with probability (w.p.) 1 at least $(1-2\epsilon)\cdot\gamma^\ell$ edges.

Consider any time step t, and let V_t be the set of vertices of level ℓ at time t. Let $A_t = A'_t \cap A''_t$, where A'_t is the event that $|V_t| = \Omega(\log^4 n/\epsilon^3)$ and A''_t is the event that an $\Omega(\epsilon)$-fraction of the vertices of V_t are TF due to the adversary at time t. We argue that $\mathbb{P}(A_t) = O(n^{-c+2})$, for some (big enough) constant c. This assertion, which is given as Lemma 6.1 i.t.f.v., is central in our proof of the almost-maxiamlity guarantee, and the almost-maximality guarantee is derived as a simple corollary (Theorems 6.5 and 6.6 i.t.f.v.). Next, we give some insights behind the proof.

Any matched edge is sampled uniformly at random from between $(1-\epsilon)\cdot\gamma^\ell$ and γ^ℓ edges. Consider the edges of the sample $S^*(e)$ of e in the order they are deleted by the adversary, *even after the edge is removed from the matching*, either by the adversary or by the algorithm. A matched edge is called *bad* if it is one of the first (at most) $2\epsilon\cdot\gamma^\ell$ edges in this ordering; otherwise it is good. Invariant 6 guarantees that the samples of all level-ℓ edges always contain $\geq (1-2\epsilon)\cdot\gamma^\ell$ edges, so at most $2\epsilon\cdot\gamma^\ell$ edges are deleted from the sample of any matched edge (while it is matched). It follows that a good edge cannot get deleted by the adversary while it is matched (hereafter, get *hit*); a bad edge may get hit.

The probability of an edge to be bad is $\leq \frac{2\epsilon\cdot\gamma^\ell}{(1-\epsilon)\cdot\gamma^\ell}$, which is at most 4ϵ for all $\epsilon < 1/2$. Our argument, alas, is not applied on all matched edges created since the algorithm's outset, but rather on a subset of edges that are matched at a certain time step t', and there are dependencies on previous coin flips of our algorithm, which are the result of edges being removed from the matching by the update algorithm itself (not the adversary). Indeed, given that some edge e is matched at time t', the sample of e may be significantly reduced, which could increase the probability of e being bad. To overcome this hurdle, we use `shuffle-schedule` to show that the fraction of bad edges at any time is $O(\epsilon)$ w.h.p. To this end, we apply a game, hereafter the *shuffling game*, where in each step a single edge is either added or deleted (starting with no edges) by the following players: (1) `Adder`: adds an edge, which is bad w.p. $\leq 4\epsilon$, (2) `Shuffler`: deletes an edge uniformly at random among the existing edges, (3) `Malicious`: deletes a good edge. A newly created matched edge is bad w.p. $\leq 4\epsilon$, thus `Adder` assumes the role of creating matched edges by the algorithm, and so only an $O(\epsilon)$-fraction of the matched edges created by `Adder` are bad w.h.p. `Shuffler` assumes the role of `shuffle-schedule` in the algorithm, deleting matched edges uniformly at random. If the fraction of bad edges during some time interval is $\Theta(\epsilon)$, the fraction of bad edges deleted by `Shuffler` in this interval is $\Theta(\epsilon)$ w.h.p., hence `Shuffler` does not change the fraction of bad edges by too much. The role of `Malicious` is not to model the exact behavior of the other (non-shuffled) parts of the algorithm that remove matched edges, but rather to capture the worst-case scenario that might happen. We show that the affect of `Malicious` to the game is negligible, which implies that even if the other (non-shuffled) parts were to delete only good edges, the fraction of bad edges would be $O(\epsilon)$; formally, we prove that the fraction of bad edges at any step t' is w.h.p. $O(\epsilon)$. This proof, provided in Section 6.2 i.t.f.v, is nontrivial and makes critical use of the property that `Shuffler` is faster than `Malicious` by a logarithmic factor; we then show that the parties corresponding to `Shuffler` and `Malicious` in the algorithm indeed satisfy this property (Lemma 6.2 i.t.f.v.).

Equipped with this bound on the fraction of bad edges at any time step, we consider the last time t' prior to t in which the queue Q_ℓ of TF level-ℓ vertices is empty, i.e., Q_ℓ is non-empty in the entire time interval $[t'+1, t]$, thus `free-schedule`$_\ell$ is never idle during that

time. We need to bound the fraction of bad edges not only among the ones matched at time t', but also among those that get matched between times t' and t. The fraction of bad edges among those matched at time t' is $O(\epsilon)$ w.h.p. by the shuffling game; as for those that get created later on, there is no dependency on coin flips that the algorithm made prior to time t', and so the probability of any of those edges to be bad is $\leq 4\epsilon$, independently of whether previously created matched edges are bad, and by Chernoff we get that the fraction of bad edges among them is $O(\epsilon)$ too. The formal proof for this bound on the fraction of bad edges among those is provided in Section 6.3 i.t.f.v., and it implies that only an $O(\epsilon)$-fraction of all those edges may get hit w.h.p., and thus get into the queue. This bound, however, does not suffice to argue that the number of vertices in Q_ℓ at time t is an $O(\epsilon)$-fraction of the matching size, due to edges that get deleted from the matching by the algorithm itself. Nonetheless, since `free-schedule`$_\ell$ is no slower than the other sub-schedulers (as its simulation parameter is Δ'), we show in Section 6.3 i.t.f.v. that it removes vertices from Q_ℓ in the interval $[t'+1, t]$ at least at the same rate as matched edges get deleted by the algorithm. By formalizing these assertions and carefully combining them, we conclude with the required result, and with the almost-maximality guarantee as a corollary. The deterministic worst-case update time follows from the description of the algorithm (refer to the first paragraph of page 8).

References

1. Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. *CoRR*, abs/1711.06625, 2017.
2. Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In *Proc. of 52nd FOCS*, pages 383–392, 2011 (see also *SICOMP'15* version, and subsequent erratum).
3. Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *Proc. 42nd ICALP*, pages 167–179, 2015.
4. Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proc. of 26th SODA*, pages 692–711, 2016.
5. Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *Proc. 26th SODA*, pages 785–804, 2015.
6. Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *Proc. 48th STOC*, pages 398–411, 2016.
7. Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In *Proc. of 28th SODA*, pages 470–489, 2017.
8. Larry Carter and Mark N. Wegman. Universal classes of hash functions. In *Proc. 9th STOC*, pages 106–112, 1977.
9. Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial barrier for worst-case time bounds. *CoRR*, abs/1711.06883, 2017.
10. Paul F Dietz and Rajeev Raman. Persistence, amortization and randomization. In *Proc. of 2nd SODA*, pages 78–88, 1991.
11. Paul F. Dietz and Daniel Dominic Sleator. Two algorithms for maintaining order in a list. In *Proc. of 19th STOC*, pages 365–372, 1987.
12. Manoj Gupta and Richard Peng. Fully dynamic $(1+\epsilon)$-approximate matchings. In *54th FOCS*, pages 548–557, 2013.

13 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proc. of 24th SODA*, pages 1131–1142, 2013.

14 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

15 Christos Levcopoulos and Mark H. Overmars. A balanced search tree with O (1) worst-case update time. *Acta Inf.*, 26(3):269–277, 1988.

16 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Proc. 45th STOC*, pages 745–754, 2013.

17 Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proc. of 42nd STOC*, pages 457–464, 2010.

18 David Peleg and Shay Solomon. Dynamic $(1 + \epsilon)$-approximate matchings: A density-sensitive approach. In *Proc. of 26th SODA*, pages 712–729, 2016.

19 Shay Solomon. Fully dynamic maximal matching in constant update time. In *Proc. 57th FOCS*, pages 325–334, 2016.

20 Shay Solomon. Dynamic approximate matchings with an optimal recourse bound. *CoRR*, abs/1803.05825, 2018.

On Estimating Edit Distance: Alignment, Dimension Reduction, and Embeddings

Moses Charikar[1]
Department of Computer Science, Stanford University, Stanford, CA, USA
moses@cs.stanford.edu

Ofir Geri[2]
Department of Computer Science, Stanford University, Stanford, CA, USA
ofirgeri@cs.stanford.edu

Michael P. Kim[3]
Department of Computer Science, Stanford University, Stanford, CA, USA
mpk@cs.stanford.edu

William Kuszmaul
Department of Computer Science, Stanford University, Stanford, CA, USA
kuszmaul@cs.stanford.edu

Abstract

Edit distance is a fundamental measure of distance between strings and has been widely studied in computer science. While the problem of estimating edit distance has been studied extensively, the equally important question of actually producing an alignment (i.e., the sequence of edits) has received far less attention. Somewhat surprisingly, we show that any algorithm to estimate edit distance can be used in a black-box fashion to produce an approximate alignment of strings, with modest loss in approximation factor and small loss in run time. Plugging in the result of Andoni, Krauthgamer, and Onak, we obtain an alignment that is a $(\log n)^{O(1/\varepsilon^2)}$ approximation in time $\tilde{O}(n^{1+\varepsilon})$.

Closely related to the study of approximation algorithms is the study of metric embeddings for edit distance. We show that min-hash techniques can be useful in designing edit distance embeddings through three results: (1) An embedding from Ulam distance (edit distance over permutations) to Hamming space that matches the best known distortion of $O(\log n)$ and also implicitly encodes a sequence of edits between the strings; (2) In the case where the edit distance between the input strings is known to have an upper bound K, we show that embeddings of edit distance into Hamming space with distortion $f(n)$ can be modified in a black-box fashion to give distortion $O(f(\text{poly}(K)))$ for a class of periodic-free strings; (3) A randomized dimension-reduction map with contraction c and asymptotically optimal expected distortion $O(c)$, improving on the previous $\tilde{O}(c^{1+2/\log\log\log n})$ distortion result of Batu, Ergun, and Sahinalp.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases edit distance, alignment, approximation algorithms, embedding, dimension reduction

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.34

Related Version A full version is available at `https://arxiv.org/abs/1804.09907`.

[1] Supported by NSF grant CCF-1617577 and a Simons Investigator Award.
[2] Supported by NSF grant CCF-1617577, a Simons Investigator Award, and the Google Graduate Fellowship in Computer Science in the School of Engineering at Stanford University.
[3] Supported by NSF grant CCF-1763299.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 34; pp. 34:1–34:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The *edit distance* $\Delta_{\text{ed}}(x, y)$ between two strings x and y is the minimum number of character insertions, deletions, and substitutions needed to transform x into y. This is a fundamental distance measure on strings, extensively studied in computer science [2–5, 9, 11, 17, 21, 24]. Edit distance has applications in areas including computational biology, signal processing, handwriting recognition, and image compression [19]. One of its oldest and most important uses is as a tool for comparing differences between genetic sequences [1, 19, 20].

The textbook dynamic-programming algorithm for edit distance runs in time $O(n^2)$ [20, 23, 24], and can be leveraged to recover a sequence of edits, also known as an *alignment*. The quadratic run time is prohibitively large for massive datasets (e.g., genomic data), and conditional lower bounds suggest that no strongly subquadratic time algorithm exists [5].

The difficulty of computing edit distance has motivated the development of fast heuristics [1, 10, 14, 19]. On the theoretical side, the tradeoff between run time and approximation factor (or distortion) is an important question (see [15, Section 6], and [16, Section 8.3.2]). Andoni and Onak [4] (building on beautiful work of Ostrovsky and Rabani [21]) gave an algorithm that estimates edit distance within a factor of $2^{\tilde{O}(\sqrt{\log n})}$ in time $n^{1+o(1)}$. The current best known tradeoff was obtained by Andoni, Krauthgamer and Onak [3], who gave an algorithm that estimates edit distance to within factor $(\log n)^{O(1/\varepsilon)}$ with run time $O(n^{1+\varepsilon})$.

Alignment Recovery

While these algorithms produce estimates of edit distance, they do not produce an alignment between strings (i.e., a sequence of edits). By decoupling the problem of numerical estimation from the problem of alignment recovery, the authors of [4] and [3] are able to exploit techniques such as metric space embeddings[4] and random sampling in order to obtain better approximations. The algorithm of [3] runs in phases, with the i-th phase distinguishing between whether $\Delta_{\text{ed}}(x, y)$ is greater than or significantly smaller than $\frac{n}{2^i}$. At the beginning of each phase, a nuanced random process is used to select a small fraction of the positions in x, and then the entire phase is performed while examining only those positions. In total, the full algorithm samples an $\tilde{O}\left(\frac{n^{\varepsilon}}{\Delta_{\text{ed}}(x,y)}\right)$-fraction of the letters in x. Note that this is a polynomially small fraction as we are interested in the case where $\Delta_{\text{ed}}(x, y) > n^{1/2}$ (if the edit distance is small, we can run the algorithm of Landau et al. [18] in linear time). Given that the algorithm only views a small portion of the positions in x, it is not clear how to recover a global alignment between the two strings.

We show, somewhat surprisingly, that any edit distance estimator can be turned into an approximate aligner in a black box fashion with modest loss in approximation factor and small loss in run time. For example, plugging the result of [3] into our framework, we get an algorithm with distortion $(\log n)^{O(1/\varepsilon^2)}$ and run time $\tilde{O}(n^{1+\varepsilon})$. To the best of our knowledge, the best previous result that gave an approximate alignment was the work of Batu, Ergun, and Sahinalp [7], which has distortion that is polynomial in n.

[4] The algorithm of [4] has a recursive structure in which at each level of recursion, every substring α of some length l is assigned a vector v_α such that the ℓ_1 distance between vectors closely approximates edit distance between substrings. The vectors at each level of recursion are constructed from the vectors in lower levels through a series of procedures culminating in an application of Bourgain's embedding to a sparse graph metric. As a result, although the distances between vectors in the top level of recursion allow for a numerical estimation of edit distance, it is not immediately clear how one might attempt to extract additional information from the vectors in order to recover an alignment.

Embeddings of Edit Distance Using Min-Hash Techniques

The study of approximation algorithms for edit distance closely relates to the study of embeddings [4, 7, 21]. An *embedding* from a metric space M_1 to a metric space M_2 is a map of points in M_1 to M_2 such that distances are preserved up to some factor D, known as the *distortion*. Loosely speaking, low-distortion embeddings from a complex metric space M_1 to a simpler metric space M_2 allow algorithm designers to focus on the simpler metric space, rather than directly handling the more complex one. Embeddings from edit distance to Hamming space have been widely studied [8, 9, 11, 21] and have played pivotal roles in the development of approximation algorithms [4] and streaming algorithms [8, 9].

The second contribution of this paper is to introduce new algorithms for three problems related to embeddings for edit distance. The algorithms are unified by the use of min-hash techniques to select pivots in strings. We find this technique to be particularly useful for edit distance because hashing the content of strings allows us to split strings in places where their content is aligned, thereby getting around the problem of insertions and deletions misaligning the strings. In several of our results, this allows us to obtain algorithms which are either more intuitive or simpler than their predecessors. The three results are summarized below.

Efficiently Embedding the Ulam Metric into Hamming Space: For the special case of the Ulam metric (edit distance on permutations), we present a randomized embedding ϕ of permutations of size n to poly(n)-dimensional Hamming space with distortion $O(\log n)$. Given strings x and y, the Hamming differences between $\phi(x)$ and $\phi(y)$ not only approximate the edit distance between x and y, but also implicitly encode a sequence of edits from x to y. If the output string of our embedding is stored using a sparse vector representation, then the embedding can be computed in linear time, and its output can be stored in linear space. The logarithmic distortion matches that of Charikar and Krauthgamer's embedding into ℓ_1-space [11], which did not encode the actual edits and needed quadratic time and number of dimensions. Our embedding also supports efficient updates, and can be modified to reflect an edit in expected time $O(\log n)$ (as opposed to the deterministic linear time of [11]).

Embedding Edit Distance in the Low-Distance Regime: Recently, there has been considerable attention devoted to edit distance in the low-distance regime [8, 9]. In this regime, we are interested in finding algorithms that run faster or perform better given the promise that the edit distance between the input strings is small. This regime is of considerable interest from the practical point of view. Landau, Myers and Schmidt [18] gave an exact algorithm for strings with edit distance K that runs in time $O(n+K^2)$. Recently, Chakraborty, Goldenberg and Koucký [9] gave a randomized embedding of edit distance into Hamming space that has distortion linear in the edit distance with probability at least $2/3$.

Given an embedding with distortion $\gamma(n)$ (a function of the input size), could one obtain an embedding whose distortion is a function of K, the edit distance, instead of n? We answer this question in the affirmative for the class of (D, R)-periodic free strings. We say that a string is (D, R)-periodic free if none of its substrings of length D are periodic with period of at most R. For $D \in$ poly(K) and $R = O(K^3)$, we show that the embedding of Ostrovksy and Rabani [21] can be used in a black-box fashion to obtain an embedding with distortion $2^{O\left(\sqrt{\log K \log\log K}\right)}$ for (D, R)-periodic free strings with edit distance of at most K. Our result can be seen as building on the min-hash techniques of [11, Section 3.5] (which in turn extends ideas from [6]). The authors of [11] give an embedding for $(t, 180tK)$-non-repetitive strings with distortion $O(t \log(tK))$ [11]. The key difference is that our notion of (D, R)-periodic free is much less restrictive than the notion of non-repetitive strings studied in [11].

Optimal Dimension Reduction for Edit Distance: The aforementioned work of Batu et al. [7] introduced and studied an interesting notion of dimension reduction for edit distance: An embedding of edit distance on length-n strings to edit distance on length-n/c strings (with larger alphabet size) is called a *dimension-reduction map* with *contraction* c. By first performing dimension reduction, one can then apply inefficient algorithms to the contracted strings at a relatively small overall cost. This idea was used in [7] to design an approximation algorithm with approximation factor $O(n^{1/3+o(1)})$. We provide a dimension-reduction map with contraction c and asymptotically optimal expected distortion $O(c)$, improving on the distortion of $\tilde{O}(c^{1+2/\log\log\log n})$ obtained by the deterministic map of [7].[5]

2 Preliminaries

Throughout the paper, we will use Σ to denote an alphabet,[6] and Σ^n to denote the set of words of length n over that alphabet. Additionally, we use $\mathcal{P}_n$ to denote the set of permutations of length n over Σ, or equivalently, the subset of Σ^n containing words whose letters are distinct. Given a string w of length n, we denote its letters by $w_1, w_2, \ldots, w_n$, and we use $w[i:j]$ to denote the substring $w_i w_{i+1} \cdots w_j$ (which is empty if $j < i$).

An *edit operation* is either an insertion, a deletion, or a substitution of a letter with another letter in a word. Given words x and y, an *alignment* from x to y is a sequence of edits transforming x to y. The *edit distance* $\Delta_{\text{ed}}(x, y)$ is the minimum number of edits needed to transform x to y. Alternatively, it is the length of an optimal alignment.

For convenience, we make the Simple Uniform Hashing Assumption [12], which assumes access to a fully independent family $\mathcal{H}$ of hash functions mapping $\Theta(\log n)$ bits to $\Theta(\log n)$ bits with constant time evaluation. For our applications, this can be simulated using the family of Pagh and Pagh [22], which is independent on any given set of size n with high probability. The family can be constructed in linear time and uses $O(n \log n)$ random bits.

3 Alignment Recovery Using a Black-Box Approximation Algorithm

In this section, we show how to transform a black-box edit distance approximation algorithm $\mathcal{A}$ into an approximate alignment algorithm $\mathcal{B}$. The algorithm $\mathcal{B}$ appears here as Algorithm 1. In the description of the algorithm, we rely on the following definition of a partition.

▶ **Definition 3.1.** A *partition* of a string u into m parts is a tuple $P = (p_0, p_1, p_2, \ldots, p_m)$ such that $p_0 = 0$, $p_m = |u|$, and $p_0 \le p_1 \le \cdots \le p_m$. For $i \in \{1, \ldots, m\}$, the *i-th part of P* is the subword $P_i := u[p_{i-1} + 1 : p_i]$, which is empty if $p_i = p_{i-1}$. A partition of a string u into m parts is an *equipartition* if each of the parts is of size either $\lfloor |u|/m \rfloor$ or $\lceil |u|/m \rceil$.

Formally, we assume that the approximation algorithm $\mathcal{A}$ has the following properties:

1. There is some non-decreasing function γ such that for all $n > 0$, and for any two strings u, v with $|u| + |v| \le n$, $\Delta_{\text{ed}}(u, v) \le \mathcal{A}(u, v) \le \gamma(n) \cdot \Delta_{\text{ed}}(u, v)$.
2. $\mathcal{A}(u, v)$ runs in time at most $T(n)$ for some non-decreasing function T which is super-additive in the sense that $T(j) + T(k) \le T(j + k)$ for $j, k \ge 0$.

We are now ready to state the main theorem of this section.

[5] When comparing these distortions, one should note that $2/\log\log\log n$ goes to zero very slowly; in particular, $c^{1+2/\log\log\log n} \ge c^{1.66}$ for all $n \le 10^{82}$, the number of atoms in the universe.

[6] We assume that characters in Σ can be represented in $\Theta(\log n)$ bits, where n is the size of input strings.

Algorithm 1 Black-Box Approximate Alignment Algorithm.

Input: Strings u, v with $|u| + |v| \leq n$.
Parameters: $m \in \mathbb{N}$ satisfying $m \geq 2$ and an approximation algorithm $\mathcal{A}$ for edit distance.

1. If $|u| \leq 1$, then find an optimal alignment in time $O(|v|)$ naively.
2. Let $P = (p_0, p_1, \ldots, p_m)$ be an equipartition of u.
3. Let S consist of the positions in v which can be reached by adding or subtracting a power of $(1 + \frac{1}{m})$ to some p_i. Formally, define

$$S = \left(\{p_0, \ldots, p_m\} \cup \{|v|\} \cup \left\{ \left\lceil p_i \pm \left(1 + \frac{1}{m}\right)^j \right\rceil \mid i, j \geq 0 \right\} \right) \cap \{0, \ldots, |v|\}.$$

4. Using dynamic programming, find a partition $Q = (q_0, \ldots, q_m)$ of v such that each q_i is in S, and such that the cost $\sum_{i=1}^{m} \mathcal{A}(P_i, Q_i)$ is minimized:
   a. For $l \in S$, let $f(l, j)$ be the subproblem of returning a choice of $q_0, q_1, \ldots, q_j$ with $q_j = l$ which minimizes $\sum_{i=1}^{j} \mathcal{A}(P_i, Q_i)$.
   b. Solve $f(l, j)$ by examining precomputed answers for subproblems of the form $f(l', j-1)$ with $l' \leq l \in S$: if $f(l', j-1)$ gives a choice of $q_0, q_1, \ldots, q_{j-1}$ with $\sum_{i=1}^{j-1} \mathcal{A}(P_i, Q_i) = t$, then we can set $q_j = l$ to get $\sum_{i=1}^{j} \mathcal{A}(P_i, Q_i) = t + \mathcal{A}(P_j, v[l' + 1 : l])$.
   (Here, $\mathcal{A}(P_j, v[l' + 1 : l])$ is computed using $\mathcal{A}$.)
5. Recurse on each pair (P_i, Q_i). Combine the resulting alignments between each P_i and Q_i to obtain an alignment between u and v.

▶ **Theorem 3.2.** *For all u, v with $|u| + |v| \leq n$ and $m \geq 2$, Algorithm 1 outputs an alignment from u to v of size at most $(3\gamma(n))^{O(\log_m n)} \cdot \Delta_{ed}(u, v)$. Moreover, the run time is $\tilde{O}(m^5 \cdot T(n))$.*

Before continuing, we provide a brief discussion of Algorithm 1. The algorithm first breaks u into a partition P of m equal parts. It then uses the black-box algorithm $\mathcal{A}$ to search for a partition Q of v such that $\sum_i \Delta_{\text{ed}}(P_i, Q_i)$ is near minimal; after finding such a Q, the algorithm recurses to find approximate alignments between P_i and Q_i for each i. Rather than considering every option for the partition $Q = (q_0, \ldots, q_m)$, the algorithm limits itself to those for which each q_i comes from a relatively small set S.

The set S is carefully designed so that although it is small, any optimal partition Q^{opt} of v can be in some sense well approximated by some partition Q using only q_i values from S. This limits the multiplicative error introduced at each level of recursion to be bounded by $3\gamma(n)$; across the $O(\log_m n)$ level of recursion, the total multiplicative error becomes $3\gamma(n)^{O(\log_m n)}$. The fact that the recursion depth appears in the exponent of the multiplicative error is why we partition u and v into many parts at each level.

Next we discuss several implications of Theorem 3.2. The parameter m allows us to trade off the approximation factor and the run time of the algorithm. When taken to the extreme, this gives two particularly interesting results.

▶ **Corollary 3.3.** *Let $0 < \varepsilon < 1$ (not necessarily constant). Then m can be chosen so that Algorithm 1 has approximation ratio $(3\gamma(n))^{O(\frac{1}{\varepsilon})}$ and run time $\tilde{O}\left(T(n) \cdot n^{\varepsilon}\right)$.*

▶ **Corollary 3.4.** *Let $0 < \varepsilon < 1$ (not necessarily constant). Then m can be chosen so that Algorithm 1 has approximation ratio $n^{O(\varepsilon)}$ and run time $\tilde{O}(T(n)) \cdot (3\gamma(n))^{O(1/\varepsilon)}$.*

We can apply Corollary 3.3 to the algorithm of Andoni et al. [3] with approximation ratio $(\log n)^{O(1/\varepsilon)}$ and run time $O(n^{1+\varepsilon})$ as follows. (Note that ε may be $o(1)$.)

▶ **Corollary 3.5.** *There exists an approximate-alignment algorithm which runs in time* $\tilde{O}(n^{1+\varepsilon})$*, and has approximation factor* $(\log n)^{O(1/\varepsilon^2)}$ *with probability* $1 - \frac{1}{\text{poly}(n)}$.

3.1 Proof of Theorem 3.2

The proof of the theorem will follow from Proposition 3.6, which bounds the run time of Algorithm 1, and Proposition 3.11, which bounds the approximation ratio.

Throughout this section, let u, v and m be the values given to Algorithm 1. Let $P = (p_0, \ldots, p_m)$ be the equipartition of u, and let S be the set defined by Algorithm 1.

▶ **Proposition 3.6.** *Algorithm 1 runs in time* $\tilde{O}(T(|u| + |v|) \cdot m^5)$.

Proof. If $|u| \leq 1$, then we can find an optimal alignment in time $O(|v|)$ naively. Suppose $|u| > 1$. Notice that $|S| \leq O(m^2 \log n)$. In particular, because $(1 + \frac{1}{m})^{(m+1)\ln n} \geq n$,

$$S \subseteq \{p_0, \ldots, p_m\} \cup \{|v|\} \cup \left\{ \left\lceil p_i \pm \left(1 + \frac{1}{m}\right)^j \right\rceil \mid i \in [0:m], j \in [0:(m+1)\ln n] \right\},$$

which has size at most $O(m^2 \log n)$.

Finding an equipartition of u can be done in linear time, and constructing S takes time $O(|S|) = \tilde{O}(m^2)$. In order to perform the fourth step which selects Q, we must compute $f(l, j)$ for each $l \in S$ and $j \in [0:m]$. To evaluate $f(l, j)$, we must consider each $l' \in S$ satisfying $l' \leq l$, and then compute the cost of $f(l', j-1)$ plus $\mathcal{A}(P_j, v[l'+1:l])$ (which takes time at most $T(|u| + |v|)$ to compute). Therefore, each $f(l, j)$ is computed in time $O(|S| \cdot T(|u|+|v|)) \leq \tilde{O}(T(|u|+|v|) \cdot m^2)$. Because there are $O(m \cdot |S|) = \tilde{O}(m^3)$ subproblems of the form $f(l, j)$, the total run time of the dynamic program is $\tilde{O}(T(|u| + |v|) \cdot m^5)$.

So far we have shown that the first level of recursion takes time $\tilde{O}(T(|u| + |v|)m^5)$. The sum of the lengths of the inputs to Algorithm 1 at a particular level of the recursion is at most $|u| + |v|$. It follows by the super-additivity of $T(n)$ that the time spent in any given level of recursion is at most $\tilde{O}(T(|u| + |v|)m^5)$. Because each level of recursion reduces the sizes of the parts of u by a factor of $\Omega(m)$, the number of levels is at most $O(\log_m n) \leq O(\log n)$. Therefore, the run time is $\tilde{O}(T(|u| + |v|) \cdot m^5)$. ◀

When discussing the approximation ratio of Algorithm 1, it will be useful to have a notion of edit distance between partitions of strings.

▶ **Definition 3.7.** Given two partitions $C = (c_0, \ldots, c_m)$ and $D = (d_0, \ldots, d_m)$ of strings a and b respectively, we define $\Delta_{\text{ed}}(C, D) := \sum_i \Delta_{\text{ed}}(C_i, D_i)$.

In order to bound the approximation ratio of Algorithm 1, we will introduce, for the sake of analysis, a partition $Q^{\text{opt}} = (q_0^{\text{opt}}, \ldots, q_m^{\text{opt}})$ of v satisfying $\Delta_{\text{ed}}(P, Q^{\text{opt}}) = \Delta_{\text{ed}}(u, v)$. Recall that P is fixed, which allows us to use it in the definition of Q^{opt}.

We claim that some partition Q^{opt} satisfying $\Delta_{\text{ed}}(P, Q^{\text{opt}}) = \Delta_{\text{ed}}(u, v)$ must exist. If u and v differed by only a single edit, one could start from P and explicitly define Q^{opt} so that $\Delta_{\text{ed}}(P, Q^{\text{opt}}) = 1$ (by a case analysis of which type of edit was performed). It can then be shown by induction on the number of edits that, in general, we can obtain a partition Q^{opt} satisfying $\Delta_{\text{ed}}(P, Q^{\text{opt}}) = \Delta_{\text{ed}}(u, v)$.

Our strategy for bounding the approximation ratio will be to compare $\Delta_{\text{ed}}(P, Q)$ for the partition Q selected by our algorithm to $\Delta_{\text{ed}}(P, Q^{\text{opt}})$. We do this through three observations.

The first observation upper bounds $\Delta_{\text{ed}}(P, Q)$. Informally, it shows that the cost in edit distance which Algorithm 1 pays for selecting Q instead of Q^{opt} is at most $2\sum_{i=0}^{m} |q_i - q_i^{\text{opt}}|$.

▶ **Lemma 3.8.** *Let $Q = (q_0, \ldots, q_m)$ be a partition of v. Then*

$$\Delta_{ed}(P, Q) \leq \Delta_{ed}(u, v) + 2\sum_{i=1}^{m} |q_i - q_i^{opt}|.$$

Proof. Observe that

$$\Delta_{\text{ed}}(P, Q) \leq \Delta_{\text{ed}}(P, Q^{\text{opt}}) + \Delta_{\text{ed}}(Q^{\text{opt}}, Q) = \Delta_{\text{ed}}(u, v) + \sum_{i=1}^{m} \Delta_{\text{ed}}(Q_i^{\text{opt}}, Q_i).$$

Because Q and Q^{opt} are both partitions of v, $\Delta_{\text{ed}}(Q_i, Q_i^{\text{opt}}) \leq |q_{i-1} - q_{i-1}^{\text{opt}}| + |q_i - q_i^{\text{opt}}|$. In particular, $|q_{i-1} - q_{i-1}^{\text{opt}}|$ insertions to the left side of one of Q_i or Q_i^{opt} (whichever has its start point further to the right) will result in the two substrings having the same start-point; and then $|q_i - q_i^{\text{opt}}|$ insertions to the right side of one of Q_i or Q_i^{opt} (whichever has its end point further to the left) will result in the two substrings having the same end-point. Thus

$$\begin{aligned}\Delta_{\text{ed}}(u, v) + \sum_{i=1}^{m} \Delta_{\text{ed}}(Q_i^{\text{opt}}, Q_i) &\leq \Delta_{\text{ed}}(u, v) + \sum_{i=1}^{m} |q_{i-1} - q_{i-1}^{\text{opt}}| + |q_i - q_i^{\text{opt}}| \\ &\leq \Delta_{\text{ed}}(u, v) + 2\sum_{i=1}^{m} |q_i - q_i^{\text{opt}}|,\end{aligned}$$

where we are able to disregard the case of $i = 0$ because $q_0 = q_0^{\text{opt}} = 0$. ◀

The next observation establishes a lower bound for $\Delta_{\text{ed}}(u, v)$.

▶ **Lemma 3.9.** $\Delta_{ed}(u, v) \geq \frac{1}{m}\sum_{i=1}^{m} |p_i - q_i^{opt}|$.

Proof. Because $\Delta_{\text{ed}}(P, Q^{\text{opt}}) = \Delta_{\text{ed}}(u, v)$, we must have that for each $i \in [m]$,

$$\Delta_{\text{ed}}(u, v) = \Delta_{\text{ed}}(u[1 : p_i], v[1 : q_i^{\text{opt}}]) + \Delta_{\text{ed}}(u[p_i + 1 : |u|], v[q_i^{\text{opt}} + 1 : |v|]).$$

Notice, however, that the strings $u[1 : p_i]$ and $v[1 : q_i^{\text{opt}}]$ differ in length by at least $|q_i^{\text{opt}} - p_i|$. Therefore, their edit distance must be at least $|q_i^{\text{opt}} - p_i|$, implying that $\Delta_{\text{ed}}(u, v) \geq |q_i^{\text{opt}} - p_i|$.

Thus $\frac{1}{m}\Delta_{\text{ed}}(u, v) \geq \frac{1}{m}|q_i^{\text{opt}} - p_i|$. Summing over $i \in [m]$ gives the desired equation. ◀

So far we have shown that the cost in edit distance which Algorithm 1 pays for selecting Q instead of Q^{opt} is at most $2\sum_{i=0}^{m} |q_i - q_i^{\text{opt}}|$ (Lemma 3.8), and that the edit distance from u to v is at least $\frac{1}{m}\sum_{i=1}^{m} |p_i - q_i^{\text{opt}}|$ (Lemma 3.9). Next we compare these two quantities. In particular, we show that if Q is chosen to mimic Q^{opt} as closely as possible, then each of the $|q_i - q_i^{\text{opt}}|$ will become small relative to each of the $|p_i - q_i^{\text{opt}}|$.

▶ **Lemma 3.10.** *There exists a partition $Q = (q_0, \ldots, q_m)$ of v such that each q_i is in S, and such that for each $i \in [0 : m]$, $|q_i - q_i^{opt}| \leq \frac{1}{m}|p_i - q_i^{opt}|$.*

Proof Sketch. Consider the partition Q in which q_i is chosen to be the largest $s \in S$ satisfying $s \leq q_i^{\text{opt}}$. Observe that: (i) because $0 \in S$, each q_i always exists; (ii) because $|v| \in S$, we will have $q_m = |v|$; and (iii) because $q_0^{\text{opt}} \leq q_1^{\text{opt}} \leq \cdots \leq q_m^{\text{opt}}$, we will have that $q_0 \leq q_1 \leq \cdots \leq q_m$. Therefore, Q is a well-defined partition of v.

It remains to prove $|q_i - q_i^{\text{opt}}| \leq \frac{1}{m}|p_i - q_i^{\text{opt}}|$. For brevity, we focus on the case of $p_i < q_i^{\text{opt}}$. The other cases are conceptually similar and appear in the full version of this paper.

Assume $p_i < q_i^{\text{opt}}$. Consider the largest non-negative integer j such that $p_i + (1+\frac{1}{m})^j \leq q_i^{\text{opt}}$. One can verify that by definition of j, we must have

$$\left(1+\frac{1}{m}\right)^j \leq q_i^{\text{opt}} - p_i \leq \left(1+\frac{1}{m}\right)^{j+1}. \tag{3.1}$$

It follows that

$$q_i^{\text{opt}} - \left(p_i + \left(1+\frac{1}{m}\right)^j\right) \leq \left(1+\frac{1}{m}\right)^{j+1} - \left(1+\frac{1}{m}\right)^j = \frac{1}{m}\left(1+\frac{1}{m}\right)^j. \tag{3.2}$$

Since $\lceil p_i + (1+\frac{1}{m})^j \rceil \in S$, the definition of q_i ensures that q_i is between $p_i + (1+\frac{1}{m})^j$ and q_i^{opt} inclusive. Therefore, (3.2) implies $q_i^{\text{opt}} - q_i \leq \frac{1}{m}\left(1+\frac{1}{m}\right)^j$. Combining this with (3.1), it follows that $q_i^{\text{opt}} - q_i \leq \frac{1}{m}(q_i^{\text{opt}} - p_i)$, as desired. ◀

We are now equipped to bound the approximation ratio of Algorithm 1, thereby completing the proof of Theorem 3.2. We will use the preceeding lemmas to bound the approximation ratio at each level of recursion to $O(\gamma(n))$. The approximation ratio will then multiply across the $O(\log_m n)$ levels of recursion, giving total approximation ratio $O(\gamma(n))^{O(\log_m n)}$.

▶ **Proposition 3.11.** *Let $E(u,v)$ be the number of edits returned by Algorithm 1. Then*

$$\Delta_{ed}(u,v) \leq E(u,v) \leq \Delta_{ed}(u,v) \cdot (3\gamma(n))^{O(\log_m n)}.$$

Proof. Because Algorithm 1 finds a sequence of edits from u to v, clearly $\Delta_{\text{ed}}(u,v) \leq E(u,v)$. By Lemma 3.10 there is some partition $Q = (q_0, \ldots, q_m)$ of v such that each q_i is in S, and such that for each $i \in [0:m]$, $|q_i - q_i^{\text{opt}}| \leq \frac{1}{m}|p_i - q_i^{\text{opt}}|$. By Lemma 3.9, it follows that

$$\sum_{i=1}^{m} |q_i - q_i^{\text{opt}}| \leq \frac{1}{m}\sum_{i=1}^{m} |p_i - q_i^{\text{opt}}| \leq \Delta_{\text{ed}}(u,v).$$

Applying Lemma 3.8, we then get that

$$\Delta_{\text{ed}}(P,Q) \leq \Delta_{\text{ed}}(u,v) + 2\sum_{i=1}^{m} |q_i - q_i^{\text{opt}}| \leq 3\Delta_{\text{ed}}(u,v).$$

Thus there is some Q which Algorithm 1 is allowed to select such that $\Delta_{\text{ed}}(P,Q) \leq 3\Delta_{\text{ed}}(u,v)$. Since the approximation ratio of $\mathcal{A}$ is $\gamma(n)$, the true partition Q selected at the first level of recursion must satisfy $\Delta_{\text{ed}}(P,Q) \leq 3\gamma(n)\Delta_{\text{ed}}(u,v)$.

After the i-th level of recursion, u has implicitly been split into a large partition P^i, v has implicitly been split into a large partition Q^i, and the recursive subproblems are searching for edits between pairs of parts of P^i and Q^i. Since there is $3\gamma(n)$ distortion at each level, we can get by induction that $\Delta_{\text{ed}}(P^i, Q^i) \leq (3\gamma(n))^i \Delta_{\text{ed}}(u,v)$. Since there are $O(\log_m n)$ levels of recursion, the number of edits returned by the algorithm is at most $\Delta_{\text{ed}}(u,v) \cdot (3\gamma(n))^{O(\log_m n)}$. ◀

4 Embeddings and Dimension Reduction Using Min-Hash Techniques

4.1 Alignment Embeddings for Permutations

Here we present a randomized embedding from $\mathcal{P}_n$, the set of permutations of length n, into Hamming space with expected distortion $O(\log n)$. The embedding has the surprising

Algorithm 2 Alignment Embedding for Permutations.

Input: A string $w = w_1 \cdots w_n \in \mathcal{P}_n$.
Parameters: ε and $m \geq \log_{1/2+\varepsilon} \frac{1}{n} + 1$.

1. At the first level of recursion only:
   a. Initialize an array A of size $2^m - 1$ (indexed starting at one) with zeros. The array A will contain the output embedding.
   b. Select a hash function h mapping Σ to $r \log n$ bits for a sufficiently large constant r.
2. Let i minimize $h(w_i)$ out of the $i \in [n/2 - \varepsilon n : n/2 + \varepsilon n]$.[8] We call w_i the *pivot* in w.
3. Set $A[2^{m-1}] = w_i$.
4. Recursively embed $w_1 \cdots w_{i-1}$ into $A[1 : 2^{m-1} - 1]$.
5. Recursively embed $w_{i+1} \cdots w_n$ into $A[2^{m-1} + 1 : 2^m - 1]$.

property that it implicitly encodes alignments between strings. If the output is stored using run-length encoding,[7] then the size of the output and the run time are both $O(n)$.

The description of the embedding appears as Algorithm 2. For simplicity, we assume $0 \notin \Sigma$, which allows us to use 0 as a null character. The algorithm takes two parameters: ε and m. The parameter ε controls a trade-off between the distortion and the output dimension. The parameter m dictates the maximum depth of recursion that can be performed within the array A. In particular, m needs to be chosen such that the algorithm does not run out of space for the embedding in the recursive calls.

Since each recursive step takes as input words of size in the range $[(1/2-\varepsilon)n : (1/2+\varepsilon)n]$, the input size at the i-th level of the recursion is at most $(1/2+\varepsilon)^{i-1}n$. We need to choose m such that at the m-th level of recursion, the input size will be at most 1. Therefore, it suffices to pick m satisfying $m \geq \log_{1/2+\varepsilon} \frac{1}{n} + 1$.

We denote the resulting embedding of the input string w into the output array A by $\phi_{\varepsilon,m}(w)$. Moreover, for $m = \lceil \log_{1/2+\varepsilon} \frac{1}{n} + 1 \rceil$, we define $\phi_\varepsilon(w)$ to be $\phi_{\varepsilon,m}(w)$. Note that ϕ_ε embeds w into an array A of size $O\left(2^{\log_{1/2+\varepsilon} 1/n}\right) = O\left(n^{-1/\log(1/2+\varepsilon)}\right)$, which one can verify for $\varepsilon \leq \frac{1}{4}$ is $O(n^{1+6\varepsilon})$.

We call ϕ_ε an *alignment embedding* because ϕ_ε maps a string x to a copy of x spread out across an array of zeros. When we compare $\phi_\varepsilon(x)$ with $\phi_\varepsilon(y)$ by Hamming differences, ϕ_ε encodes an alignment between x and y; it pays for every letter which it fails to match up with another copy of the same letter. In particular, every pairing of a letter with a null corresponds to an insertion or deletion, and every pairing of a letter with a different letter corresponds to a substitution. Thus $\mathrm{Ham}(\phi_\varepsilon(x), \phi_\varepsilon(y))$ will always be at least $\Delta_{\mathrm{ed}}(x, y)$.

In the rest of this subsection we will prove the following theorem.

▶ **Theorem 4.1.** *For $\varepsilon \leq \frac{1}{4}$, there exists a randomized embedding ϕ_ε from $\mathcal{P}_n$ to $O(n^{1+6\varepsilon})$-dimensional Hamming space with the following properties.*

- *For $x, y \in \mathcal{P}_n$, $\phi_\varepsilon(x)$ and $\phi_{\varepsilon(y)}$ encode a sequence of $\mathrm{Ham}(\phi_\varepsilon(x), \phi_\varepsilon(y))$ edits from x to y. In particular, $\mathrm{Ham}(\phi_\varepsilon(x), \phi_\varepsilon(y)) \geq \Delta_{ed}(x, y)$.*
- *For $x, y \in \mathcal{P}_n$, $\mathbb{E}[\mathrm{Ham}(\phi_\varepsilon(x), \phi_\varepsilon(y))] \leq O\left(\frac{1}{\varepsilon} \log n\right) \cdot \Delta_{ed}(x, y)$.*
- *For $x \in \mathcal{P}_n$, $\phi_\varepsilon(x)$ is sparse in the sense that it only contains n non-zero entries. Moreover, if $\phi_\varepsilon(x)$ is stored with run-length encoding, it can be computed in time $O(n)$.*

[7] In run-length encoding, runs of identical characters are stored as a pair whose first entry is the character and the second entry is the length of the run.
[8] With high probability, there are no hash collisions.

The first property in the theorem follows from the discussion above. In order to prove $\mathbb{E}[\mathrm{Ham}(\phi_\varepsilon(x), \phi_\varepsilon(y))] \leq \Delta_{\mathrm{ed}}(x,y) O\left(\frac{1}{\varepsilon}\log n\right)$, we will consider a series of at most $2\Delta_{\mathrm{ed}}(x,y)$ insertions or deletions that are used to transform x into y. Each substitution operation can be emulated by an insertion and a deletion. Moreover, note that by ordering deletions before insertions, each of the intermediate strings will still be a permutation. In the following lemma, we bound the expected Hamming distance for just a single insertion (or equivalently, a deletion). By the triangle inequality, we get the bound on $\mathbb{E}[\mathrm{Ham}(\phi_\varepsilon(x), \phi_\varepsilon(y))]$.

▶ **Lemma 4.2.** *Let $x \in \mathcal{P}_n$ be a permutation, and let y be a permutation derived from x by a single insertion. Let $0 < \varepsilon \leq \frac{1}{4}$ and let m be large enough so that $\phi_{\varepsilon,m}$ is well-defined on x and y. Then $\mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y))] \leq O\left(\frac{1}{\varepsilon}\log n\right)$.*

Proof. Observe that the set of letters in position-range $[(1/2-\varepsilon)|x| : (1/2+\varepsilon)|x|]$ in x differs by at most $O(1)$ elements from the set of letters in position-range $[(1/2-\varepsilon)|y| : (1/2+\varepsilon)|y|]$ in y. Thus with probability $1 - O(1/(\varepsilon n))$, there will be a letter l in the overlap between the two ranges whose hash is smaller than that of any other letter in either of the two ranges. In other words, the pivot in x (i.e., the letter in the position range with minimum hash) will differ from the pivot in y with probability $O(1/(\varepsilon n))$. Therefore,

$$\begin{aligned}
\mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y))] &= \Pr[\text{pivots differ}] \cdot \mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y)) \mid \text{ pivots differ}] \\
&\quad + \Pr[\text{pivots same}] \cdot \mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y)) \mid \text{ pivots same}] \\
&\leq O\left(\frac{1}{\varepsilon n}\right) \cdot \mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y)) \mid \text{ pivots differ}] \\
&\quad + \mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y)) \mid \text{ pivots same}].
\end{aligned}$$

In general, $\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y))$ cannot exceed $O(n)$. Thus

$$\begin{aligned}
\mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y))] &\leq O\left(\frac{1}{\varepsilon n}\right) \cdot O(n) + \mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y)) \mid \text{ pivots same}] \\
&\leq O\left(\frac{1}{\varepsilon}\right) + \mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y)) \mid \text{ pivots same}].
\end{aligned}$$

If the pivot in x is the same as in y, then the insertion must take place to either the left or the right of the pivot. Clearly $\phi_{\varepsilon,m}(x)$ and $\phi_{\varepsilon,m}(y)$ will agree on the side of the pivot in which the edit does not occur. Inductively applying our argument to the side on which the edit occurs, we incur a cost of $O(1/\varepsilon)$ once for each level in the recursion. The maximum depth of the recursion is $O\left(\log_{1/2+\varepsilon}\frac{1}{n}\right) = O(\log n)$. This gives us $\mathbb{E}[\mathrm{Ham}(\phi_{\varepsilon,m}(x), \phi_{\varepsilon,m}(y))] \leq O\left(\frac{1}{\varepsilon}\right) \cdot \log n$. ◀

It remains only to analyze the run time. Notice that $\phi_\varepsilon(x)$ can be stored in space $\Theta(n)$ using run-length encoding. We can compute $\phi_\varepsilon(x)$ in time $O(n)$, as follows. Using Range Minimum Query [13] we can build a data structure which supports constant-time queries returning minimum hashes in contiguous substrings of x. This allows each recursive step in the embedding to be performed in constant time. Since each recursive step writes one of the n letters to the output, the total run time is bounded by $O(n)$.

4.2 Embedding into Hamming Space in the Low-Distance Regime

Assume we are given an embedding from edit distance in Σ^n into Hamming space with subpolynomial distortion $\gamma(n)$. We wish to use such an embedding as a black box in order to obtain a new embedding for the low edit distance regime: the new embedding, which would

Algorithm 3 Choose Next Block (Informal).

Input: A string x, the index i where the current block begins.
Output: The index where the next block begins.
Parameters: $W'' \ll W' \ll W$ set as needed.

1. Consider the window $x_i \cdots x_{i+W-1}$ of size W. Divide the second half of the window into non-overlapping sub-windows of size W', and pick one such sub-window at random.
2. Each substring of length W'' inside the sub-window is called a sub-sub-window (sub-sub-windows may overlap). Compute a hash of each of the sub-sub-windows.[9]
3. Return the start position of the sub-sub-window with the smallest hash.

be parameterized by a value K, would take any two strings $x, y \in \Sigma^n$ with $\Delta_{\text{ed}}(x, y) \leq K$ and map x and y into Hamming space with distortion $\gamma'(K)$, a function of K rather than n.

We make progress toward such an embedding with the added constraint that our strings x and y are (D, R)-periodic free for $D \in \text{poly}(K)$ of our choice and $R \in O(K^3)$. A string is (D, R)-*periodic free* if it contains no contiguous substrings of length D that are periodic with period at most R. Our embedding takes two such strings with $\Delta_{\text{ed}}(x, y) \leq K$ and maps them into Hamming space with distortion $\gamma(\text{poly}(K))$. If we select the black-box embedding to be embedding of Ostrovsky and Rabani [21], then this gives distortion $2^{O\left(\sqrt{\log K \log\log K}\right)}$.

Our main result in this section is stated as the following theorem.

▶ **Theorem 4.3.** *Suppose we have an embedding from edit distance in Σ^n to Hamming space with subpolynomial distortion $\gamma(n) \geq 2$. Let $K \in \mathbb{N}$ and pick some $D \in \text{poly}(K)$. Then there exists $R \in O(K^3)$ and an embedding α from edit distance into scaled Hamming space with the following property. For strings x and y that are (D, R)-periodic free and of edit distance at most K apart, α distorts the distance between x and y by at most $O(\gamma(K^3 D)) \leq O(\gamma(\text{poly}(K)))$ in expectation.*

The complete proof of Theorem 4.3 appears in the full version of the paper. Below we discuss the key ideas, which once again make use of min-hash techniques.

The main step in our embedding is to partition the strings x and y into parts of length $\text{poly}(K)$ in a way so that the sum of the edit distances between the parts equals $\Delta_{\text{ed}}(x, y)$ (with some probability), and then to apply the black-box embedding to each individual part.

We select the partition of x by going from left to right, and at each step choosing the next index at which the current block ends and the next one begins. The algorithm for selecting each successive block appears as Algorithm 3.

The goal of the algorithm is to find partitions of x and y that are aligned despite the edits. By picking a sub-window uniformly at random, we guarantee that with high probability (as a function of W, W') no edits occur directly within that sub-window. However, if x and y differ by an insertion or deletion prior to that sub-window, the sub-window within x may be misaligned with the sub-window within y. Nonetheless, the set of W''-letter sub-sub-windows of the sub-window of x will be almost the same as the set of W''-letter sub-sub-windows of the sub-window of y. By selecting the sub-sub-window with minimum hash as the start position for the next block, we are then able to guarantee that with high probability (as a function of W') we pick positions in x and y which are aligned with each other.

[9] By selecting W'', W', W appropriately, we can use the (D, R)-periodic free property to guarantee that these sub-sub-windows are distinct.

4.3 Dimension Reduction

A mapping of edit distance on length-n strings to edit distance on length-at-most-n/c strings (with larger alphabet size) is called a *dimension-reduction* map with *contraction* c [7]. The *distortion* of the map measures the multiplicative factor to which edit distance is preserved.

▶ **Theorem 4.4.** *There is a randomized dimension-reduction map ϕ with contraction c and expected distortion $O(c)$. In particular, for $x, y \in \Sigma^n$,*

$$\frac{1}{2c} \cdot \Delta_{ed}(x, y) \leq \Delta_{ed}(\phi(x), \phi(y)),$$

and

$$\mathbb{E}[\Delta_{ed}(\phi(x), \phi(y))] \leq O(1) \cdot \Delta_{ed}(x, y).$$

Moreover, for this definition of distortion, ϕ is within a constant factor of optimal. Additionally, ϕ can be evaluated in time $O(n \log c)$.

Note that the output of a dimension-reduction map may be over a much larger alphabet than the input. One should not be too alarmed by this, however, because alphabet reduction can be performed after the dimension reduction (by simply hashing to $\Theta(\log n)$ bits).

Below we summarize our approach to proving Theorem 4.4. The complete proof appears in the full version of the paper. When computing our dimension-reduction map $\phi(w)$ on a word w, one approach would be to split w into blocks of size c and to then define $\phi(w)$'s letters to correspond to blocks. This would achieve the desired reduction in dimension, and would have the effect that a single edit to $\phi(w)$ would correspond to at most c edits in w. However, a single insertion to w could change the content of linearly many blocks, corresponding to a large number of edits to $\phi(w)$. Thus the challenge is to instead break w into blocks in a way so that an edit to w affects only a small number of blocks.

In order to accomplish this, our actual ϕ breaks w into long periodic substrings and non-periodic substrings. The two types of substrings are then handled as follows:

Handling Long Periodic Substrings: Within the periodic substrings, we break the substring into blocks based on the periodic behavior. The key insight is that the embedding of the periodic substring will consist of the same block repeating many times. If an edit occurs in the middle of a long periodic substring, the embedding will still include the same block repeated many times, but $O(1)$ blocks around the edit will be modified. Since the blocks correspond to letters in $\phi(w)$, the edit to w results in $O(1)$ edits to $\phi(w)$.

Handling Non-Periodic Substrings: Within the non-periodic substrings, we select markers in a randomized fashion to determine block boundaries. The definition of non-periodic substrings guarantees that for every c adjacent letters, the $8c$-letter substrings $w_i w_{i+1} \cdots w_{i+8c-1}$ beginning at each of those c letters w_i are distinct. We utilize this property to select markers based on the minimum hash of $8c$-letter substrings. A letter is selected as a marker if the hash of the $8c$-letter string beginning at that letter is smaller than the hashes of any of the $8c$-letter strings beginning in the $c/2$ letters to its left or right. This localizes the selection so that edits to the string will only affect nearby markers.

When two markers are more than c apart, we additionally break the space between them into sub-blocks of size at most c. By preventing blocks in $\phi(w)$ from exceeding $O(c)$ in size, we can take a sequence of edits in $\phi(w)$ and generate a corresponding sequence of edits in w with distortion $O(c)$. When bounding the distortion in the other direction, we risk edits in

w taking place between two far-apart markers, which in turn could affect all of the blocks between those markers in $\phi(w)$. The main technical challenge is bounding the effect this has on the distortion. We do so by showing probabilistically that wherever there is an edit, there will be markers nearby which mitigate the impact of that edit on the block structure of $\phi(w)$.

References

1 Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

2 Alexandr Andoni and Robert Krauthgamer. The computational hardness of estimating edit distance. *SIAM J. Comput.*, 39(6):2398–2429, 2010.

3 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 377–386. IEEE, 2010.

4 Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM J. Comput.*, 41(6):1635–1648, 2012.

5 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the 47th Annual Symposium on Theory of Computing (STOC)*, pages 51–58. ACM, 2015.

6 Ziv Bar-Yossef, T.S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of 45th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 550–559. IEEE, 2004.

7 Tugkan Batu, Funda Ergün, and Süleyman Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the 17th Annual Symposium on Discrete Algorithms (SODA)*, pages 792–801, 2006.

8 Djamal Belazzougui and Qin Zhang. Edit distance: Sketching, streaming, and document exchange. In *Proceedings of the 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 51–60. IEEE, 2016.

9 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *Proceedings of the 48th Annual Symposium on Theory of Computing (STOC)*, pages 712–725. ACM, 2016.

10 Kun-Mao Chao, William R. Pearson, and Webb Miller. Aligning two sequences within a specified diagonal band. *Bioinformatics*, 8(5):481–487, 1992.

11 Moses Charikar and Robert Krauthgamer. Embedding the ulam metric into l_1. *Theory of Computing*, 2(11):207–224, 2006.

12 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. The MIT Press, third edition, 2009.

13 Johannes Fischer and Volker Heun. Theoretical and practical improvements on the rmq-problem, with applications to LCA and LCE. In *Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 36–48, 2006.

14 Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.

15 Piotr Indyk. Algorithmic aspects of geometric embeddings (tutorial). In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 10–33. IEEE, 2001.

16 Piotr Indyk and Jiri Matoušek. Low-distortion embeddings of finite metric spaces. *Handbook of Discrete and Computational Geometry*, page 177, 2004.

17 Robert Krauthgamer and Yuval Rabani. Improved lower bounds for embeddings into L_1. In *Proceedings of the 17th Annual Symposium on Discrete Algorithms (SODA)*, pages 1010–1017, 2006.

18 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998.

19 Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

20 Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

21 Rafail Ostrovsky and Yuval Rabani. Low distortion embeddings for edit distance. *J. ACM*, 54(5):23, 2007.

22 Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and optimal space. *SIAM Journal on Computing*, 38(1):85–96, 2008.

23 Taras K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968.

24 Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.

How Hard Is It to Satisfy (Almost) All Roommates?

Jiehua Chen
Ben-Gurion University of the Negev, Beer Sheva, Israel
jiehua.chen2@gmail.com

Danny Hermelin
Ben-Gurion University of the Negev, Beer Sheva, Israel
hermelin@bgu.ac.il

Manuel Sorge
Ben-Gurion University of the Negev, Beer Sheva, Israel
sorge@post.bgu.ac.il

Harel Yedidsion
Ben-Gurion University of the Negev, Beer Sheva, Israel
yedidsio@post.bgu.ac.il

Abstract

The classic Stable Roommates problem (the non-bipartite generalization of the well-known Stable Marriage problem) asks whether there is a *stable matching* for a given set of agents, *i.e.* a partitioning of the agents into disjoint pairs such that *no* two agents induce a *blocking pair*. Herein, each agent has a *preference list* denoting who it prefers to have as a partner, and two agents are blocking if they prefer to be with each other rather than with their assigned partners.

Since stable matchings may not be unique, we study an NP-hard optimization variant of Stable Roommates, called Egal Stable Roommates, which seeks to find a stable matching with a minimum egalitarian cost γ, *i.e.* the sum of the dissatisfaction of the agents is minimum. The *dissatisfaction* of an agent is the number of agents that this agent prefers over its partner if it is matched; otherwise it is the length of its preference list. We also study almost stable matchings, called Min-Block-Pair Stable Roommates, which seeks to find a matching with a minimum number β of blocking pairs. Our main result is that Egal Stable Roommates parameterized by γ is fixed-parameter tractable, while Min-Block-Pair Stable Roommates parameterized by β is W[1]-hard, even if the length of each preference list is at most five.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms, Theory of computation → Algorithmic game theory, Mathematics of computing → Combinatorial optimization

Keywords and phrases NP-hard problems Data reduction rules Kernelizations Parameterized complexity analysis and algorithmics

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.35

Related Version A full version of this work is available at [14], `https://arxiv.org/abs/1707.04316`.

Acknowledgements This work is supported by the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement number 631163.11 and Israel Science Foundation (grant number 551145/14).

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 35; pp. 35:1–35:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

This paper presents algorithms and hardness results for two variants of the Stable Roommates problem, a well-studied generalization of the classic Stable Marriage problem. Before describing our results, we give a brief background that will help motivate our work.

Stable Marriage and Stable Roommates. An instance of the Stable Marriage problem consists of two disjoint sets of n men and n women (collectively called *agents*), who are each equipped with his or her own personal *strict* preference list that ranks *every* member of the opposite sex. The goal is to find a bijection, or *matching*, between the men and the women that does not contain any blocking pairs. A *blocking pair* is a pair of man and woman who are not matched together but both prefer each other over their own matched partner. A matching with no blocking pairs is called a *stable matching*, and *perfect* if it is a bijection between *all* men and women.

Stable Marriage is a classic and fundamental problem in computer science and applied mathematics, and as such, entire books were devoted to it [24, 32, 50, 37]. The problem emerged from the economic field of matching theory, and it can be thought of as a generalization of the Maximum Matching problem when restricted to complete bipartite graphs. The most important result in this context is the celebrated Gale-Shapley algorithm [22]: This algorithm computes in polynomial time a perfect stable matching in *any* given instance, showing that regardless of their preference lists, there always exists a perfect stable matching between any equal number of men and women.

The Stable Marriage problem has several interesting variants. First, the preference lists of the agents may be *incomplete*, meaning that not every agent is an acceptable partner to every agent of the opposite sex. In graph theoretic terms, this corresponds to the bipartite incomplete case. The preference lists could also have *ties*, meaning that two or more agents may be considered equally good as partners. Finally, the agents may not be partitioned into two disjoint sets, but rather each agent may be allowed to be matched to any other agent. This corresponds to the non-bipartite case in graph theoretic terms, and is referred to in the literature as the Stable Roommates problem.

While Stable Marriage and Stable Roommates seem very similar, there is quite a big difference between them in terms of their structure and complexity. For one, any instance of Stable Marriage always contains a stable matching (albeit perhaps not perfect), even if the preference lists are incomplete and with ties. Moreover, computing some stable matching in any Stable Marriage instance with $2n$ agents can be done in $O(n^2)$ time [22]. However, an instance of Stable Roommates may have no stable matchings at all, even in the case of complete preference lists without ties (see the third example in Figure 1). Furthermore, when ties are present, deciding whether an instance of Stable Roommates contains a stable matching is NP-complete [47], even in the case of complete preference lists.

All variants of Stable Marriage and Stable Roommates mentioned here have several applications in a wide range of application domains. These include partnership issues in the real-world [22], resource allocation [5, 16, 27], centralized automated mechanisms that assign children to schools [3, 4], assigning school graduates to universities [7, 8], assigning medical students to hospitals [1, 2], and several others [6, 21, 29, 30, 33, 34, 35, 37, 38, 48, 49].

Optimization variants. As noted above, some Stable Roommates instances do not admit any stable matching at all, and in fact, empirical study suggests that a constant fraction of all sufficiently large instances will have no solution [46]. Moreover, even if a given Stable

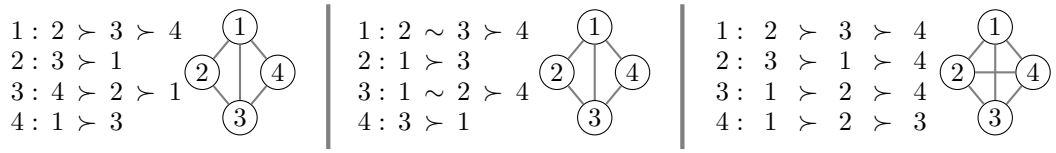


Figure 1 An example of three Stable Roommates instances, where $x \succ y$ means that x is strictly preferred to y, and $x \sim y$ means that they are equally good and tied as a partner. The instance on the left is incomplete without ties and has exactly two stable matchings $\{\{1,2\},\{3,4\}\}$ and $\{\{1,4\},\{2,3\}\}$, both of which are perfect. The instance in the middle is incomplete with ties and has two stable matchings $\{\{1,3\}\}$ and $\{\{1,2\},\{3,4\}\}$, the latter being perfect while the former not. The right instance is complete without ties and has no stable matchings at all.

Roommates instance admits a solution, this solution may not be unique, and there might be other stable matchings with which the agents are more satisfied overall. Given these two facts, it makes sense to consider two types of optimization variants for Stable Roommates: In one type, one would want to compute a stable matching that optimizes a certain social criterion in order to maximize the overall satisfaction of the agents. In the other, one would want to compute matchings which are as close as possible to being stable, where closeness can be measured by various metrics. In this paper, we focus on one prominent example of each of these two types – minimizing the egalitarian cost of a stable matching, and minimizing the number of blocking pairs in a matching which is close to being stable.

Egalitarian optimal stable matchings. Over the years, several social optimality criteria have been considered, yet arguably one of the most popular of these is the egalitarian cost metric [41, 32, 31, 36, 39]. The *egalitarian cost* of a given matching is the sum of the *rank*s of the partners of all agents, where the *rank* of the partner y of an agent x is the number of agents that are strictly preferred over y by x. The corresponding Egal Stable Marriage and Egal Stable Roommates problems ask whether there is a stable matching with egalitarian cost at most γ, for some given bound $\gamma \in \mathbb{N}$ (Section 2 contains the formal definition).

When the input preferences do not have ties (but could be incomplete), Egal Stable Marriage is solvable in $O(n^4)$ time [31]. For preferences with ties, Egal Stable Marriage becomes NP-hard [36]. Thus, already in the bipartite case, it becomes apparent that allowing ties in preference lists makes the task of computing an optimal egalitarian matching much more challenging. Marx and Schlotter [39] showed that Egal Stable Marriage is fixed-parameter tractable when parameterized by the parameter "sum of the lengths of all ties".

For Egal Stable Roommates, Feder [20] showed that the problem is NP-hard even if the preferences are complete and have no ties, and gave a 2-approximation algorithm for this case. Halldórsson et al. [25] showed inapproximability results for Egal Stable Roommates, and Teo and Sethuraman [51] proposed a specific LP formulation for Egal Stable Roommates and other variants. Cseh et al. [17] studied Egal Stable Roommates for preferences with bounded length ℓ and without ties. They showed that the problem is polynomial-time solvable if $\ell = 2$, and is NP-hard for $\ell \geq 3$.

Matchings with minimum number of blocking pairs. For the case where no stable matchings exist, the agents may still be satisfied with a matching that is close to being stable. One very natural way to measure how close a matching is to being stable is to count the number of blocking pairs [45, 19]. Accordingly, the Min-Block-Pair Stable Roommates problem asks to find a matching with a minimum number of blocking pairs.

Abraham et al. [6] showed that Min-Block-Pair Stable Roommates is NP-hard, and cannot be approximated within a factor of $n^{0.5-\varepsilon}$ unless P = NP, even if the given preferences

are complete. They also showed that the problem can be solved in $n^{O(\beta)}$ time, where n and β denote the number of agents and the number of blocking pairs, respectively. This implies that the problem is in the XP class (for parameter β) of parameterized complexity. Biró et al. [9] showed that the problem is NP-hard and APX-hard even if each agent has a preference list of length at most 3, and presented a $(2\ell - 3)$-approximation algorithm for bounded list length ℓ. Biró et al. [10] and Hamada et al. [26] showed that the related variant of Stable Marriage, where the goal is to find a matching with minimum blocking pairs among all maximum-cardinality matchings, cannot be approximated within $n^{1-\varepsilon}$ unless P = NP.

Our contributions. We analyze both Egal Stable Roommates and Min-Block-Pair Stable Roommates from the perspective of parameterized complexity, under the natural parameterization of each problem (*i.e.* the egalitarian cost and number of blocking pairs, respectively). We show that while the former is fixed-parameter tractable, the latter is W[1]-hard even when each preference list has length at most five and has no ties. This shows a sharp contrast between the two problems: Computing an optimal egalitarian stable matching is a much easier task than computing a matching with minimum blocking pairs.

When no ties are present, an instance of the Egal Stable Roommates problem has a lot of structure, and so we can apply a simple branching strategy for finding a stable matching with egalitarian cost of at most γ in $2^{O(\gamma)}n^2$ time. Moreover, we derive a kernelization algorithm, obtaining a polynomial problem kernel (Theorems 3 and 4). Note that the original reduction of Feder [20] already shows that Egal Stable Roommates cannot be solved in $2^{o(\gamma)}n^{O(1)}$ time unless the Exponential Time Hypothesis [18] fails.

When ties are present, the problem becomes much more challenging because several agents may be tied as a first ranked partner and it is not clear how to match them to obtain an optimal egalitarian stable matching. Moreover, we have to handle unmatched agents. When preferences are complete or without ties, all stable matchings match the same (sub)set of agents and this subset can be found in polynomial time [24, Chapter 4.5.2]. Thus, unmatched agents do not cause any real difficulties. However, in the case of ties and with incomplete preferences, stable matchings may involve different sets of unmatched agents. Aiming at a socially optimal egalitarian stable matching, we consider the cost of an unmatched agent to be the length of its preference list [39]. (For the sake of completeness, we also consider two other variants where the cost of an unmatched agent is either zero or a constant value, and show that both these variants are unlikely to be fixed-parameter tractable.) Our first main result is given in the following theorem:

▶ **Theorem 1.** Egal Stable Roommates *can be solved in* $\gamma^{O(\gamma)} \cdot (n \log n)^3$ *time, even for incomplete preferences with ties, where* n *is the number of agents and* γ *the egalitarian cost.*

The general idea behind our algorithm is to apply random separation [13] to "separate" irrelevant pairs from the pairs that belong to the solution matching, and from some other pairs that would not block our solution. This is done in two phases, each involving some technicalities, but in total the whole separation can be computed in $\gamma^{O(\gamma)} \cdot n^{O(1)}$ time. After the separation step, the problem reduces to Minimum-Weight Perfect Matching, and we can apply known techniques. Recall that for the case where the preferences have no ties, a simple depth-bounded search tree algorithm suffices (Theorem 4).

In Section 4, we show that Min-Block-Pair Stable Roommates is W[1]-hard with respect to the parameter β (the number of the blocking pairs) even if each input preference list has length at most five and does not have ties. This implies that assuming bounded

length of the preferences does not help in designing an $f(\beta) \cdot n^{O(1)}$-time algorithm for Min-Block-Pair Stable Roommates, unless FPT = W[1]. Our W[1]-hardness result also implies as a corollary a lower-bound on the running time of any algorithm. By adapting our reduction, we also answer in the negative an open question regarding the number of blocking agents proposed by Manlove [37, Chapter 4.6.5] (Corollary 14).

▶ **Theorem 2.** *Let n denote the number of agents and β denote the number of blocking pairs. Even when each input preference list has length at most five and has no ties,* Min-Block-Pair Stable Roommates *is W[1]-hard with respect to β and admits* no $f(\beta) \cdot n^{o(\beta)}$*-time algorithms unless the Exponential Time Hypothesis is false.*

Besides the relevant work mentioned above there is a growing body of research regarding the parameterized complexity of preference-based stable matching problems [39, 40, 43, 42, 23, 15]. Due to space constraints we deferred the proofs for results marked by $\star$ to a full version [14].

2 Definitions and notation

Let $V = \{1, 2, \ldots, n\}$ be a set of even number n agents. Each agent $i \in V$ has a subset of agents $V_i \subseteq V$ which it finds *acceptable* as a partner and has a *preference list* $\succeq_i$ on V_i (*i.e.* a transitive and complete binary relation on V_i). Here, $x \succeq_i y$ means that i weakly prefers x over y (*i.e.* x is better or as good as y). We use $\succ_i$ to denote the asymmetric part (*i.e.* $x \succeq_i y$ and $\neg(y \succeq_i x)$) and $\sim_i$ to denote the symmetric part of $\succeq_i$ (*i.e.* $x \succeq_i y$ and $y \succeq_i x$). For two agents x and y, we call x *most acceptable* to y if x is a maximal element in the preference list of y. Note that an agent can have more than one most acceptable agent. We extend $\succeq$ to $X \succeq Y$ for pairs of disjoint subsets $X, Y \subseteq V$ in the natural way.

A preference profile $\mathcal{P}$ for V is a collection $(\succeq_i)_{i \in V}$ of preference lists for each agent $i \in V$. A profile $\mathcal{P}$ may have the following properties: It is *complete* if for each agent $i \in V$ it holds that $V_i \cup \{i\} = V$; otherwise it is *incomplete*. If there are three agents $i \in V$, $x, y \in V_i$ such that $x \sim_i y$, then we say that x and y are *tied* by i and that the profile $\mathcal{P}$ has *ties*. To an instance $(V, \mathcal{P})$ we assign an *acceptability graph*, which has V as its vertex set and two agents are connected by an edge if each finds the other acceptable. Without loss of generality, G does not contain isolated vertices. The *rank* of an agent i in the preference list of some agent j is the number of agents x that j strictly prefers over i: $\mathsf{rank}_j(i) \coloneqq |\{x \mid x \succ_j i\}|$.

For a preference profile with acceptability graph G and edge set $E(G)$, a *matching* $M \subseteq E(G)$ is a subset of disjoint pairs $\{x, y\}$ of agents with $x \neq y$. If $\{x, y\} \in M$, then we denote the partner y of x by $M(x)$; otherwise we call the pair $\{x, y\}$ *unmatched*. We write $M(x) = \perp$ if agent x has *no partner*; *i.e.* if agent x is not involved in any pair in M. If *no* agent x has $M(x) = \perp$ then M is *perfect*. Given a matching M of $\mathcal{P}$, an unmatched pair $\{x, y\} \in E(G) \setminus M$ *is blocking* M if both x and y prefer each other to being unmatched or to their assigned partners, *i.e.* it holds that $\big(M(x) = \perp \vee y \succ_x M(x)\big) \wedge \big(M(y) = \perp \vee x \succ_y M(y)\big)$. We call a matching M *stable* if no unmatched pair is blocking M. The Stable Roommates problem has as input a preference profile $\mathcal{P}$ for a set V of (even number) n agents and asks whether $\mathcal{P}$ admits a stable matching. When preferences are complete, each stable matching is perfect.

The two problems we consider in the paper are Egal Stable Roommates and Min-Block-Pair Stable Roommates. The latter asks to determine whether a given preference profile $\mathcal{P}$ for a set of agents V has a stable matching with at most β blocking pairs. The former problem asks to find a stable matching with minimum egalitarian cost; the egalitarian cost of a given matching M is as follows: $\gamma(M) \coloneqq \sum_{i \in V} \mathsf{rank}_i(M(i))$, where we augment the definition rank with $\mathsf{rank}_i(\perp) \coloneqq |V_i|$. For example, the second profile in Figure 1 has two stable matchings $M_1 = \{\{1, 3\}\}$ and $M_2 = \{\{1, 2\}, \{3, 4\}\}$ with $\gamma(M_1) = 4$ and $\gamma(M_2) = 2$.

Algorithm 1: A modified version of the phase-1 algorithm of Irving [28].

```
repeat
    foreach agent u ∈ U whose preference list contains at least one unmarked agent do
        w ← the first agent in the preference list of u such that {u, w} is not yet marked
        foreach u′ with u ≻_w u′ do mark {u′, w}
until no new pair was marked in the last iteration
```

The egalitarian cost, as originally introduced for the Stable Marriage problem, does not include the cost of an unmatched agent because the preference lists are complete. For complete preferences, a stable matching must assign a partner to each agent, meaning that our notion of egalitarian cost equals the one used in the literature. For preferences without ties, all stable matchings match the same subset of agents [24, Chapter 4.5.2]. Thus, the two concepts differ only by a fixed value which can be pre-determined in polynomial time [24, Chapter 4.5.2]. For incomplete preferences with ties, there seems to be no consensus on whether to "penalize" stable matchings by the cost of unmatched agents [17]. Our concept of egalitarian cost complies with Marx and Schlotter [39], but we tackle other concepts as well (Section 3.3).

3 Minimizing the egalitarian cost

In this section we give our algorithmic and hardness results for Egal Stable Roommates. Section 3.1 treats the case when no ties are present, where we can use a straightforward branching strategy. In Section 3.2 we solve the case where ties are present. Herein, we need a more sophisticated approach based on random separation. Finally, in Section 3.3, we study variants of the egalitarian cost, differing in the cost assigned to unmatched agents.

3.1 Warm-up: Preferences without ties

By the stability concept, if the preferences have no ties and two agents x and y that are each other's most acceptable agents, then any stable matching must contain $\{x, y\}$, which has cost zero. Hence, we can safely add such pairs to a solution matching. After we have matched all pairs of agents with zero cost, all remaining, unmatched agents induce cost at least one when they are matched. This leads to a simple depth-bounded branching algorithm. In terms of kernelization, we can delete any two agents that induce zero cost and delete agents from some preference list that are ranked higher than γ. This gives us a polynomial kernel.

First, we recall a part of the polynomial-time algorithm by Irving [28] which finds an *arbitrary* stable matching for preferences without ties. The whole algorithm works in two phases. We present here a modified version of the first phase to determine "relevant" agents by sorting out *fixed* pairs – pairs of agents that occur in every stable matching [24, Chapter 4.4.2] – and *marked pairs* – pairs of agents that cannot occur in any stable matching. The modified phase-1 algorithm is given in Algorithm 1. Herein, by *marking a pair* $\{u, w\}$ we mean marking the agents u and w in the preference lists of w and u, respectively.

Let $\mathcal{P}_0$ be the preference profile produced by Algorithm 1. We introduce some more notions. For each agent x, let $\mathsf{first}(\mathcal{P}_0, x)$ and $\mathsf{last}(\mathcal{P}_0, x)$ denote the first and the last agent in the preference list of x that are not marked, respectively. We call a pair $\{x, y\}$ a *fixed pair* if $\mathsf{first}(\mathcal{P}_0, x) = y$ and $\mathsf{first}(\mathcal{P}_0, y) = x$. Let $\mathsf{marked}(\mathcal{P}_0)$ denote the set of all agents whose preference lists consist of only marked agents, and let $\mathsf{unmarked}(\mathcal{P}_0)$ denote the set of all agents whose preference lists have at least one unmarked agent. By [24, Chapters 4.4.2 and

4.5.2], we can neglect all agents that are in the fixed pairs and ignore all "irrelevant" agents from $\mathsf{marked}(\mathcal{P}_0)$. We can now shrink our instance to obtain a polynomial size problem kernel.

▶ **Theorem 3.** Egal Stable Roommates *for preferences without ties but with possibly incomplete preferences admits a size-$O(\gamma^2)$ problem kernel with at most $3\gamma+1$ agents and at most $\gamma+1$ agents in each of the preference lists.*

Proof sketch. Let $I=(\mathcal{P},V,\gamma)$ be an instance of Egal Stable Roommates and let $\mathcal{P}_0$ be the profile that Algorithm 1 produces for $\mathcal{P}$. We use F to denote the set of agents of all fixed pairs, and we use O to denote the set of ordered pairs (x,y) of agents such that x ranks y higher than γ. Briefly put, our kernelization algorithm will delete all agents in $F\cup\mathsf{marked}(\mathcal{P}_0)$, and introduce $O(\gamma)$ dummy agents to replace the deleted agents and some more that are identified by O. Initially, F and O are set to empty sets.

1. If $|\mathsf{marked}(\mathcal{P}_0)|>\gamma$ or if there is an agent x in $\mathsf{unmarked}(\mathcal{P}_0)$ with $\mathsf{rank}_x(\mathsf{first}(\mathcal{P}_0,x))>\gamma$, then return a trivial no-instance.
2. For each two agents $x,y\in\mathsf{unmarked}(\mathcal{P}_0)$ with $\mathsf{first}(\mathcal{P}_0,x)=y$ and $\mathsf{first}(\mathcal{P}_0,y)=x$, add to F the agents x and y. Let $\hat{\gamma}=\gamma-\sum_{x\in F}\mathsf{rank}_x(\mathsf{first}(\mathcal{P}_0,x))-\sum_{x\in\mathsf{marked}(\mathcal{P}_0)}|V_x|$.
3. If $2\hat{\gamma}<|\mathsf{unmarked}(\mathcal{P}_0)\setminus F|$, then return a trivial no-instance.
4. Add to the original agent set a set D of $2k$ *dummy agents* $d_1,d_2,\ldots,d_{2k}$, where $k=2\lceil\hat{\gamma}/2\rceil$, such that for each $i\in\{1,2,\ldots,k\}$, the preference list of d_i consists of only d_{k+i}, and the preference list of d_{k+i} consists of only d_i.
5. For each two $x,y\in\mathsf{unmarked}(\mathcal{P}_0)$ with $\mathsf{rank}_x(y)>\hat{\gamma}$, add to O the ordered pair (x,y).
6. For each agent $a\in\mathsf{unmarked}(\mathcal{P}_0)\setminus F$ do the following.
   (1) For each $i\in\{0,1,2,\ldots,\hat{\gamma}\}$, let x be the agent with $\mathsf{rank}_a(x)=i$. If $x\in F\cup\mathsf{marked}(\mathcal{P}_0)$ or if $(x,a)\in O$, then replace in a's preference list agent x with a dummy agent d, using a different dummy for each i, and append a to the preference list of d.
   (2) Delete all agents y in the preference list of a with $\mathsf{rank}_a(y)>\hat{\gamma}$.
7. Delete $F\cup\mathsf{marked}(\mathcal{P}_0)$ from $\mathcal{P}_0$.

The proof that the above algorithm produces a problem kernel with the desired size in the desired running time is deferred to a full version [14]. ◀

Using a simple branching algorithm, we obtain the following.

▶ **Theorem 4** ($\star$). *Let n denote the number of agents and γ denote the egalitarian cost.* Egal Stable Roommates *without ties can be solved in $O(2^\gamma\cdot n^2)$ time.*

3.2 Preferences with ties

When the preferences may contain ties, we can no longer assume that if two agents are each other's most acceptable agents, denoted as a *good pair*, then a minimum egalitarian cost stable matching would match them together; note that good pairs do not induce any egalitarian cost. This is because their match could force other pairs to be matched together that have large cost. Nevertheless, a good pair will never block any other pair, *i.e.* no agent in a good pair will form with an agent in some other pair a blocking pair. However, a stable matching may still contain some other pairs which have non-zero cost. We call such pairs *costly pairs*. Aiming to find a stable matching M with egalitarian cost at most γ, it turns out that we can also identify in $\gamma^{O(\gamma)}\cdot n^{O(1)}$ time a subset S of pairs of agents, which contains all costly pairs of M and contains no two pairs that may *induce a blocking pair*. It hence suffices to find a minimum-cost maximal matching in the graph induced by S and the good pairs.

The crucial idea is to use the random separation technique [13] to highlight the difference between the matched costly pairs in M and the unmatched costly pairs. This enables us to ignore the costly pairs which pairwisely block each other or are blocked by some pair in M so as to obtain the desired subset S. Before describing the algorithm, we show that we can focus on the case of perfect matchings, even for incomplete preferences. (Note that the case with complete preferences is covered because stable matchings for such case are always perfect.) We show this by introducing dummy agents to extend each non-perfect stable matching to a perfect one, without altering the egalitarian cost.

▶ **Lemma 5** (⋆). Egal Stable Roommates *for n agents and egalitarian cost γ is $O(\gamma \cdot n^2)$-time reducible to* Egal Stable Roommates *for at most $n+\gamma$ agents and egalitarian cost γ with an additional requirement that the stable matching should be perfect.*

Lemma 5 allows, in a subprocedure of our main algorithm, to compute a min-cost *perfect* matching in polynomial time instead of a min-cost *maximal* matching (which is NP-hard).

The algorithm. As mentioned, we use random separation [13]. We apply it already in derandomized form using Bshouty's construction of cover-free families [11], a notion related to universal sets [44]. Let $\hat{n}, p, q \in \mathbb{N}$ such that $p + q \leq \hat{n}$. A family $\mathcal{F}$ of subsets of some $\hat{n}$-element universe U is called $(\hat{n}, p, q)$-*cover-free family* if for each subset $S \subseteq U$ of cardinality $p+q$ and each subset $S' \subseteq S$ of cardinality p, there is a member $A \in \mathcal{F}$ with $S \cap A = S'$.[1] The result by Bshouty [11, Theorem 4] implies that if $p \in o(q)$, then there is an $(\hat{n}, p, q)$-cover-free family of cardinality $q^{O(p)} \cdot \log \hat{n}$ which can be computed in time linear of this cardinality.

In the remainder of this section, we prove Theorem 1. Let $\mathcal{P}$ be a preference profile for a set V of agents, possibly incomplete and with ties. For brevity we denote by a *solution* (of $\mathcal{P}$) a stable matching M with egalitarian cost at most γ. By Lemma 5, we assume that each solution is perfect. Our goal is to construct a graph with vertex set V which contains all matched "edges", representing the pairs, of some solution and some other edges for which no two edges in this graph are blocking each other. Herein, we say that two edges $e, e' \in \binom{V}{2}$ are *blocking each other* if, assuming both edges (which are two disjoint pairs of agents) are in the matching, they would *induce a blocking pair*, *i.e.* $u' \succ_u v$ and $u \succ_{u'} v'$, where $e \coloneqq \{u, v\}$ and $e' \coloneqq \{u', v'\}$. Pricing the edges with their corresponding cost, by Lemma 5, it is then enough to find a minimum-cost perfect matching. The graph is constructed in three phases (see Algorithm 2). In the first phase, we start with the acceptability graph of our profile $\mathcal{P}$ and remove all edges whose "costs" each exceed γ. In the second and the third phases, we remove all edges that block each other while keeping a stable matching with minimum egalitarian cost intact.

We introduce some more necessary concepts. Let G be the acceptability graph corresponding to $\mathcal{P}$ with vertex set V, which also denotes the agent set, and with edge set E. The *cost* of an edge $\{x, y\}$ is the sum of the ranks of each endpoint in the preference list of the other: $\mathsf{rank}_x(y) + \mathsf{rank}_y(x)$. We call an edge $e \coloneqq \{x, y\}$ a *zero edge* if it has *cost* zero, *i.e.* $\mathsf{rank}_x(y) + \mathsf{rank}_y(x) = 0$, otherwise it is a *costly edge* if the cost does not exceed γ. We ignore all edges with cost exceeding γ. Note that no such edge belongs to or is blocking any stable matching with egalitarian cost at most γ. To distinguish between zero edges and costly edges, we construct two subsets E^{zero} and E^{exp} such that E^{zero} consists of all zero edges, *i.e.* $E^{\mathsf{zero}} \coloneqq \{\{x, y\} \in E \mid \mathsf{rank}_x(y) + \mathsf{rank}_y(x) = 0\}$, and E^{exp} consists of all costly edges, *i.e.* $E^{\mathsf{exp}} \coloneqq \{\{x, y\} \in E \mid 0 < \mathsf{rank}_x(y) + \mathsf{rank}_y(x) \leq \gamma\}$.

[1] The standard definition of cover-free families [11] is stated differently from but equivalent [12] to ours.

Algorithm 2: Constructing a perfect stable matching of egalitarian cost at most γ.

Input: A set V of agents, a preference profile $\mathcal{P}$ over V, and a budget $\gamma \in \mathbb{N}$.
Output: A stable matching of egalitarian cost at most γ if it exists.

/* Phase 1 */
1 $(V, E) \leftarrow$ The acceptability graph of $\mathcal{P}$
2 $E^{\mathsf{zero}} \leftarrow \{\{x, y\} \in E \mid \mathsf{rank}_x(y) + \mathsf{rank}_y(x) = 0\}$ // The set of zero edges in E
3 $E^{\mathsf{exp}} \leftarrow \{\{x, y\} \in E \mid 1 \leq \mathsf{rank}_x(y) + \mathsf{rank}_y(x) \leq \gamma\}$ // The set of costly edges in E
4 $E_1 \leftarrow E^{\mathsf{zero}} \cup E^{\mathsf{exp}}$
/* Phase 2 */
5 $\mathcal{F}^{\mathsf{exp}} \leftarrow (|E^{\mathsf{exp}}|, \gamma, \gamma^3)$-cover-free family over the universe E^{exp}
6 **foreach** $E' \in \mathcal{F}^{\mathsf{exp}}$ **do**
7 Apply Rule 1 to E_1 to obtain E_2
/* Phase 3 */
8 $\mathcal{C} \leftarrow (|V|, \gamma^2 + 2 \cdot \gamma, 2 \cdot \gamma)$-cover-free family over the universe V
9 **foreach** $V' \in \mathcal{C}$ **do**
10 Apply Rules 2 and 3 to E_2 to obtain E_3
11 $M \leftarrow$ Minimum-cost perfect matching in the graph (V, E_3) or $\perp$ if none exists
12 **if** $M \neq \perp$ and M has cost at most γ **then return** M

Phase 1. We construct a graph $G_1 = (V, E_1)$ from G with vertex set V and with edge set $E_1 := E^{\mathsf{zero}} \cup E^{\mathsf{exp}}$. The following is easy to see.

▶ **Lemma 6.** *If $\mathcal{P}$ has a stable matching M with egalitarian cost at most γ, then $M \subseteq E_1$.*

Observe also that a zero edge cannot block any other edge because the agents in a zero edge already obtain their most acceptable agents. Thus, we have the following.

▶ **Lemma 7.** *If two edges in E_1 block each other, then they are both costly edges.*

Phase 2. In this phase, comprising Lines 5–7 in Algorithm 2, we remove from G_1 some of the costly edges that block each other (by Lemma 7, no zero edges are blocking any other edge). For technical reasons, we distinguish two types of costly edges: We say that a costly edge e with $e := \{u, v\}$ is *critical for its endpoint* u if the largest possible rank of v over all linearizations of the preference list of u exceeds γ, *i.e.* $|\{x \in V_u \setminus \{v\} \mid x \succeq_u v\}| > \gamma$. Otherwise, e is *harmless for* u. If an edge is critical for at least one endpoint, then we call it *critical* and otherwise *harmless*. Observe that a critical edge could still belong to a solution. If two edges e and e' block each other due to the blocking pair $\{u, u'\}$ with $u \in e, u' \in e'$ such that e' is harmless for u', then we say that e is *harmlessly blocking* e' (at the endpoint u'). Note that blocking is symmetric while harmlessly blocking is not.

Intuitively, we want to distinguish the solution edges from all edges blocked by the solution. There is a "small" number of harmless edges blocked by the solution, so we can easily distinguish between them. For the critical edges, we do not have such a bound; we deal with the critical edges blocked by the solution in Phase 3 in some other way.

▶ **Lemma 8** ($\star$). *Let M be a stable matching with egalitarian cost at most γ. In G_1, at most γ^2 edges are harmlessly blocked by some edge in M.*

Let $M' := M \cap E^{\mathsf{exp}}$ be the set of all costly edges in some solution M and let B_M be the set of all edges harmlessly blocked by some edge in M. By the definition of costly edges and by Lemma 8, it follows that $|M'| \leq \gamma$ and $|B_M| \leq \gamma^2$. In order to identify and delete all edges in B_M we apply random separation. Compute a $(|E^{\mathsf{exp}}|, \gamma, \gamma^2)$-cover-free family $\mathcal{F}^{\mathsf{exp}}$ over the universe E^{exp}. For each member of $\mathcal{F}^{\mathsf{exp}}$, perform all the computations below (in this phase and in Phase 3). By the properties of cover-free families, $\mathcal{F}^{\mathsf{exp}}$ contains a *good*

member E' that "separates" M' from B_M, *i.e.* $M' \subseteq E'$ and $B_M \subseteq E^{\mathsf{exp}} \setminus E'$. Formally, we call a member $E' \in \mathcal{F}^{\mathsf{exp}}$ *good* if there is a solution M such that each costly edge in M belongs to E', and each edge that is harmlessly blocked by M belongs to $E^{\mathsf{exp}} \setminus E'$. We also call E' *good for* M. By the property of cover-free families, if there is a solution M, then $\mathcal{F}^{\mathsf{exp}}$ contains a member E' which is good for M. In the following we present two data reduction rules that delete edges and show their correctness. By *correctness* we mean that, if some member $E' \in \mathcal{F}^{\mathsf{exp}}$ is good, then the corresponding solution is still present after the edge deletion.

Recall that the goal was to compute a graph that contains all edges from a solution and some other edges such that no two edges in the graph block each other. Observe that we can ignore the edges in $E^{\mathsf{exp}} \setminus E'$, because, if E' is good, then it contains all costly edges in the corresponding solution. This implies the correctness of the first part of the following reduction rule. The correctness for the second part follows from the definition of being good.

▶ **Rule 1.** Remove all edges in $E^{\mathsf{exp}} \setminus E'$ from E_1. If there are two edges $e, e' \in E'$ that are harmlessly blocking each other, then remove both e and e' from E_1.

Let $G_2 = (V, E_2)$ be the graph obtained from G_1 by exhaustively applying Rule 1. By the goodness of E' and by the correctness of Rule 1, we have the following.

▶ **Lemma 9.** *If there is a stable matching M with egalitarian cost at most γ, then $\mathcal{F}^{exp}$ contains a member E' such that the edge set E_2 of G_2 defined for E' contains all edges of M.*

By Lemma 7 and since all pairs of edges that are harmlessly blocking each other are deleted by Rule 1, we have the following.

▶ **Lemma 10.** *If two edges in G_2 block each other due to a blocking pair $\{u, u'\}$, then one of the edges is critical for u or u'.*

Phase 3. In Line 10 of Algorithm 2 we remove from G_2 the remaining (critical) edges that do not belong to M but are blocked by some other edges. This includes the edges that are blocked by M. While the number of edges blocked by M could still be unbounded, we show that there are only $O(\gamma^2)$ agents due to which an edge could be blocked by M. The idea here is to identify such agents, helping to find and delete edges blocked by M or blocking some other edges. We introduce one more notion. Consider an arbitrary matching N (*i.e.* a set of disjoint pairs of agents) of G_2. Let $e \in N$ and $e' \in E_2 \setminus N$ be two edges. If they induce a blocking pair $\{u, u'\}$ with $u \in e$ and $u' \in e'$, then we say that u' is a *culprit* of N. We obtain the following upper bound on the number of culprits with respect to a solution.

▶ **Lemma 11 ($\star$).** *Let M be a stable matching. Then, each culprit of M is incident with some edge in M. If M has egalitarian cost at most γ, then it admits at most γ culprits.*

Consider a solution M and let $\mathsf{CI}(M) = \{v \in V \mid v$ is a culprit of or incident with some costly edge of $M\}$. By Lemma 11 and since M has at most γ costly edges, it follows that $|\mathsf{CI}(M)| \leq 3\gamma$. We aim to identify in $\mathsf{CI}(M)$ a subset $\mathsf{R}(M)$ of agents incident with a critical edge in M, *i.e.* $\mathsf{R}(M) = \{v \in \mathsf{CI}(M) \mid \{v, w\} \in M$ with $\{v, w\}$ being critical for $v\}$. Since M has at most γ costly edges, it follows that $|\mathsf{R}(M)| \leq 2\gamma$. To "separate" $\mathsf{R}(M)$ from $\mathsf{CI}(M)$, we compute a $(|V|, 2\gamma, 3\gamma)$-cover-free family $\mathcal{C}$ on the set V. We call a member $V' \in \mathcal{C}$ *good* if there is a solution $M \subseteq E_2$ such that $\mathsf{R}(M) \subseteq V'$ and $(\mathsf{CI}(M) \setminus \mathsf{R}(M)) \subseteq V \setminus V'$. By a similar reasoning as given for Phase 2 and by the properties of cover-free families, if there is a solution $M \subseteq E_2$, then $\mathcal{C}$ contains a good member V'. For this member, the following two reduction rules will not destroy the solution.

▶ **Rule 2** (⋆). For each agent $y \in V \setminus V'$, delete all incident edges that are critical for y.

After having exhaustively applied Rule 2, we use the following reduction rule.

▶ **Rule 3** (⋆). If E_2 contains two edges e and e' that induce a blocking pair $\{u, u'\}$ with $u \in e$ and $u' \in e'$ such that e is critical for u, then remove e' from E_2.

Let $G_3 = (V, E_3)$ be the graph obtained after having exhaustively applied Rules 2 and 3 to G_2. By the correctness of Rules 2 and 3 we have the following.

▶ **Lemma 12.** *If there is a stable matching $M \subseteq E_2$ with egalitarian cost at most γ, then the constructed cover-free family $\mathcal{C}$ contains a good member $V' \in \mathcal{C}$ such that the edge set E_3 of G_3 resulting from the application of Rules 2 and 3 contains all edges of M.*

Since for each member $V' \in \mathcal{C}$, we delete all edges that pairwisely block each other, each perfect matching in G_3 induces a stable matching. We thus have the following.

▶ **Lemma 13** (⋆). *If G_3 admits a perfect matching M with edge cost at most γ, then M corresponds to a stable matching with egalitarian cost at most γ.*

Thus, to complete Algorithm 2, in Line 11 we compute a minimum-cost perfect matching for G_3 and output yes, if it has egalitarian cost at most γ. Summarizing, by Lemma 5 if there is a stable matching of egalitarian cost at most γ, then it is perfect and thus, by Lemmas 6, 9 and 12, there is a perfect matching in G_3 of cost at most γ. Hence, if our input is a yes-instance, then Algorithm 2 accepts by returning a desired solution. Ifi it accepts, then by Lemma 13 the input is a yes-instance. The running time is proved in a full version [14].

3.3 Variants of the egalitarian cost for unmatched agents

As discussed in Sections 1 and 2, when the input preferences are incomplete, a stable matching may leave some agents unmatched. In the absence of ties, all stable matchings leave the same set of agents unmatched [24, Chapter 4.5.2]. Hence, whether an unmatched agent should infer any cost is not relevant in terms of complexity. However, when preferences are incomplete and with ties, stable matchings may involve different sets of matched agents. The cost of unmatched agents changes the parameterized complexity dramatically. In particular, as soon as the cost of an unmatched agent is bounded by a fixed constant, seeking for an optimal egalitarian stable matching is parameterized intractable. See the full version [14].

4 Minimizing the number of blocking pairs

In this section, we strengthen the known result [6] by showing that Min-Block-Pair Stable Roommates is W[1]-hard with respect to "the number β blocking pairs", even when each preference list has length at most five. The main building block of our reduction, which is from the W[1]-hard Multi-Colored Independent Set problem (see our full version [14] for the definition), is a selector gadget (Construction 1) that always induces at least one blocking pair and allows for many different configurations. To keep the lengths of the preference lists short we use "duplicating" agents (Construction 2).

First, we discuss a vertex-selection gadget which we later use to select a vertex of the input graph into the independent set. The selected vertex is indicated by an agent which is matched to someone *outside* of the vertex-selection gadget. The gadget always induces at least one blocking pair. An illustration is shown in a full version [14]. In the following, let n' be a positive integer, and all additions and subtractions in the superscript are taken modulo $2n' + 1$:

▶ **Construction 1.** Consider the following four disjoint sets U, A, C, D of $2n'+1$ agents each, where $A \coloneqq \{a^i \mid 0 \le i \le 2n'\}$, $U \coloneqq \{u^i \mid 0 \le i \le 2n'\}$, $C \coloneqq \{c^i \mid 0 \le i \le 2n'\}$, and $D \coloneqq \{d^i \mid 0 \le i \le 2n'\}$. The preferences of the agents in $A \cup C \cup D$ are: $\forall i \in \{0, 1, \ldots, 2n'\}$: agent a^i: $a^{i+1} \succ a^{i-1} \succ u^i \succ c^i \succ d^i$, agent c^i: $d^i \succ a^i$, agent d^i: $a^i \succ c^i$.

The preferences of the agents in U are intentionally left unspecified and we define them later when we use the gadget. Regardless of the preferences of the agents in U, we can verify that if no a^i obtains an agent u^i as a partner, then it induces at least two blocking pairs.

Next, we construct verification gadgets that ensure that no two adjacent vertices are chosen into the independent set solution. See the full version for an illustration [14]. Herein, let δ be a positive integer, and all additions and subtractions in the superscript are taken modulo $2\delta + 2$.

▶ **Construction 2.** Consider two disjoint sets $X \uplus Y$ where $X = \{x^i \mid 0 \le i \le 2\delta+1\}$ is a set of $2\delta+2$ agents and $Y = \{y^i \mid 1 \le i \le \delta\}$ is a set of δ agents. Let a, b be two agents distinct from the agents in $X \cup Y$. The preference lists of the agents from X are as follows.

	Agent x^0:	$x^1 \succ a \succ x^{2\delta+1}$,	Agent $x^{2\delta+1}$:	$x^0 \succ b \succ x^{2\delta}$.
$\forall i \in \{1, \ldots, \delta\}$:	Agent x^{2i-1}:	$x^{2i} \succ x^{2i-2}$,	Agent x^{2i}:	$x^{2i+1} \succ y^i \succ x^{2i-1}$.

The preferences of the agents a, b and those in Y are intentionally left unspecified and will be defined when we use the gadget later. Regardless of the concrete preferences of agents in $Y \cup \{a, b\}$, we claim that the above gadget has two possible matchings such that no blocking pair involves any agent from X. The first one is straightforward from the definition of the preference lists: $\{\{x^{2i}, x^{2i+1}\} \mid i \in \{0, 1, \ldots, \delta\}\}$. The second one matches x^0 to a, $x^{2\delta+1}$ to b, while keeping the remaining agents matched in some stable way.

Proof sketch of Theorem 2. Let $(G = (V_1, V_2, \ldots, V_k, E))$ be a Multi-Colored Independent Set instance (see our full version [14] for the definition). Without loss of generality, assume that each vertex subset V_j has exactly $2n'+1$ vertices with the form $V_j = \{v_j^0, v_j^1, \ldots, v_j^{2n'}\}$. Construct a Min-Block-Pair Stable Roommates instance with the following groups of agents: $U_j, A_j, B_j, C_j, D_j, F_j, W_j$, $j \in \{1, 2, \ldots, k\}$, where U_j corresponds to the vertex subset V_j. Let δ_j^i be the degree of vertex v_j^i. For each vertex $v_j^i \in V_j$, construct $2\delta_j^i + 2$ agents $u_j^{i,0}, u_j^{i,1}, \ldots, u_j^{i,2\delta_j^i+1}$ and let $U_j^i = \{u_j^{i,z} \mid 0 \le z \le 2\delta_j^i + 1\}$. Define $U_j = \cup_{0 \le i \le 2n'} U_j^i$. For each $(Q, q) \in \{(A, a), (B, b), (C, c), (D, d), (F, f), (W, w)\}$ and for each $i \in \{1, 2, \ldots, k\}$, the set $Q_j \coloneqq \{q_j^i \mid 0 \le i \le 2n'\}$ consists of $2n'+1$ agents. The preference lists of the agents in U_j^i obey the verification gadget constructed in Construction 2. Formally, for each $j \in \{1, \ldots, k\}$ and each $i \in \{0, 1, \ldots, 2n'\}$ we introduce a *verification gadget for* v_j^i as in Construction 2 where we set $\delta = \delta_j^i$, $x^z = u_j^{i,z}$, $0 \le z \le 2\delta_j^i + 1$, $a = a_j^i$, and $b = b_j^i$. The agents from Y correspond to the neighbors of v_j^i: For each neighbor $v_{j'}^{i'}$ of v_j^i we pick a not-yet-set agent y^z in the verification gadget for v_j^i and a not-yet-set agent $y^{z'}$ in the verification gadget for $v_{j'}^{i'}$, and define $y^z = u_{j'}^{i',2z'}$ and $y^{z'} = u_j^{i,2z}$.

For each $j \in \{1, \ldots, k\}$, the preference lists of $A_j \cup C_j \cup D_j \cup \{u_j^{i,0} \mid 0 \le i \le 2n'\}$ obey Construction 1. Formally, for each $j \in \{1, \ldots, k\}$ we introduce a vertex-selection gadget as in Construction 1 and for each $i \in \{0, 1, \ldots, 2n'\}$ we set $a^i = a_j^i$, $c^i = c_j^i$, $d^i = d_j^i$, and $u^i = u_j^{i,0}$. Analogously, for each $j \in \{1, \ldots, k\}$ we introduce a vertex-selection gadget for $B_j \cup F_j \cup W_j \cup \{u_j^{i,2\delta_j^i+1} \mid 0 \le i \le 2n'\}$: For each $i \in \{0, 1, \ldots, 2n'\}$ we set $a^i = b_j^i$, $c^i = f_j^i$, $d^i = w_j^i$, and $u^i = u_j^{i,2\delta_i^j+1}$. To complete the construction, we set the upper bound on the number of blocking pairs as $\beta = 2k$. The correctness proof is deferred to a full version [14]. ◀

The reduction given in the proof of Theorem 2 shows that the lower-bound on β in [6, Lemma 4] is tight. The reduction also answers an open question by Manlove [37, Chapter 4.6.5] about the complexity of the following problem. Given a preference profile and an integer η, Min-Block-Agents Stable Roommates asks whether there is a matching with at most η *blocking agents*. Herein, an agent is a *blocking agent* if it is involved in a blocking pair.

▶ **Corollary 14** ($\star$). *Let n be the number of agents and η be the number of blocking agents. Even when each input preference list has length at most five and has no ties,* Min-Block-Agents Stable Roommates *is NP-hard and W[1]-hard with respect to η.* Min-Block-Agents Stable Roommates *for preferences without ties is solvable in $O(2^{\eta^2} \cdot n^{\eta+2})$ time.*

References

1 National resident matching program website. URL: `http://www.nrmp.org`.
2 Scottish foundation allocation scheme website. URL: `http://www.nes.scot.nhs.uk/sfas`.
3 Atila Abdulkadiroğlu, Parag A. Pathak, and Alvin E. Roth. The Boston public school match. *American Economic Review*, 95(2):368—371, 2005.
4 Atila Abdulkadiroğlu, Parag A. Pathak, and Alvin E. Roth. The New York City high school match. *American Economic Review*, 95(2):364–367, 2005.
5 David Abraham, Ning Chen, Vijay Kumar, and Vahab S. Mirrokni. Assignment problems in rental markets. In *Proceedings of the Second International Workshop on Internet and Network Economics (WINE '06)*, pages 198–213, 2006.
6 David J. Abraham, Péter Biró, and David Manlove. "Almost stable" matchings in the roommates problem. In *Proceedings of the Third International Workshop on Approximation and Online Algorithms (WAOA '05)*, pages 1–14, 2005. `doi:10.1007/11671411_1`.
7 Mourad Baïou and Michel Balinski. Student admissions and faculty recruitment. *Theoretical Computer Science*, 322(2):245–265, 2004.
8 Péter Biró and Sofya Kiselgof. College admissions with stable score-limits. *Central European Journal of Operations Research*, 23(4):727–741, 2015. `doi:10.1007/s10100-013-0320-9`.
9 Péter Biró, David Manlove, and Eric McDermid. "Almost stable" matchings in the Roommates problem with bounded preference lists. *Theoretical Computer Science*, 432:10–20, 2012.
10 Péter Biró, David Manlove, and Shubham Mittal. Size versus stability in the marriage problem. *Theoretical Computer Science*, 411(16-18):1828–1841, 2010.
11 Nader H. Bshouty. Linear time Constructions of some d-Restriction Problems. In *Proceedings of the 9th International Conference on Algorithms and Complexity (CIAC '15)*, volume 9079 of *Lecture Notes in Computer Science*, pages 74–88. Springer, 2015.
12 Nader H. Bshouty and Ariel Gabizon. Almost Optimal Cover-Free Families. In *Proceedings of the 11th International Conference on Algorithms and Complexity (CIAC '17)*, volume 10236 of *Lecture Notes in Computer Science*, pages 140–151. Springer, 2017.
13 Leizhen Cai, Siu Man Chan, and Siu On Chan. Random Separation: A New Method for Solving Fixed-Cardinality Optimization Problems. In *Proceedings of the Second International Workshop on Parameterized and Exact Computation (IWPEC '06)*, pages 239–250. Springer, 2006.
14 Jiehua Chen, Danny Hermelin, Manuel Sorge, and Harel Yedidsion. How hard is it to satisfy (almost) all roommates? Technical report, arXiv:1707.04316 [cs.CC], 2018.
15 Jiehua Chen, Rolf Niedermeier, and Piotr Skowron. Stable marriage with multi-modal preferences. Technical report, arXiv:1801.02693 [cs.MA,cs.DS], 2018.
16 Yan. Chen and Tayfun Sönmez. Improving efficiency of on-campus housing: An experimental study. *American Economic Review*, 92(5):1669—-1686, 2002.

17 Ágnes Cseh, Robert W. Irving, and David F. Manlove. The stable roommates problem with short lists. In *Proceedings of the 9th International Symposium on Algorithmic Game Theory (SAGT '16)*, pages 207–219, 2016.

18 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

19 Kimmo Eriksson and Olle Häggström. Instability of matchings in decentralized markets with various preference structures. *International Journal of Game Theory*, 36(3):409–420, 2008.

20 Tomás Feder. A new fixed point approach for stable networks and stable marriages. *Journal of Computer and System Sciences*, 45(2):233–284, 1992.

21 Anh-Tuan Gai, Dmitry Lebedev, Fabien Mathieu, Fabien de Montgolfier, Julien Reynier, and Laurent Viennot. Acyclic preference systems in P2P networks. In *Proceedings of the 13th International Euro-Par Conference*, pages 825–834, 2007.

22 D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 120(5):386–391, 2013. doi:10.4169/amer.math.monthly.120.05.386.

23 Sushmita Gupta, Sanjukta Roy, Saket Saurabh, and Meirav Zehavi. Balanced stable marriage: How close is close enough? Technical report, arXiv:1707.09545 [cs.DS], 2017.

24 Dan Gusfield and Robert W. Irving. *The Stable marriage problem–Structure and algorithms*. Foundations of computing series. MIT Press, 1989.

25 Magnús M. Halldórsson, Robert W. Irving, Kazuo Iwama, David Manlove, Shuichi Miyazaki, Yasufumi Morita, and Sandy Scott. Approximability results for stable marriage problems with ties. *Theoretical Computer Science*, 306(1-3):431–447, 2003. doi:10.1016/S0304-3975(03)00321-9.

26 Koki Hamada, Kazuo Iwama, and Shuichi Miyazaki. An improved approximation lower bound for finding almost stable maximum matchings. *Information Processing Letters*, 109(18):1036–1040, 2009.

27 Aanund Hylland and Richard Zeckhauser. The efficient allocation of individuals to positions. *Journal of Political Economy*, 87(2):293–314, 1979.

28 Robert W. Irving. An efficient algorithm for the 'stable roommates' problem. *Journal of Algorithms*, 6(4):577–595, 1985.

29 Robert W. Irving. Optimal stable marriage. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 1470–1473. Springer, 2016. doi:10.1007/978-1-4939-2864-4_271.

30 Robert W. Irving. Stable marriage. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 2060–2064. Springer, 2016. doi:10.1007/978-1-4939-2864-4_393.

31 Robert W. Irving, Paul Leather, and Dan Gusfield. An efficient algorithm for the 'optimal' stable marriage. *Journal of the ACM*, 34(3):532–543, 1987. doi:10.1145/28869.28871.

32 Donald Knuth. *Mariages Stables*. Les Presses de L'Université de Montréal, 1976.

33 Eija Kujansuu, Tuukka Lindberg, and Erkki Mäkinen. The Stable Roommates problem and chess tournament pairings. *Divulgaciones Matemáticas*, 7(1):19–28, 1999.

34 Dmitry Lebedev, Fabien Mathieu, Laurent Viennot, Anh-Tuan Gai, Julien Reynier, and Fabien de Montgolfier. On using matching theory to understand P2P network design. In *Proceedings of the International Network Optimization Conference (INOC 2007)*, pages 1–7, 2007.

35 David Manlove. Hospitals/residents problem. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, 2008.

36 David Manlove, Robert W. Irving, Kazuo Iwama, Shuichi Miyazaki, and Yasufumi Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276(1-2):261–279, 2002. doi:10.1016/S0304-3975(01)00206-7.

37 David F. Manlove. *Algorithmics of Matching Under Preferences*, volume 2 of *Series on Theoretical Computer Science*. WorldScientific, 2013. doi:10.1142/8591.

38 David F. Manlove and Gregg O'Malley. Paired and altruistic kidney donation in the UK: Algorithms and experimentation. *ACM Journal of Experimental Algorithmics*, 19(1), 2014. doi:10.1145/2670129.

39 Dániel Marx and Ildikó Schlotter. Parameterized complexity and local search approaches for the stable marriage problem with ties. *Algorithmica*, 58(1):170–187, 2010. doi:10.1007/s00453-009-9326-z.

40 Dániel Marx and Ildikó Schlotter. Stable assignment with couples: Parameterized complexity and local search. *Discrete Optimization*, 8(1):25–40, 2011. doi:10.1016/j.disopt.2010.07.004.

41 D. G. McVitie and L. B. Wilson. The Stable Marriage problem. *Communications of the ACM*, 14(7):486–490, 1971.

42 Kitty Meeks and Baharak Rastegari. Solving hard stable matching problems involving groups of similar agents. Technical report, arXiv:1708.04109 [cs.GT], 2017.

43 Matthias Mnich and Ildikó Schlotter. Stable marriage with covering constraints-A complete computational trichotomy. In *Proceedings of the 10th International Symposium on Algorithmic Game Theory (SAGT '17)*, pages 320–332, 2017. doi:10.1007/978-3-319-66700-3_25.

44 M. Naor, L. J. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *Proceedings of 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 182–191, 1995.

45 Muriel Niederle and Alvin E. Roth. Market culture: How norms governing exploding offers affect market performance. Working Paper 10256, National Bureau of Economic Research, February 2004.

46 Boris Pittel and Robert W. Irving. An upper bound for the solvability of a random stable roommates instance. *Random Structures and Algorithms*, 5(3):465–487, 1994.

47 Eytan Ronn. NP-complete stable matching problems. *Journal of Algorithms*, 11(2):285–304, 1990. doi:10.1016/0196-6774(90)90007-2.

48 Alvin E. Roth, Tayfun Sönmez, and M. Utku Ünver. Pairwise kidney exchange. *Journal of Economic Theory*, 125(2):151–188, 2005. doi:10.1016/j.jet.2005.04.004.

49 Alvin E. Roth, Tayfun Sönmez, and M. Utku Ünver. Efficient kidney exchange: Coincidence of wants in markets with compatibility-based preferences. *American Economic Review*, 97(3):828–851, 2007.

50 Alvin E. Roth and Marilda A. Oliveira Sotomayor. *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis*. Cambridge University Press, 1992. Part of Econometric Society Monographs.

51 Chung-Piaw Teo and Jay Sethuraman. On a cutting plane heuristic for the stable roommates problem and its applications. *European Journal of Operational Research*, 123(1):195–205, 2000.

A Quadratic Size-Hierarchy Theorem for Small-Depth Multilinear Formulas

Suryajith Chillara
Department of CSE, IIT Bombay, Mumbai, India
suryajith@cse.iitb.ac.in

Nutan Limaye
Department of CSE, IIT Bombay, Mumbai, India
nutan@cse.iitb.ac.in

Srikanth Srinivasan
Department of Mathematics, IIT Bombay, Mumbai, India
srikanth@math.iitb.ac.in

Abstract

We show explicit separations between the expressive powers of multilinear formulas of small-depth and all polynomial sizes.

Formally, for any $s = s(n) = n^{O(1)}$ and any $\delta > 0$, we construct explicit families of multilinear polynomials $P_n \in \mathbb{F}[x_1, \ldots, x_n]$ that have multilinear formulas of size s and depth three but no multilinear formulas of size $s^{1/2-\delta}$ and depth $o(\log n / \log\log n)$.

As far as we know, this is the first such result for an algebraic model of computation.

Our proof can be viewed as a derandomization of a lower bound technique of Raz (JACM 2009) using ε-biased spaces.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases Algebraic circuit complexity, Multilinear formulas, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.36

1 Introduction

The main aim of Computational Complexity is to understand as precisely as possible the amounts of computational resources required to perform interesting computational tasks. These resources could be of various kinds depending on the computational model under consideration, e.g., time and space for traditional algorithms, size and depth for Boolean and Algebraic circuits, the number of random bits for randomized algorithms, total communication for communication protocols and so on.

A fundamental question regarding any given resource is if access to more of that resource strictly increases the power of the underlying computational model. Classical theorems in Computational Complexity theory such as the *Time Hierarchy theorem* [13] and *Space Hierarchy Theorem* [18] answer this question (in the affirmative) for the resources of time and space on multitape Turing Machines.

In this paper, we consider an analogous question for *Algebraic formulas*. Algebraic formulas (and their variants such as Algebraic circuits, Algebraic Branching Programs etc.) are the natural computational model for computing multivariate polynomials over some underlying domain, usually a field $\mathbb{F}$. Many natural problems, such as the Determinant, Permanent, Matrix Multiplication, the Fast Fourier Transform etc. fit into this general framework. Algebraic formulas compute multivariate polynomials from the ring $\mathbb{F}[x_1, \ldots, x_n]$ using the natural algebraic operations of sum and product.


The *size* of an algebraic formula is the number of algebraic operations it uses and is a measure of the efficiency of the formula (it roughly corresponds to the time in the case of traditional algorithms). One can also consider the *depth* of the formula, which measures how nested the algebraic operations in the formula are, and corresponds to how parallelizable the underlying procedure is. In this paper, we consider the question of proving a *Size-hierarchy theorem* for Algebraic formulas. Somewhat informally, we ask the following.

▶ **Question 1** (The Size-Hierarchy Question). *For any $\delta > 0$, are there explicit families of polynomials $P_n \in \mathbb{F}[x_1, \ldots, x_n]$ that can be computed by formulas of size $s(n)$ but not by formulas of size less than $s(n)^{1-\delta}$?*

(See Section 2 for the definition of "explicit". We note that requiring explicit polynomial families is necessary since counting arguments easily yield the existence of polynomials that have formulas of size $s(n)$ but not size $s(n)^{1-\delta}$ for most reasonable functions $s(n)$. However, as is standard in Circuit Complexity, the interesting question is finding an explicit function that witnesses this separation.)

As of now, the size-hierarchy question is far beyond the range of our techniques for most non-trivial parameters. Indeed, we do not have techniques to prove *any* explicit strong lower bounds for general algebraic formulas, let alone lower bounds for explicit polynomials that further have algebraic formulas of some prescribed size $s(n)$.

So, we restrict ourselves to the setting of *multilinear formulas*, which are algebraic formulas that are required to compute a multilinear polynomial[1] at each intermediate stage of computation. Note that the most efficient formula for computing some multilinear polynomial need not be multilinear (this is known to be true for small-depth multilinear formulas [8]) and so this is indeed a restriction. Nevertheless, it is a reasonable restriction for formulas that compute multilinear polynomials and has been the focus of a large body of work [20, 22, 21, 24, 23, 25, 15, 11, 17, 8, 3, 7] with interesting upper as well as lower bound results.

Therefore, it is natural to consider the size-hierarchy question in the setting of multilinear formulas. It follows from the work of Raz [21] and Raz and Yehudayoff [24] that for $s(n) \leq n^{O(1)}$, there are explicit polynomial families that can be computed by multilinear formulas of size s but not by multilinear formulas of size less than s^{δ_0} for some positive, but small, δ_0. (One needs to mine the proofs for the exact value of δ_0. The best value that we could obtain for δ_0 was less than $1/30$.)

In this paper, we prove a near-tight multilinear size-hierarchy theorem for *small-depth* multilinear formulas. It is known [6, 5] that any multilinear formula of polynomial size s can be converted to another of size at most $s^{1+\delta}$ and depth $O(\log n)$ (for any fixed $\delta > 0$). Below, we consider multilinear formulas of smaller depth $O(\log n / \log\log n)$. The main result is the following.

▶ **Theorem 2.** *For any fixed $c \in \mathbb{N}$ and $\delta \in (0, 1/2)$, there exists an explicit polynomial family $P_n \in \mathbb{F}[x_1, \ldots, x_n]$ that has a multilinear formula of depth 3 and size at most $s = O(n^c)$ but no multilinear formulas of size less than $s^{(1/2)-\delta}$ and depth $\Delta < \log n / 100 \log\log n$.*

As such, our result is incomparable with the separation implied by [21, 24] since we further assume that our formulas have small depth. However, in the setting of small-depth multilinear formulas, our result improves on the separation of [21, 24] in two ways. The first is that we obtain a separation of s versus $s^{(1/2)-\delta}$ as opposed to the s versus s^{δ_0} separation

[1] Recall that a *multilinear* polynomial $P \in \mathbb{F}[x_1, \ldots, x_n]$ is one in which each variable has degree at most 1.

obtained by [21, 24]. The second is that the polynomials P_n that we consider have *depth-three* formulas of size s. This is in contrast to the polynomials constructed in [21, 24], that have formulas of size s but also depth $\Omega(\sqrt{\log s})$, which is considerably larger.[2] Finally, our proof technique is based on a derandomization of a special case of a lower bound technique of Raz [22]. This derandomization leaves some scope for improvement: an optimal result along these lines would resolve the size-hierarchy question optimally yielding a separation between sizes s and $s^{1-\delta}$ for any $\delta > 0$. An optimal derandomization of the more general lower bound technique of Raz would yield the same result for general multilinear formulas (without the depth restriction).

1.1 Related Work

Our work is partially motivated by hierarchy theorems for *Boolean circuits*, which compute functions $f : \{0,1\}^n \to \{0,1\}$ using simple Boolean operations such as AND and OR. Superpolynomial lower bounds have been known for constant-depth Boolean circuits since the early 1980s [12, 1, 14]. However, a size hierarchy theorem in this setting was obtained relatively recently by Rossman [26], who proved that for any constant $k \in \mathbb{N}$, there are explicit functions that have depth-two Boolean circuits of size $O(n^k)$ but not of size less than $n^{k/4}$. This was then improved by Amano [4] who showed that for any fixed k and δ there are explicit functions that have depth two Boolean circuits of size $O(n^k)$ but no circuits of size less than $n^{k-\delta}$.

Our proofs build on standard techniques for proving lower bounds for multilinear formulas. While these ideas are essentially due to Raz [22], we use a high-level reformulation of this argument that appears in the survey of Shpilka and Yehudayoff [28].

1.2 Proof Outline

Our proof can be seen as a derandomization of (a special case of a) technique of Raz [22] for proving lower bounds for multilinear formulas. Here, we follow a well-known reformulation of this proof that appears in [15, 28, 11].

Say we want to show that a multilinear polynomial $P \in \mathbb{F}[X]$ does not have a small-depth multilinear formula of size s'. The proof strategy consists of two steps. The first step is a decomposition lemma that says that any multilinear polynomial P that is computed by a small-depth multilinear formula of size s' can be written as a sum of s' polynomials, each of which is of the form $f = f_1 \cdot f_2 \cdot \ldots \cdot f_t$ where the f_i's are multilinear polynomials over pairwise disjoint *non-empty* sets of variables $X_1, \ldots, X_t$ that partition the variable set X. Following [28], we call such a polynomial a *t-product polynomial.* Here, t is some growing function of the number of variables n and the depth of the formula.

Thus, to show that P does not have a small-depth multilinear formula of size s', it suffices to show that it cannot be written as a sum of s' many t-product polynomials for a large t. This is the second step. To argue this, Raz used a rank-based argument. Specifically, we partition the variables X into any two sets[3] Y and Z and consider any polynomial $P(X)$ as a polynomial in the variables in Y, with coefficients from $\mathbb{F}[Z]$. The dimension of the space of coefficients (as vectors over the base field $\mathbb{F}$) is considered to be a measure of the complexity of P. The idea is that polynomials with small formulas will have low complexity and hence, by choosing a P of high complexity we obtain a lower bound.

[2] In fact, the polynomial families from [21, 24] do not have formulas of constant-depth and comparable size. This follows from a later lower bound result of Raz and Yehudayoff [25].

[3] Actually, the sets Y and Z also need to be of equal size. We ignore this for now for the sake of exposition.

Unfortunately, this idea by itself is not enough to prove a strong lower bound. This is because of the fact that given any partition (Y, Z), there is a small depth-2 multilinear formula $F_{(Y,Z)}$ (which is also a t-product polynomial for large t) that has maximum dimension w.r.t. this partition. To overcome this, we consider a *random* partition (Y, Z) and show that any t-product polynomial will have low-rank w.r.t. this random partition with high probability. Using a union bound, we then show that any sum of s' many t-product polynomials must be of low-rank w.r.t. some partition. If, on the other hand, our choice of polynomial P has high rank w.r.t. *every* partition, we obtain a lower bound.

The crux of the matter therefore is to argue that given any t-product polynomial $f = f_1 \cdots f_t$ as above, and a random partition (Y, Z), the polynomial f is low-rank w.h.p., w.r.t. this partition. Formally, for the union bound over s' many such polynomials to go through, we need the following criterion to hold.

$$\Pr_{(Y,Z)}[f \text{ does not have small rank w.r.t. } (Y, Z)] < \frac{1}{s'}.$$

Raz [22] showed that this reduces to a combinatorial discrepancy question. Note that any choice of partition (Y, Z) of X induces a partition (Y_i, Z_i) of each X_i ($i \in [t]$). To prove the above bound, it actually suffices to show[4] that

$$\Pr_{(Y,Z)}[|\{i \in [t] \mid |Y_i| - \lfloor |X_i|/2 \rfloor \text{ odd}\}| \text{ is small}] < \frac{1}{s'}.$$

But this is quite easy to argue. Since (Y, Z) is a random partition, each $|Y_i| - \lfloor |X_i|/2 \rfloor$ is odd with probability $1/2$. Since the X_i ($i \in [t]$) are pairwise disjoint, these events are mutually independent and hence by a Chernoff bound, it is easy to show that the probability of the above event is $1/2^{\Omega(t)} < 1/s'$ for the specific t that we obtain in the decomposition lemma (this is where the small-depth assumption comes in). This completes the proof.

The derandomization. Our idea is to simulate the purely random partition argument of Raz, but using instead a random partition from a small predefined set $\mathcal{S} = \{(Y^{(1)}, Z^{(1)}), \ldots, (Y^{(s)}, Z^{(s)})\}$ of partitions. We would like to show that for a random $j \in [s]$ and any product polynomial $f = f_1 \cdots f_t$ as above, we similarly obtain

$$\Pr_j[|\{i \in [t] \mid |Y_i^{(j)}| - \lfloor |X_i|/2 \rfloor \text{ is odd}\}| \text{ is small}] < \frac{1}{s'} \tag{1}$$

where $Y_i^{(j)}$ denotes the set $X_i \cap Y^{(j)}$.

If we have a set $\mathcal{S}$ as above, we obtain a size-hierarchy theorem as follows. First, we construct a multilinear formula $F_{\mathcal{S}}$ of size roughly s that is full-rank w.r.t. each of the partitions in $\mathcal{S}$: this is done by simply taking a suitable linear combination of the formulas $F_{(Y^{(j)},Z^{(j)})}$ ($j \in [s]$) mentioned above. On the other hand, we know that given any small-depth multilinear formula F' of size s', (1) implies that F' cannot compute a polynomial that is full-rank w.r.t. all the partitions in $\mathcal{S}$, and in particular cannot compute the same polynomial as F. This proves a separation between small-depth multilinear formulas of size s and size s'. The question now is – how do we construct such a set $\mathcal{S}$ as described above while keeping s as close to s' as possible?

[4] Raz in fact shows that it suffices to bound the probability that $\sum_{i \in [t]} ||Y_i| - |X_i|/2|$ is small. Here, we only use the fact that for each i such that $|Y_i| - \lfloor |X_i|/2 \rfloor$ is odd, we must have $||Y_i| - |X_i|/2| \geq 1/2$. This harks back to an earlier result of Nisan and Wigderson [20] who use a simpler parity argument to prove a lower bound for *set-multilinear* formulas.

Our construction of the set $\mathcal{S}$ follows two steps. We first show that it suffices to construct a set that satisfies the following somewhat weaker condition.

$$\Pr_j[|\{i \in [t] \mid |Y_i^{(j)}| - \lfloor |X_i|/2 \rfloor \text{ is odd}\}| \ = 0] < \frac{1}{2s'} \tag{2}$$

We deduce (1) from (2) by adapting a combinatorial proof of the Chernoff bound that appears in a result of Impagliazzo and Kabanets [16].

Finally, we show that for (2) it suffices to use *Small-Bias* spaces, which are a standard tool in the derandomization literature [19]. Known explicit constructions of small-bias spaces [2] yield sets $\mathcal{S}$ satisfying (2) of size roughly $s = (s')^2$. This yields a separation between size s and size roughly $\sqrt{s}$ as stated in Theorem 2.

We remark that non-constructively, we can show that there exist sets $\mathcal{S}$ satisfying (2) of size roughly s'. Constructing such sets explicitly would improve our result to a near-tight size-hierarchy theorem.

2 Preliminaries

Recall that a polynomial $P \in \mathbb{F}[x_1, \ldots, x_n]$ is *multilinear* if each variable has degree at most 1 in P.

A family $\{P_n \in \mathbb{F}[x_1, \ldots, x_n] \mid n \geq 1\}$ of multilinear polynomials is said to be *explicit* if there is a deterministic algorithm that given as input n and a monomial m over the variables $x_1, \ldots, x_n$, computes in time poly(n) the coefficient of the monomial m in P_n.

2.1 Multilinear formulas

For the detailed introduction to algebraic formulas, we refer the reader to standard resources such as [28, 27]. Having said that, we do make a few remarks.

- All the gates in our formulas may have *unbounded* fan-in.
- The size of a formula refers to the number of gates (including input gates) in it, and depth of the formula refers to the number of gates on the longest path from an input gate to output gate.

An algebraic formula F computing a polynomial from $\mathbb{F}[X]$ is said to be *multilinear* if each gate in the formula computes a multilinear polynomial.

We state below a decomposition lemma for small-depth multilinear formulas.

Define a polynomial $f \in \mathbb{F}[X]$ to be a *t-product polynomial* if we can write f as $f_1 \cdots f_t$, where we can find a partition of X into pairwise disjoint non-empty sets $X_1^f, \ldots, X_t^f$ such that f_i is a multilinear polynomial from $\mathbb{F}[X_i^f]$.[5] We say that X_i^f is the set *ascribed* to f_i in the t-product polynomial f. We use Vars(f_i) (with a slight abuse of notation)[6] to denote X_i^f.

The following is easily implied by Lemma 3.8 in [28].

▶ **Lemma 3.** *Assume that $f \in \mathbb{F}[X]$ can be computed by a multilinear circuit over n variables of size at most s and depth at most Δ. Then, f is the sum of at most $s \cdot n$ many t-product polynomials for $t = \Omega(n^{1/2\Delta})$.*

[5] Note that we do not need f_i ($i \in [t]$) to depend non-trivially on all (or any) of the variables in X_i^f.
[6] Vars$(\cdot)$ is used to describe variables ascribed to gates in a circuit as well as to denote variables ascribed to polynomials.

2.2 Partial derivative matrices and relative rank

From here on, for the sake of simplicity, we will assume that n is even[7].

Let X, W be disjoint sets of variables with $X = \{x_1, \ldots, x_n\}$. Let $\mathbb{F}$ be any field. Let $\mathbb{G} = \mathbb{F}(W)$ be the field of rational functions over $\mathbb{F}$ generated by set of variables W. Let Y and Z be disjoint variable sets $\{y_1, \ldots, y_n\}$ and $\{z_1, \ldots, z_n\}$. We consider *injective* maps $\rho : X \to Y \cup Z$ which we call *partitioning functions*.

We can index partitioning functions by elements of $\{0,1\}^n$ as follows. Fix $\mathbf{a} \in \{0,1\}^n$ be any vector. Let $1_\mathbf{a} = \{i \mid \mathbf{a}(i) = 1\}$ and $0_\mathbf{a} = \{i \mid \mathbf{a}(i) = 0\}$, and thus, $|1_\mathbf{a}| + |0_\mathbf{a}| = n$. For a vector $\mathbf{a} \in \{0,1\}^n$, define the partitioning function $\rho_\mathbf{a}$ by $\rho_\mathbf{a}(x_i) = y_i$ if $i \in 1_\mathbf{a}$ and $\rho_\mathbf{a}(x_i) = z_i$ otherwise, i.e. if $i \in 0_\mathbf{a}$. Let $\mathrm{Img}(\rho_\mathbf{a})$ denote the subset of $Y \cup Z$ that $\rho_\mathbf{a}$ maps the set X to. Let $Y_\mathbf{a} = Y \cap \mathrm{Img}(\rho_\mathbf{a})$ and let $Z_\mathbf{a} = Z \cap \mathrm{Img}(\rho_\mathbf{a})$. If the vector $\mathbf{a}$ is balanced[8] then we also get that $|Y_\mathbf{a}| = |Z_\mathbf{a}| = n/2$. For a balanced vector $\mathbf{a}$, we call $\rho_\mathbf{a}$ a *balanced partition*.

Note that given any $\mathbf{a} \in \{0,1\}^n$ and any multilinear polynomial $f \in \mathbb{F}[X, W]$, the partitioning function $\rho_\mathbf{a}$ defines by substitution a *multilinear*[9] polynomial in $\mathbb{F}[Y_\mathbf{a} \cup Z_\mathbf{a} \cup W]$, which we denote $f|_{\rho_\mathbf{a}}$. We will consider $f|_{\rho_\mathbf{a}}$ as a polynomial in $\mathbb{G}[Y_\mathbf{a} \cup Z_\mathbf{a}]$.

For any disjoint sets of variables Y' and Z', let $g \in \mathbb{G}[Y' \cup Z']$ be a multilinear polynomial. Define the $2^{|Y'|} \times 2^{|Z'|}$ matrix $M_{(Y',Z')}(g)$ whose rows and columns are labelled by distinct multilinear monomials in Y' and Z' respectively and the (m_1, m_2)th entry of $M_{(Y',Z')}(g)$ is the coefficient of the monomial $m_1 \cdot m_2$ in g. We will use the rank of this matrix as a measure of the complexity of g.

We define the *relative-rank* of g w.r.t. (Y', Z'), denoted $\mathrm{relrk}_{(Y',Z')}(g)$, by

$$\mathrm{relrk}_{(Y',Z')}(g) = \frac{\mathrm{rank}(M_{(Y',Z')}(g))}{2^{(|Y'|+|Z'|)/2}}.$$

The above notion is implicit in the work of Nisan and Wigderson [20] and Raz [22].

We note the following properties of relative rank.

▶ **Proposition 4.** *Let $g, g_1, g_2 \in \mathbb{G}[Y' \cup Z']$ be multilinear polynomials.*

1. $\mathrm{relrk}_{(Y',Z')}(g) \leq 1$. *Further if $|Y'| \neq |Z'|$, then* $\mathrm{relrk}_{(Y',Z')}(g) \leq 1/\sqrt{2}$.
2. $\mathrm{relrk}_{(Y',Z')}(g_1 + g_2) \leq \mathrm{relrk}_{(Y',Z')}(g_1) + \mathrm{relrk}_{(Y',Z')}(g_2)$.
3. *If Y' is partitioned into Y'_1, Y'_2 and Z' into Z'_1, Z'_2 with $g_i \in \mathbb{G}[Y'_i \cup Z'_i]$ ($i \in [2]$), then* $\mathrm{rank}(M_{(Y',Z')}(g)) = \mathrm{rank}(M_{(Y'_1,Z'_1)}(g_1)) \cdot \mathrm{rank}(M_{(Y'_2,Z'_2)}(g_2))$. *In particular,* $\mathrm{relrk}_{(Y',Z')}(g_1 \cdot g_2) = \mathrm{relrk}_{(Y'_1,Z'_1)}(g_1) \cdot \mathrm{relrk}_{(Y'_2,Z'_2)}(g_2)$.

2.3 Explicit ε-biased spaces

The following notions are borrowed from [2]. For any $\mathbf{a}, \mathbf{b} \in \{0,1\}^n$, let $(\mathbf{a}, \mathbf{b})_2$ denote the inner product of the binary vectors $\mathbf{a}$ and $\mathbf{b}$ modulo 2, that is, $(\mathbf{a}, \mathbf{b})_2 = \sum_{i=1}^n \mathbf{a}(i) \cdot \mathbf{b}(i) \pmod 2$.

▶ **Definition 5.** Let $\mathcal{S}$ be a multiset in $\{0,1\}^n$. Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ be chosen uniformly from $\mathcal{S}$. The multiset $\mathcal{S}$ is said to be an ε-biased space if for every $\mathbf{b} = (b_1, b_2, \ldots, b_n) \in$

[7] If n is odd, everything will work as is with n replaced by $n - 1$.
[8] A vector $\mathbf{a} \in \{0,1\}^n$ is said to be balanced if $|1_\mathbf{a}| = |0_\mathbf{a}|$.
[9] The polynomial is multilinear by the injectivity of $\rho_\mathbf{a}$.

$\{0,1\}^n \setminus \{0\}^n$, the random variable $(\mathbf{x}, \mathbf{b})_2$ is ε-biased. That is, for all $\mathbf{b} \in \{0,1\}^n \setminus \{0\}^n$,

$$\left| \mathop{\mathbf{E}}_{\mathbf{x} \in \mathcal{S}} [(-1)^{(\mathbf{x},\mathbf{b})_2}] \right| \leq \varepsilon.$$

A standard probabilistic argument implies the existence of ε-biased spaces of size $O(n/\varepsilon^2)$. Explicit constructions of size $\mathrm{poly}(n/\varepsilon)$ were first presented by Naor and Naor in [19]. We use the following construction of Alon, Goldreich, Håstad and Peralta [2].

▶ **Theorem 6** ([2], Proposition 3). *There is a deterministic algorithm, which given as input $n \in \mathbb{N}$ and $\varepsilon > 0$, produces an ε-biased set $\mathcal{S}$ of size $O(n^2/\varepsilon^2)$ in time $\mathrm{poly}(|\mathcal{S}|)$.*

3 The hard polynomial and restrictions

3.1 Subspace-avoiding sets

▶ **Definition 7.** We say that a multiset $\mathcal{S} \subseteq \{0,1\}^n$ is (ε, k)-*subspace avoiding* if for any affine subspace V of $\{0,1\}^n$ (here identified with $\mathbb{F}_2^n$) with co-dimension k,

$$\Pr_{\mathbf{x} \in \mathcal{S}}[\mathbf{x} \in V] \leq \frac{1}{2^k} + \varepsilon.$$

The above definition is quite similar to the notion of *subspace evasive sets* that have been studied in the literature (see, e.g. [10]). However, there also seems to be a crucial difference between the two settings, since in [10] the interest is in evading subspaces of small *dimension* whereas we are trying to avoid subspaces of somewhat large but still relative small *co-dimension*. In particular, it is not clear to us if [10] can be used to give better constructions of subspace avoiding sets than the ones we obtain here.

The following fact is immediate from the definition above.

▶ **Fact 8.** *If $k = 10 \log \frac{1}{\varepsilon}$ and $\mathcal{S}$ is an (ε, k)-subspace avoiding set, then $\Pr_{\mathbf{x} \in \mathcal{S}}[\mathbf{x} \in V] \leq 2\varepsilon$.*

It is a standard fact [19, 9] that ε-biased spaces are in particular (ε, k)-subspace avoiding. We state this claim below. The proof is omitted for lack of space.

▶ **Claim 9.** *Any ε-biased space $\mathcal{S}$ is also an (ε, k)-subspace avoiding set.*

We will use the vectors from an (ε, k)-subspace avoiding set to define our hard polynomial. For reasons that will become apparent, it is helpful to have the vectors in the subspace avoiding set to be *balanced*, i.e. have an equal number of 0s and 1s. However, a priori, there is no reason to assume that the vectors we obtain via some construction of such a set will be balanced. In order to make them balanced, we will use the following trick.

For $i \in [n]$, let $\boldsymbol{\alpha}_i \in \{0,1\}^n$ denote the vector defined by $\boldsymbol{\alpha}_i(j) = 0$ if $j > i$ and $\boldsymbol{\alpha}_i(j) = 1$ otherwise. Let $\mathbf{0}$ denote the all zero vector, i.e. $\mathbf{0} = 0^n$. Let $\mathcal{V} = \{\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_n, \mathbf{0}\}$. Note that, for any $\mathbf{a} \in \{0,1\}^n$, there exists an $\mathbf{x} \in \mathcal{V}$ such that $\mathbf{a} \oplus \mathbf{x}$ is balanced. In particular for $\mathbf{x} \in \mathcal{V}$ chosen *at random*, the probability that $\mathbf{a} \oplus \mathbf{x}$ is balanced is at least $1/(n+1)$.

Now, let $\mathcal{S}$ be an (ε, k)-subspace avoiding set. Let $\mathcal{S} \oplus \mathbf{x}$ denote the set of all vectors obtained by shifting all the vectors in $\mathcal{S}$ by $\mathbf{x}$, i.e. $\mathcal{S} \oplus \mathbf{x} = \{\mathbf{a} \oplus \mathbf{x} \mid \mathbf{a} \in \mathcal{S}\}$. We have the following easily verifiable fact.

▶ **Fact 10.** *Let $\mathcal{S}$ be an (ε, k)-subspace avoiding set. Then for each $\mathbf{x} \in \{0,1\}^n$, $\mathcal{S} \oplus \mathbf{x}$ is also an (ε, k)-subspace avoiding set.*

For any $\mathbf{x}$, let $\mathcal{B}_{\mathbf{x}}$ be the balanced vectors in $\mathcal{S} \oplus \mathbf{x}$. From our reasoning above, we get that $\mathbf{E}_{\mathbf{x}\in\mathcal{V}}[|\mathcal{B}_{\mathbf{x}}|] \geq |\mathcal{S}|/(n+1)$. By averaging, we see that there exists an $\mathbf{x} \in \mathcal{V}$ such that $|\mathcal{B}_{\mathbf{x}}| \geq |\mathcal{S}|/(n+1)$. We will now fix such an $\mathbf{x}$. We will denote it by $\mathbf{x}_0$ and work with some $\mathcal{B} \subseteq \mathcal{B}_{\mathbf{x}_0}$ of size exactly $\lceil |\mathcal{S}|/(n+1) \rceil$. Clearly, given $\mathcal{S}$, such an $\mathbf{x}_0$ and $\mathcal{B}$ can be found in time $\text{poly}(|\mathcal{S}|, n)$ by simply computing all the sets $\mathcal{S} \oplus \mathbf{x}$ ($\mathbf{x} \in \mathcal{V}$) and counting the number of balanced vectors in them.

We have shown the following claim.

▶ **Claim 11.** *Let $\mathcal{S} \subseteq \{0,1\}^n$ be any (ε, k)-subspace avoiding set. Then, there is an $\mathbf{x}_0 \in \{0,1\}^n$ and a set $\mathcal{B}$ of balanced vectors from $\mathcal{S} \oplus \mathbf{x}_0$ such that $|\mathcal{B}| = \Theta(|\mathcal{S}|/n)$. Further, given $\mathcal{S}$, such an $\mathbf{x}_0$ and $\mathcal{B}$ can be found deterministically in time* $\text{poly}(|\mathcal{S}|, n)$.

3.2 The hard polynomial

We now define the explicit polynomial family that we will use to prove our size hierarchy theorem.

Let $n \in \mathbb{N}$ be any positive even integer and let $\varepsilon > 0$ be any positive real parameter. Let $\mathcal{S}_{n,\varepsilon}$ be the explicit ε-biased space from Theorem 6. We further fix an $\mathbf{x}_0 \in \{0,1\}^n$ and a set $\mathcal{B}$ of balanced vectors from $\mathcal{S}_{n,\varepsilon} \oplus \mathbf{x}_0$ such that $|\mathcal{B}| = \Theta(|\mathcal{S}_{n,\varepsilon}|/n)$ as guaranteed to exist by Claim 11. Note that by Theorem 6 and Claim 11, $|\mathcal{B}|$ can be computed in time $\text{poly}(n/\varepsilon)$. We denote $|\mathcal{B}|$ by τ.

Fix any $\mathbf{a} \in \mathcal{B}$. Let $1_{\mathbf{a}} = \{i_1, \ldots, i_{n/2}\}$ and $0_{\mathbf{a}} = \{j_1, \ldots, j_{n/2}\}$, where $i_1 < i_2 < \ldots < i_{n/2}$ and $j_1 < j_2 \ldots < j_{n/2}$. Define $\Gamma_{\mathbf{a}}(X) = \prod_{t=1}^{n/2}(x_{i_t} + x_{j_t})$.

We will use $\Gamma_{\mathbf{a}}(X)$ for $\mathbf{a} \in \mathcal{B}$ to define our hard polynomial. As in [22, 24] we define such a polynomial using a set W of auxiliary variables. Intuitively, the variables W help us in *tagging* a certain polynomial $\Gamma_{\mathbf{a}}(X)$ with the appropriate vector $\mathbf{a}$ from the set $\mathcal{B}$. We will now formally describe this idea.

Since $|\mathcal{B}| = \tau$, we can fix a one-one map $\mathcal{C} : \mathcal{B} \to \{0,1\}^{\log \tau}$. Let the vectors in $\mathcal{B}$ be enumerated in some arbitrary order, say $\mathbf{a}_1, \ldots, \mathbf{a}_\tau$. For an index $i \in [\tau]$, let $\mathcal{C}(\mathbf{a}_i) = (u_{i,1} \ldots, u_{i,T})$. We will denote this vector by $\mathbf{u}_i$, we will call $\mathbf{u}_i$ the *encoding* of $\mathbf{a}_i$.

Let $T = \log \tau$. Let $W = \{w_1, \ldots, w_T\}$ be a new set of auxiliary variables. For a vector $\mathbf{u} \in \{0,1\}^T$, let $\phi_{\mathbf{u}}(j) = w_j$ if $u_j = 1$ and $\phi_{\mathbf{u}}(j) = 1 - w_j$ otherwise. Then let $W_{\mathbf{u}} = \prod_{j\in[T]} \phi_{\mathbf{u}}(j)$. We will say that a polynomial $W_{\mathbf{u}}$ is the *label* of the vector $\mathbf{u}$. We will say that the tagging of the polynomial $\Gamma_{\mathbf{a}}(X)$ is obtained by multiplying $\Gamma_{\mathbf{a}}(X)$ with the label of the encoding of $\mathbf{a}$, i.e. with $W_{\mathcal{C}(\mathbf{a})}$. Note that, given $\mathbf{a} \in \mathcal{B}$, the polynomial $W_{\mathcal{C}(a)} \cdot \Gamma_{\mathbf{a}}(X)$ can be computed by a depth-2 multilinear formula, which itself can be constructed in time $\text{poly}(n, \tau) = \text{poly}(n/\varepsilon)$.

We are now ready to define our hard polynomial.

$$P_{n,\varepsilon}(X, W) = \sum_{\mathbf{a}\in\mathcal{B}} W_{\mathcal{C}(\mathbf{a})} \cdot \Gamma_{\mathbf{a}}(X).$$

We have the following.

▶ **Lemma 12.** *The polynomial $P_{n,\varepsilon}(X, W)$ can be computed by a depth 3 multilinear formula $F_{n,\varepsilon}$ of size $s = O(\tau(n+2\log\tau)) = O(n^2/\varepsilon^2 + (n/\varepsilon^2)\log(n/\varepsilon))$ that can be constructed in time* $\text{poly}(s)$. *Further, there is a deterministic* $\text{poly}(s)$*-time algorithm that, given a multilinear monomial m over the variables $X \cup W$, computes the coefficient of m in $P_{n,\varepsilon}(X, W)$.*

Proof. Everything but the last statement is immediate from the preceding discussion. To prove the final statement, it suffices to note that the coefficient of m can be found in each constituent depth-2 formula in time $\text{poly}(n) = \text{poly}(s)$. Summing these coefficients yields the coefficient of m in $P_{n,\varepsilon}(X, W)$. ◀

Another property of our hard polynomial that is true by construction is the following.

▶ **Lemma 13.** *For any $n \in \mathbb{N}$ and $\varepsilon > 0$, let $P_{n,\varepsilon}(X,W) \in \mathbb{F}[X,W]$ be the polynomial defined above, which we will consider as a polynomial from $\mathbb{G}[X]$ where $\mathbb{G} = \mathbb{F}(W)$. For any $\mathbf{a} \in \mathcal{B}$, let restriction $\rho_{\mathbf{a}} : X \to Y \cup Z$ be the restriction as defined in Section 2.2. Then* $\operatorname{relrk}_{(Y_{\mathbf{a}},Z_{\mathbf{a}})}(P_{n,\varepsilon}(X,W)|_{\rho_{\mathbf{a}}}) = 1$.

Proof. Fix any $\mathbf{a} \in \mathcal{B}$ and consider the balanced partition $\rho_{\mathbf{a}} : X \to Y_{\mathbf{a}} \cup Z_{\mathbf{a}}$. We analyze the partial derivative matrix $M_{(Y_{\mathbf{a}},Z_{\mathbf{a}})}(P_{n,\varepsilon}(X,W)|_{\rho_{\mathbf{a}}})$ whose entries are polynomials over the variables in W. To show that $\operatorname{relrk}_{(Y_{\mathbf{a}},Z_{\mathbf{a}})}(P_{n,\varepsilon}(X,W)|_{\rho_{\mathbf{a}}}) = 1$, we need to show that $M_{(Y_{\mathbf{a}},Z_{\mathbf{a}})}(P_{n,\varepsilon}(X,W)|_{\rho_{\mathbf{a}}})$ is a full rank matrix over $\mathbb{G}$. Towards that, it is sufficient to show that $\det(M_{(Y_{\mathbf{a}},Z_{\mathbf{a}})}(P_{n,\varepsilon}(X,W)|_{\rho_{\mathbf{a}}}))$ is a non-zero polynomial over the variables in W. Further, it is enough to show that there is an assignment $A : W \to \{0,1\}$ to the W-variables such that $\det(M_{(Y_{\mathbf{a}},Z_{\mathbf{a}})}(P_{n,\varepsilon}(X,A(W))|_{\rho_{\mathbf{a}}}))$ evaluates to a non-zero value. This is what we will do. (A similar proof strategy is used in the proof of Claim 4.6 in [3].)

Let the vector $\mathbf{u} = \mathcal{C}(\mathbf{a})$. For all $i \in [\log \tau]$, A sets the variable w_i to 1 if $u_i = 1$ and 0 otherwise. Now, it is easy to see that $P_{n,\varepsilon}(X, A(W)) = \Gamma_{\mathbf{a}}(X)$. This also implies that

$$\det(M_{(Y_{\mathbf{a}},Z_{\mathbf{a}})}(P_{n,\varepsilon}(X,A(W))|_{\rho_{\mathbf{a}}})) = \det(M_{(Y_{\mathbf{a}},Z_{\mathbf{a}})}(\Gamma_{\mathbf{a}}(X)|_{\rho_{\mathbf{a}}}))$$

Now it is easy to check that $M_{(Y_{\mathbf{a}},Z_{\mathbf{a}})}(\Gamma_{\mathbf{a}}(X)|_{\rho_{\mathbf{a}}})$ is a permutation matrix and hence $\det(M_{(Y_{\mathbf{a}},Z_{\mathbf{a}})}(\Gamma_{\mathbf{a}}(X)|_{\rho_{\mathbf{a}}}))$ is non-zero. This implies that $\det(M_{(Y_{\mathbf{a}},Z_{\mathbf{a}})}(P_{n,\varepsilon}(X,A(W))|_{\rho_{\mathbf{a}}}))$ is non-zero as well. Thus, $\det(M_{(Y_{\mathbf{a}},Z_{\mathbf{a}})}(P_{n,\varepsilon}(X,W)|_{\rho_{\mathbf{a}}}))$ is a non-zero polynomial over the variables in W and we get that $M_{(Y_{\mathbf{a}},Z_{\mathbf{a}})}(P_{n,\varepsilon}(X,W)|_{\rho_{\mathbf{a}}})$ is a full rank matrix. ◀

4 The lower bound

In this section, we show that for f, a t-product polynomial, and $\mathbf{a}$ chosen randomly from an $(\varepsilon, 10\log(1/\varepsilon))$-subspace avoiding set, the polynomial $f|_{\rho_{\mathbf{a}}}$ has low relative-rank with high probability. We then use this to prove the main theorem.

▶ **Lemma 14.** *Let $s \in \mathbb{N}$ and $\varepsilon > 0$ be parameters such that $s \geq 1/\varepsilon$. Let $f \in \mathbb{G}[X]$ be a t-product polynomial with $t \geq (\log s)^3$. Let $\mathcal{S}_0$ be any $(\varepsilon, 10\log\frac{1}{\varepsilon})$-subspace avoiding set defined in Section 3.2. For any $\mathbf{a} \in \{0,1\}^n$, let $\rho_{\mathbf{a}}$ denote the partitioning function defined in Section 2.2. Then,*

$$\Pr_{\mathbf{a}\in\mathcal{S}_0}[\operatorname{relrk}_{(Y_{\mathbf{a}},Z_{\mathbf{a}})}(f|_{\rho_{\mathbf{a}}}) \geq \frac{1}{s}] \leq 5 \cdot \varepsilon.$$

Proof of Lemma 14. For all $k \in [t]$, let $\mathbf{r}^{(k)} = (r_1^{(k)}, r_2^{(k)}, \ldots, r_n^{(k)})$ be a vector in $\{0,1\}^n$ such that

$$r_i^{(k)} = \begin{cases} 1 & \text{if } x_i \in \operatorname{Vars}(f_k), \\ 0 & \text{otherwise.} \end{cases}$$

Let $\mathcal{E}_k(\mathbf{a})$ be a 0-1 random variable defined as follows. If $|\operatorname{Vars}(f_k)|$ is odd, $\mathcal{E}_k(\mathbf{a})$ is always 0. Otherwise, define $\beta_k = |\operatorname{Vars}(f_k)|/2 \pmod 2$ and

$$\mathcal{E}_k(\mathbf{a}) = \begin{cases} 1 & \text{if } (\mathbf{r}^{(k)}, \mathbf{a})_2 = \beta_k, \\ 0 & \text{otherwise.} \end{cases}$$

The main step in the proof of Lemma 14, is the following claim.

▶ **Claim 15.** *If $\mathcal{S}_0$ is a $(\varepsilon, 10\log\frac{1}{\varepsilon})$-subspace avoiding set, then* $\Pr_{\mathbf{a}\in\mathcal{S}_0}[\sum_{k\in[t]}\mathcal{E}_k(\mathbf{a}) \geq t - 2\log s] \leq 5\cdot\varepsilon$.

First, we will prove Lemma 14 using this claim. In order to do that, we will show that for all $\mathbf{a}\in\mathcal{S}_0$ the condition that $(\sum_{k\in[t]}\mathcal{E}_k(\mathbf{a}) < t - 2\log s$ implies $\mathrm{relrk}_{(Y_\mathbf{a},Z_\mathbf{a})}(f|_{\rho_\mathbf{a}}) \leq \frac{1}{s}$.

For any vector $\mathbf{a}\in\mathcal{S}_0$ and for $k\in[t]$, let $Y_{k,\mathbf{a}} = \rho_\mathbf{a}(\mathrm{Vars}(f_k))\cap Y_\mathbf{a}$ and let $Z_{k,\mathbf{a}} = \rho_\mathbf{a}(\mathrm{Vars}(f_k))\cap Z_\mathbf{a}$. By Item 3 in Proposition 4 we know that

$$\mathrm{relrk}_{(Y_\mathbf{a},Z_\mathbf{a})}(f|_{\rho_\mathbf{a}}) = \prod_{k\in[t]}\mathrm{relrk}_{(Y_{k,\mathbf{a}},Z_{k,\mathbf{a}})}(f_k|_{\rho_\mathbf{a}}).$$

We claim that for any $k\in[t]$, if $\mathcal{E}_k(\mathbf{a})=0$, then $\mathrm{relrk}_{(Y_{k,\mathbf{a}},Z_{k,\mathbf{a}})}(f_k|_{\rho_\mathbf{a}}) \leq 1/\sqrt{2}$. To see this, note that, for any $k\in[t]$, $\mathcal{E}_k(\mathbf{a})=1$ if and only if $|\mathrm{Vars}(f_k)|$ is even and $(\mathbf{r}^{(k)},\mathbf{a})_2 = \beta_k$. Now, if $|\mathrm{Vars}(f_k)|$ is odd then we can never have $|Y_{k,\mathbf{a}}| = |Z_{k,\mathbf{a}}|$ and hence by Item 1 of Proposition 4, $\mathrm{relrk}_{(Y_{k,\mathbf{a}},Z_{k,\mathbf{a}})}(f_k|_{\rho_\mathbf{a}}) \leq 1/\sqrt{2}$. Further, if $|\mathrm{Vars}(f_k)|$ is even and $|Y_{k,\mathbf{a}}| - |\mathrm{Vars}(f_k)|/2 \neq 0 \pmod 2$, then again we must have $|Y_{k,\mathbf{a}}| \neq |Z_{k,\mathbf{a}}|$ and hence $\mathrm{relrk}_{(Y_{k,\mathbf{a}},Z_{k,\mathbf{a}})}(f_k|_{\rho_\mathbf{a}}) \leq 1/\sqrt{2}$.

Now if $\sum_{k\in[t]}\mathcal{E}_k(\mathbf{a}) \leq t - 2\log s$ then there exist at least $(2\log s)$ elements $k\in[t]$ such that $\mathcal{E}_k(\mathbf{a})=0$. Hence, $\mathrm{relrk}_{(Y_\mathbf{a},Z_\mathbf{a})}(f|_{\rho_\mathbf{a}}) \leq (1/\sqrt{2})^{2\log s} \leq 1/s$.

Thus, in order to upper bound the probability of the event that $\mathrm{relrk}_{(Y_\mathbf{a},Z_\mathbf{a})}(f|_{\rho_\mathbf{a}}) \geq 1/s$, it suffices to upper bound the probability of $\sum_{k\in[t]}\mathcal{E}_k(\mathbf{a}) \geq t - 2\log s$, which by Claim 15 is at most 5ε. This concludes the proof of Lemma 14.

It remains to prove Claim 15, which we do now. The proof follows a combinatorial proof of the Chernoff bound due to Impagliazzo and Kabanets [16].

Proof of Claim 15. Let $\ell = t - 2\log s$ and $\mathcal{E}(\mathbf{a}) = \sum_{k\in[t]}\mathcal{E}_k(\mathbf{a})$. Let $R(\mathbf{a})$ be a Boolean random variable such that

$$R(\mathbf{a}) = \begin{cases} 1 & \text{if } \sum_{k\in[t]}\mathcal{E}_k(\mathbf{a}) \geq \ell, \\ 0 & \text{otherwise.}\end{cases}$$

Thus, $\Pr_{\mathbf{a}\in\mathcal{S}}[\mathcal{E}(\mathbf{a}) \geq \ell] = \mathbf{E}_{\mathbf{a}\in\mathcal{S}}[R(\mathbf{a})]$. Fix a vector $\mathbf{a}$. Let $\tilde{R}(\mathbf{a})\in[0,1]$ be the random variable defined by $\tilde{R}(\mathbf{a}) = \mathbf{E}_A[\prod_{i\in A}\mathcal{E}_i(\mathbf{a})]$, where $A\subseteq[t]$ is an independently and uniformly randomly chosen subset of size $2\log s$.

We claim that for every $\mathbf{a}\in\mathcal{S}_0$, $R(\mathbf{a}) \leq 2\cdot\tilde{R}(\mathbf{a})$. Assuming this, we get the following.

$$\begin{aligned}\Pr_{\mathbf{a}}[\sum_{k\in[t]}\mathcal{E}_k(\mathbf{a}) \geq \ell] = \mathop{\mathbf{E}}_{\mathbf{a}}[R(\mathbf{a})] &\leq 2\mathop{\mathbf{E}}_{\mathbf{a}}[\mathop{\mathbf{E}}_{A}[\prod_{i\in A}\mathcal{E}_i(\mathbf{a})]] \\ &= 2\mathop{\mathbf{E}}_{A}[\mathop{\mathbf{E}}_{\mathbf{a}}[\prod_{i\in A}\mathcal{E}_i(\mathbf{a})]] = 2\mathop{\mathbf{E}}_{A}[\Pr_{\mathbf{a}}[\prod_{i\in A}\mathcal{E}_i(\mathbf{a})=1]]. \end{aligned} \tag{3}$$

Consider an individual term $\mathcal{E}_A(\mathbf{a}) := \prod_{i\in A}\mathcal{E}_i(\mathbf{a})$ in the above expression. We claim that $\mathbf{E}_\mathbf{a}[\mathcal{E}_A(\mathbf{a})] \leq 2\varepsilon$. To see this, note that we have one of the following two scenarios. Either A contains an i such that $|\mathrm{Vars}(f_i)|$ is odd, in which case $\mathcal{E}_A(\mathbf{a}) = \mathcal{E}_i(\mathbf{a}) = 0$ with probability 1. Otherwise, $|\mathrm{Vars}(f_i)|$ is even for each $i\in A$ and then $\mathbf{a}$ satisfies $\mathcal{E}_A(\mathbf{a})=1$ if and only if $\mathbf{a}$ satisfies the system of linear equations $\{(\mathbf{a},\mathbf{r}^{(i)})_2 = \beta_i \mid i\in A\}$. Since the $\mathbf{r}^{(i)}$'s are non-zero and linearly independent, this system of equations defines an affine subspace of codimension $|A| = 2\log s$. Now, by invoking Claim 9, we get that $\Pr_\mathbf{a}[\prod_{i\in A}\mathcal{E}_i(\mathbf{a})=1] \leq 1/s^2 + \varepsilon \leq 2\varepsilon$, where the final inequality uses $s\geq 1/\varepsilon$. Substituting this back in (3), we get that $\Pr_\mathbf{a}[\mathcal{E}(\mathbf{a}) \geq \ell] \leq 4\varepsilon$.

To complete the proof, we need to show that for every $\mathbf{a} \in \mathcal{S}_0$, $R(\mathbf{a}) \leq 2\tilde{R}(\mathbf{a})$. If $R(\mathbf{a}) = 0$, then this statement is trivial. If not, $R(\mathbf{a}) = 1$. That is, there exist at least ℓ many $k \in [t]$ such that $\mathcal{E}_k(\mathbf{a}) = 1$. Then,

$$\tilde{R}(\mathbf{a}) = \mathop{\mathbf{E}}_{A}[\prod_{i \in A} \mathcal{E}_i(\mathbf{a})] = \Pr_{A}[\text{for all } i \in A,\ \mathcal{E}_i(\mathbf{a}) = 1] \geq \frac{\binom{\ell}{2\log s}}{\binom{t}{2\log s}} = \frac{\binom{t-2\log s}{2\log s}}{\binom{t}{2\log s}} \geq 1/2,$$

where the final inequality follows from the fact that $t \geq (\log s)^3$. This completes the proof of Lemma 14. ◀

The main theorem

We now prove our main theorem. It is restated here for the sake of convenience.

▶ **Theorem 16.** *For any fixed positive $c \in \mathbb{N}$ and $\delta \in (0, 1/2)$, there exists an explicit polynomial family $P_n \in \mathbb{F}[x_1, \ldots, x_n]$ that has multilinear formulas of depth 3 and size at most $O(s)$ where $s = n^c$, but no multilinear formulas of size less than $s^{(1/2)-\delta}$ and depth $\Delta < \log n/100 \log\log n$.*

Proof. We first show that we can assume without loss of generality that $c \geq 10/\delta$. Say this is not the case: we then have $s = n^c < n^{10/\delta}$. Now, let $m = s^{\delta/10} \leq n$. We will then define a polynomial over only the variables $\{x_1, \ldots, x_m\}$. So the number of variables reduces to m and $s = m^{10/\delta}$ (in particular, the new value of c is now $10/\delta$). Thus, we can always reduce the problem to the case when $c \geq 10/\delta$.

So we assume without loss of generality that $s \geq n^{10/\delta}$. We will fix our polynomial P_n to be $P_{n/2,\varepsilon}(X, W)$ as defined in Section 3.2 for $\varepsilon = n/\sqrt{s}$, where the variable set $X = \{x_1, \cdots, x_{n/2}\}$ and $W \subseteq \{x_{n/2+1}, \ldots, x_n\}$ is some fixed set of size $\log \tau \leq \log s$ (since $s \leq n^c$, it is clear that $\log s \leq n/2$). From Lemma 12, we know that $\{P_n \mid n \in \mathbb{N}\}$ is an explicit family of multilinear polynomials such that each P_n is computed by a depth three multilinear formula of size $O(s)$. Further, from Lemma 13, we get that $\text{relrk}_{(Y_\mathbf{a}, Z_\mathbf{a})}(P_n(X, W)|_{\rho_\mathbf{a}}) = 1$ for every choice of $\mathbf{a} \in \mathcal{B}$, where $\mathcal{B}$ is as defined in Section 3.2.

Let us assume that P_n can be computed by a depth Δ multilinear formula Φ of size $s' < s^{1/2-\delta}$. We consider $P_n(X, W)$ as a polynomial from the ring $\mathbb{G}[X]$ where $\mathbb{G} = \mathbb{F}(W)$. We also consider Φ as a multilinear formula computing a polynomial from $\mathbb{G}[X]$ (i.e. we consider the variables from W in Φ as constants from the underlying field $\mathbb{G}$.)

From Lemma 3, we know that P_n can be written as a sum of at most $s'' = s' \cdot n$ many t-product polynomials where $t = \Omega(n^{1/2\Delta})$. Note that as $\Delta < \log n/100 \log\log n$, we have $t = \Omega((\log n)^{50}) \geq (\log s)^{40}$, the latter inequality following from the fact that $\log s = O(\log n)$.

Assume that the t-product polynomials in the above decomposition are $f_1, \ldots, f_{s''} \in \mathbb{G}[X]$. So, we have $P_n = \sum_{i \in [s'']} f_i$. We would like to show that

$$\Pr_{\mathbf{a} \in \mathcal{B}}[\text{relrk}_{(Y_\mathbf{a}, Z_\mathbf{a})}(P_n|_{\rho_\mathbf{a}}) < 1] > 0 \tag{4}$$

which would contradict Lemma 13 and hence prove the theorem.

In order to prove inequality (4), by the sub-additivity property (Proposition 4 Item 2) of the relative-rank, it suffices to show the following.

$$\Pr_{\mathbf{a} \in \mathcal{B}}[\forall i \in [s'], \text{relrk}_{(Y_\mathbf{a}, Z_\mathbf{a})}(f_i|_{\rho_\mathbf{a}}) < 1/s''] > 0.$$

Equivalently, it suffices to prove that

$$\Pr_{\mathbf{a} \in \mathcal{B}}[\exists i \in [s''], \text{relrk}_{(Y_\mathbf{a}, Z_\mathbf{a})}(f_i|_{\rho_\mathbf{a}}) \geq 1/s''] < 1. \tag{5}$$

Recall from Section 3.2 that the set $\mathcal{B}$ is defined to be a subset of the set $\mathcal{S}_0 = \mathcal{S}_{n,\varepsilon} \oplus \mathbf{x}_0$ of size $\tau = \Theta(|\mathcal{S}_0|/n)$. Also, by Claim 9 and Fact 10, it follows that $\mathcal{S}_0$ is an $(\varepsilon, 10\log(1/\varepsilon))$-biased set. By using Lemma 14, we get that for each t-product polynomial f_i, we have that

$$\Pr_{\mathbf{a}\in\mathcal{S}_0}[\mathrm{relrk}_{(Y_\mathbf{a},Z_\mathbf{a})}(f_i|_{\rho_\mathbf{a}}) \geq \frac{1}{s}] \leq 5\varepsilon.$$

Therefore, by a simple union bound, we get that

$$\Pr_{\mathbf{a}\in\mathcal{S}_0}[\exists i : \mathrm{relrk}_{(Y_\mathbf{a},Z_\mathbf{a})}(f_i|_{\rho_\mathbf{a}}) \geq \frac{1}{s}] \leq 5s'' \cdot \varepsilon. \tag{6}$$

As $\mathcal{B}$ is a subset of $\mathcal{S}_0$, we get that

$$\frac{|\mathcal{B}|}{|\mathcal{S}_0|} \cdot \Pr_{\mathbf{a}\in\mathcal{B}}[\exists i : \mathrm{relrk}_{(Y_\mathbf{a},Z_\mathbf{a})}(f_i|_{\rho_\mathbf{a}}) \geq \frac{1}{s}] \leq \Pr_{\mathbf{a}\in\mathcal{S}_0}[\exists i : \mathrm{relrk}_{(Y_\mathbf{a},Z_\mathbf{a})}(f_i|_{\rho_\mathbf{a}}) \geq \frac{1}{s}]. \tag{7}$$

Hence, using the inequalities (6), (7) and the fact that $|\mathcal{B}|/|\mathcal{S}_0| = \Theta(1/n)$, we get

$$\Pr_{\mathbf{a}\in\mathcal{B}}[\exists i : \mathrm{relrk}_{(Y_\mathbf{a},Z_\mathbf{a})}(f_i|_{\rho_\mathbf{a}}) \geq \frac{1}{s}] \leq O\left(\frac{s''n^2}{\sqrt{s}}\right) \leq \frac{s'n^4}{\sqrt{s}}. \tag{8}$$

As $s \geq n^{10/\delta}$ we get $n^4 \leq s^{\delta/2}$. Therefore, as long as $s' \leq O(s^{1/2-\delta})$, inequality (5) is satisfied, which then implies that inequality (4) is also satisfied.

If inequality (4) is satisfied, then there exists a partitioning function $\rho_\mathbf{a}$ for $\mathbf{a} \in \mathcal{B}$ such that $\mathrm{relrk}_{(Y_\mathbf{a},Z_\mathbf{a})}(P_n|_{\rho_\mathbf{a}}) < 1$. This contradicts Lemma 13 which tells us $\mathrm{relrk}_{(Y_\mathbf{a},Z_\mathbf{a})}(P_n|_{\rho_\mathbf{a}}) = 1$ with respect to every partitioning function $\rho_\mathbf{a}$ ($\mathbf{a} \in \mathcal{B}$). ◀

References

1. M. Ajtai. Σ^1_1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24(1):1–48, 1983. doi:10.1016/0168-0072(83)90038-6.
2. Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple construction of almost k-wise independent random variables. *Random Struct. Algorithms*, 3(3):289–304, 1992. doi:10.1002/rsa.3240030308.
3. Noga Alon, Mrinal Kumar, and Ben Lee Volk. Unbalancing sets and an almost quadratic lower bound for syntactically multilinear arithmetic circuits, 2017. URL: https://mrinalkr.bitbucket.io/papers/synt-multilinear.pdf.
4. Kazuyuki Amano. k-subgraph isomorphism on AC^0 circuits. *Computational Complexity*, 19(2):183–210, 2010. doi:10.1007/s00037-010-0288-y.
5. Maria Luisa Bonet and Samuel R. Buss. Size-depth tradeoffs for boolean formulae. *Information Processing Letters*, 49(3):151–155, 1994. doi:10.1016/0020-0190(94)90093-0.
6. Richard P. Brent. The parallel evaluation of general arithmetic expressions. *Journal of the ACM*, 21(2):201–206, 1974. doi:10.1145/321812.321815.
7. Suryajith Chillara, Christian Engels, Nutan Limaye, and Srikanth Srinivasan. A Near-Optimal Depth-Hierarchy Theorem for Small-Depth Multilinear Circuits. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:066, 2018. URL: https://eccc.weizmann.ac.il/report/2018/066/s.
8. Suryajith Chillara, Nutan Limaye, and Srikanth Srinivasan. Small-depth multilinear formula lower bounds for iterated matrix multiplication, with applications. In *STACS*, volume 96 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

9 Anindya De, Omid Etesami, Luca Trevisan, and Madhur Tulsiani. Improved pseudorandom generators for depth 2 circuits. In *APPROX-RANDOM*, volume 6302 of *Lecture Notes in Computer Science*, pages 504–517. Springer, 2010.

10 Zeev Dvir and Shachar Lovett. Subspace evasive sets. In *STOC*, pages 351–358. ACM, 2012.

11 Zeev Dvir, Guillaume Malod, Sylvain Perifel, and Amir Yehudayoff. Separating multilinear branching programs and formulas. In *proceedings of Symposium on Theory of Computing (STOC)*, pages 615–624, 2012. doi:10.1145/2213977.2214034.

12 Merrick Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical systems theory*, 17(1):13–27, Dec 1984. doi:10.1007/BF01744431.

13 J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965. URL: http://www.jstor.org/stable/1994208.

14 J. Håstad. *Computational limitations of small-depth circuits*. ACM doctoral dissertation award. MIT Press, 1987. URL: https://books.google.co.in/books?id=_h0ZAQAAIAAJ.

15 Pavel Hrubeš and Amir Yehudayoff. Homogeneous formulas and symmetric polynomials. *Computational Complexity*, 20(3):559–578, 2011.

16 Russell Impagliazzo and Valentine Kabanets. Constructive proofs of concentration bounds. In *APPROX-RANDOM*, volume 6302 of *Lecture Notes in Computer Science*, pages 617–631. Springer, 2010.

17 Neeraj Kayal, Vineet Nair, and Chandan Saha. Separation between read-once oblivious algebraic branching programs (ROABPs) and multilinear depth three circuits. In *proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 46:1–46:15, 2016. doi:10.4230/LIPIcs.STACS.2016.46.

18 P. M. Lewis, R. E. Stearns, and J. Hartmanis. Hierarchies of memory limited computations. In *6th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965)(FOCS)*, volume 00, pages 179–190, 10 1965. doi:10.1109/FOCS.1965.11.

19 Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993. doi:10.1137/0222053.

20 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997. doi:10.1007/BF01294256.

21 Ran Raz. Multilinear-$\mathsf{NC}^2 \neq$ multilinear-NC^1. In *proceedings of Foundations of Computer Science (FOCS)*, pages 344–351, 2004. doi:10.1109/FOCS.2004.42.

22 Ran Raz. Separation of multilinear circuit and formula size. *Theory of Computing*, 2(1):121–135, 2006. doi:10.4086/toc.2006.v002a006.

23 Ran Raz, Amir Shpilka, and Amir Yehudayoff. A lower bound for the size of syntactically multilinear arithmetic circuits. *SIAM Journal of Computing*, 38(4):1624–1647, 2008. doi:10.1137/070707932.

24 Ran Raz and Amir Yehudayoff. Balancing syntactically multilinear arithmetic circuits. *Computational Complexity*, 17(4):515–535, 2008. doi:10.1007/s00037-008-0254-0.

25 Ran Raz and Amir Yehudayoff. Lower bounds and separations for constant depth multilinear circuits. *Computational Complexity*, 18(2):171–207, 2009. doi:10.1007/s00037-009-0270-8.

26 Benjamin Rossman. *Average-case Complexity of Detecting Cliques*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2010. AAI0823246.

27 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. Github survey, 2015. URL: https://github.com/dasarpmar/lowerbounds-survey/releases/.

28 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5:207–388, March 2010. doi:10.1561/0400000039.

Restricted Max-Min Fair Allocation

Siu-Wing Cheng
Department of Computer Science and Engineering, HKUST, Hong Kong
scheng@cse.ust.hk
https://orcid.org/0000-0002-3557-9935

Yuchen Mao
Department of Computer Science and Engineering, HKUST, Hong Kong
ymaoad@cse.ust.hk
https://orcid.org/0000-0002-1075-344X

Abstract

The restricted max-min fair allocation problem seeks an allocation of resources to players that maximizes the minimum total value obtained by any player. It is NP-hard to approximate the problem to a ratio less than 2. Comparing the current best algorithm for estimating the optimal value with the current best for constructing an allocation, there is quite a gap between the ratios that can be achieved in polynomial time: $4+\delta$ for estimation and $6+2\sqrt{10}+\delta \approx 12.325+\delta$ for construction, where δ is an arbitrarily small constant greater than 0. We propose an algorithm that constructs an allocation with value within a factor $6+\delta$ from the optimum for any constant $\delta > 0$. The running time is polynomial in the input size for any constant δ chosen.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Fair allocation, approximation, local search

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.37

Related Version A full version of the paper can be found at `https://arxiv.org/abs/1804.10902`.

Funding Supported by the Research Grants Council, Hong Kong, China (project no. 16201116).

1 Introduction

Background. Let P be a set of m players. Let R be a set of n indivisible resources. Resource $r \in R$ is worth a non-negative integer value v_{pr} for player $p \in P$. An allocation is a partition of R into disjoint subsets $\{C_p : p \in P\}$ so that player p is assigned the resources in C_p. The *max-min fair allocation* problem is to distribute resources to players so that the minimum total value of resources received by any player is maximized. The *value of an allocation* is $\min_{p\in P} \sum_{r\in C_p} v_{pr}$. So we want to find an allocation with maximum value.

No algorithm can achieve an approximation ratio less than 2 unless P = NP [5]. Bansal and Sviridenko [4] proposed the configuration LP and showed that it can be solve to any desired accuracy in polynomial time. The configuration LP turns out to be a useful tool for this problem. Using it, approximation ratios of $O(\sqrt{m \log m})$ and $O(n^{\delta} \log n)$ for any $\delta \geqslant \frac{9 \log\log n}{\log n}$ have been attained [3, 4, 6, 12]. In this paper, we focus on the *restricted case* in which each resource r is desired by some subset of players, and has the same value v_r for those who desire it and value 0 for the rest. Even in this case, no approximation ratio better than 2 can be obtained unless P = NP [5]. Bansal and Sviridenko [4] designed a $O\big(\frac{\log\log m}{\log\log\log m}\big)$-approximation algorithm which is based on rounding the configuration LP. Feige [8] proved that the integrality gap of the configuration LP is bounded by a constant


(large and unspecified). His proof was made constructive by Haeupler et al. [10], and hence, results in a constant-approximation algorithm. Asadpour et al. [2] proved that the integrality gap of the configuration LP is at most 4. As a consequence, by solving the configuration LP approximately, one can estimate the optimal solution value within a factor of $4 + \delta$ for any constant $\delta > 0$. However, it is not known how to construct a $(4 + \delta)$-approximate allocation in polynomial time. Annamalai et al. [1] developed a $(6+2\sqrt{10}+\delta)$-approximation algorithm. Their algorithm is purely combinatorial, but the analysis still relies on the configuration LP. There is quite a gap between the current best estimation ratio $4 + \delta$ and the current best approximation ratio $6 + 2\sqrt{10} + \delta \approx 12.325 + \delta$.

If one constrains the *restricted case* further by requiring $v_r \in \{1, \varepsilon\}$ for some fixed constant $\varepsilon \in (0, 1)$, then it becomes the $(1, \varepsilon)$-*restricted case*. Golovin proposed an $O(\sqrt{n})$-approximation algorithm [9]. Chan et al. [7] showed that it is still NP-hard to obtain an approximation ratio less than 2 and that the algorithm in [1] achieves an approximation ratio of 9 in this case. The analysis in [7] does not rely on the configuration LP.

Our contributions. We propose an algorithm for the restricted max-min fair allocation problem that achieves an approximation ratio of $6 + \delta$ for any constant $\delta > 0$. It runs in polynomial time for any constant δ chosen. Our algorithm uses the same framework as [1]: we maintain a stack of layers to record the relation between players and resources, and use lazy update and a greedy strategy to achieve a polynomial running time.

Our first contribution is a greedy strategy that is much more aggressive than that in [1]. Let τ^* be the optimal solution value. Let $\lambda > 2$ be the target approximation ratio. To obtain a λ-approximate solution, the value of resources a player need is τ^*/λ. The greedy strategy in [1] considers a player greedy if that player claims at least $\tau^*/2$ worth of resources, which is more than needed. In contrast, we consider a player greedy if it claims (nearly) the largest total value among all the candidates. When building the stack, both [1] and we add greedy players and the resources claimed by them to the stack. Intuitively, our more aggressive definition of greedy leads to faster growth of the stack, and hence a significantly smaller approximation ratio can be achieved.

Our aggressive strategy brings challenge to the analysis that approaches in [1, 7] cannot cope with. Our second contribution is a new analysis tool: an injection that maps a lot of players in the stack to their *competing players* who can access resources of large total value. Since players added to the stack must be greedy, they claim more than their competing players. Therefore, such an injection allows us to conclude that players in the stack claim large worth of resources. By incorporating competing players into the analysis framework in [7], we improve the approximation ratio to $6 + \delta$. Our analysis does not rely on the configuration LP, and is purely combinatorial.

2 Preliminaries

Let τ^* be the optimal solution value. Let λ denote our target approximation ratio. Given any value $\tau \leqslant \tau^*$, our algorithm returns an allocation of value τ/λ in polynomial time. We will show how to combine this algorithm with binary search to obtain an allocation of value at least τ^*/λ in the end. We assume that τ is no more than τ^* in the rest of this section.

Bipartite graph and thin edges. A resource r is *fat* if $v_r \geqslant \tau/\lambda$; otherwise, r is *thin*. Let G be the bipartite graph formed by representing the players in P and the fat resources in R as vertices, and connecting a player p and a fat resource r by an edge if p desires r. Similarly,

players and thin resources form a hypergraph, namely, there are vertices representing players in P and thin resources in R, and a player p and a subset B of thin resources form an edge (p, B) if p desires all resources in B. (Note that $(p, \emptyset)$ is included.) We call the edges of this hypergraph *thin edges*. For a subset B of thin resources, we define the value of B as $value(B) = \sum_{r \in B} v_r$. For a thin edge $e = (p, B)$, its value $value(e)$ is defined to be the total value of thin resources covered by it, i.e., $value(e) = value(B)$. We use uppercase calligraphic letters to denote subsets of thin edges. Given a set $\mathcal{S}$ of thin edges, define $value(\mathcal{S})$ to be the total value of the thin resources covered by $\mathcal{S}$.

Partial allocation. Since our target approximation ratio is λ, it suffices to assign each player p either a single fat resource r such that $\{p, r\}$ is an edge of G, or a subset B of thin resources such that (p, B) is a thin edge and $value(B) \geqslant \tau/\lambda$. Hence, it suffices to consider allocations that consist of two parts, one being a maximum matching M of G and the other being a subset $\mathcal{E}$ of thin edges such that every player is covered by either M or $\mathcal{E}$, no two edges in $\mathcal{E}$ share any resource, and every edge in $\mathcal{E}$ has value at least τ/λ.

Our algorithm will start with an arbitrary maximum matching of G alone, grow and update the set $\mathcal{E}$ of thin edges, and whenever necessary, update the maximum matching as well. We call the intermediate solutions *partial allocations*. A **partial allocation** consists of a maximum matching M of G and a set $\mathcal{E}$ of thin edges such that: (i) no player is covered by both M and $\mathcal{E}$; (ii) no two edges in $\mathcal{E}$ share any resource; (iii) every edge $(p, B) \in \mathcal{E}$ is minimal in the sense that $value(B) \geqslant \tau/\lambda$ and every proper subset $B' \subset B$ has value less than τ/λ. We say a player p is *satisfied* by a partial allocation if p is covered by M or $\mathcal{E}$. A partial allocation is an allocation if it satisfies every player.

Node-disjoint paths. We define a family of networks which are heavily used in both our algorithm and its analysis. With respect to any arbitrary maximum matching M of G, define G_M to be a directed bipartite graph such that G_M has the same vertex set as G (i.e., players and fat resources), there is a directed edge from player p to resource r if $\{p, r\}$ is an edge of G that is not used in M, and there is a directed edge from resource r to player p if $\{p, r\}$ is an edge of G that is used in M.

We use P_M and $\overline{P}_M$ to denote the subsets of players matched and unmatched in M, respectively. Given $S \subseteq \overline{P}_M$ and $T \subseteq P$, we use $G_M(S, T)$ to denote the problem of finding the maximum number of node-disjoint paths from S to T in G_M. This problem will arise in this paper for different choices of S and T. A feasible solution of $G_M(S, T)$ is just any set of node-disjoint paths from S to T in G_M. An optimal solution maximizes the number of paths. Let $f_M(S, T)$ denote the size of an optimal solution of $G_M(S, T)$. In case that $S \cap T \neq \emptyset$, a feasible solution may allow a path from a player $p \in S \cap T$ to itself, i.e., a path with no edge. We call such a path a *trivial path*. Other paths are *non-trivial*.

Let Π be any feasible solution of $G_M(S, T)$. The paths in Π originate from a subset of S, which we call the *sources*, and terminate at a subset of T, which we call the *sinks*. We denote the sets of sources and sinks by $source(\Pi)$ and $sink(\Pi)$, respectively. A trivial path has only one node which is both its source and sink. We use Π^0 and Π^+ to denote the sets of the trivial paths and the non-trivial paths in Π, respectively.

An optimal solution of $G_M(S, T)$ can be found by solving a maximum s-t flow problem as follows. Add a super source s and directed edges from s to all vertices in S. Add a super sink t and directed edges from all vertices in T to t. Set the capacities of all edges to 1. Find an integral maximum flow in the resulting network. The paths in G_M used by this maximum flow is an optimal solution of $G_M(S, T)$. Node-disjointness is ensured because, in the s-t flow network, each player has in-degree at most one and each resource has out-degree at most one.

Let π be a non-trivial path from $\bar{P}_M$ to P in G_M. If we ignore the directions of edges in π, then π is called an *alternating path* in the matching literature [11]. We use $M \oplus \pi$ to denote the result of flipping π, i.e., removing the edges in $\pi \cap M$ from the matching and adding the edges in $\pi \setminus M$ to the matching. $M \oplus \pi$ is also a maximum matching of G. We can extend the above operation and form $M \oplus \Pi^+$ for any feasible solution Π of $G_M(S,T)$ for any $S \subseteq \bar{P}_M$ and any $T \subseteq P$. $M \oplus \Pi^+$ is a maximum matching of G. The following results follow from basic theories of matching and network flow.

▶ **Claim 2.1.** *For any maximum matchings M and M' of G, (i) $f_M(\bar{P}_M, \bar{P}_{M'}) = |\bar{P}_M|$, and (ii) for every subset T of players, $f_M(\bar{P}_M, T) = f_{M'}(\bar{P}_{M'}, T)$.*

If one treats Π like a flow in the network G_M, then $G_{M \oplus \Pi^+}$ behaves like the residual graph with respect to Π. Claims 2.2 and 2.3 below concerns with augmentation using $G_{M \oplus \Pi^+}$.

▶ **Claim 2.2.** *Let Π be a feasible solution of $G_M(S,T)$. Then, $M \oplus \Pi^+$ is a maximum matching of G, so the directed bipartite graph $G_{M \oplus \Pi^+}$ is well defined. Also, Π is an optimal solution of $G_M(S,T)$ if and only if $G_{M \oplus \Pi^+}$ contains no path from $S \setminus source(\Pi)$ to $T \setminus sink(\Pi)$.*

▶ **Claim 2.3.** *Let Π be a feasible solution of $G_M(S,T)$. Suppose that $G_{M \oplus \Pi^+}$ contains a path π from $S \setminus source(\Pi)$ to $T \setminus sink(\Pi)$. We can use π to augment Π to a feasible solution Π' of $G_M(S,T)$ such that $|\Pi'| = |\Pi| + 1$, the vertex set of Π' is a subset of the vertices in $\Pi \cup \{\pi\}$, $source(\Pi') = source(\Pi) \cup \{source(\pi)\}$, and $sink(\Pi') = sink(\Pi) \cup \{sink(\pi)\}$.*

We can also push flow along a path in $G_{M \oplus \Pi^+}$ from $sink(\Pi)$ to T. This reroutes the flow in G_M without changing its flow value. Claim 2.4 below gives a precise statement.

▶ **Claim 2.4.** *Let Π be a feasible solution of $G_M(S,T)$. Suppose that there is a non-trivial path π in $G_{M \oplus \Pi^+}$ from $sink(\Pi)$ to T. Clearly, $sink(\pi) \notin sink(\Pi)$ because every node in $sink(\Pi)$ has zero in-degree in $G_{M \oplus \Pi^+}$. We can use π to convert Π to a feasible solution Π' of $G_M(S,T)$ such that $|\Pi'| = |\Pi|$, the vertex set of Π' is a subset of the vertices in $\Pi \cup \{\pi\}$, $source(\Pi') = source(\Pi)$, and $sink(\Pi') = (sink(\Pi) \setminus \{source(\pi)\}) \cup \{sink(\pi)\}$.*

3 The Algorithm

3.1 Overview

We give an overview of the common framework that our algorithm shares with that in [1]. Let M and $\mathcal{E}$ denote the maximum matching of G and the set of thin edges in the current partial allocation, respectively. Let p_0 be an arbitrary player who is not yet satisfied.

To satisfy p_0, the simplest case is that we can find a minimal thin edge (p_0, B) such that $value(B)$ is at least τ/λ and B excludes the resources covered by edges in $\mathcal{E}$, i.e., not blocked by any thin edge in $\mathcal{E}$. We can extend the partial allocation by adding (p_0, B) to $\mathcal{E}$.

More generally, we can use any thin edge (q, B) such that B meets the above requirements even if $q \neq p_0$, provided that there is a path from p_0 to q in G_M. If $q \neq p_0$, such a path is an alternating path in G with respect to M, and q is matched by M. We can flip this path to match p_0 with a fat resource and then include (q, B) in $\mathcal{E}$ to satisfy q.

We may have the situation that the thin edge (q, B) mentioned above is blocked by some thin edges in $\mathcal{E}$. Pick such a (q, B) arbitrarily, and call it (q_0, B_0). Let $\{(p_1, B'_1), \ldots, (p_k, B'_k)\}$ be the thin edges in $\mathcal{E}$ that block (q_0, B_0), i.e., $B_0 \cap B'_i \neq \emptyset$ for $i \in [1, k]$. To make (q_0, B_0) unblocked, we need to satisfy each player p_i, $i \in [1, k]$, with a fat resource or another thin edge. Afterwards, we can satisfy p_0 as before. To record the different states of the algorithm, we initialize a stack to contain $(p_0, \emptyset)$ as the first layer and then create another layer on top

that stores the sets $\mathcal{X}_2 = \{(q_0, B_0)\}$ and $\mathcal{Y}_2 = \{(p_1, B'_1), \ldots, (p_k, B'_k)\}$ among other things for bookkeeping. We change our focus to satisfy the set of players $Y_2 = \{p_1, \ldots, p_k\}$.

To satisfy a player in Y_2 (by a new edge), we need to identify a minimal thin edge (q_1, B_1) such that $value(B_1)$ is at least τ/λ and G_M contains two node-disjoint paths from $\{p_0\} \cup Y_2$ to $\{q_0, q_1\}$, and we also require B_1 to exclude the resources already covered by thin edges in the current stack (i.e., $\mathcal{X}_2$ and $\mathcal{Y}_2$) because the current plan to satisfy q_0 in the future involves some of these thin resources. If (q_1, B_1) is blocked by thin edges in $\mathcal{E}$, we initialize a set $\mathcal{X}_3 = \{(q_1, B_1)\}$; otherwise, we initialize a set $\mathcal{I} = \{(q_1, B_1)\}$. Ideally, if (q_1, B_1) is unblocked, we could immediately make some progress. Since there are two node-disjoint paths from $\{p_0\} \cup Y_2$ to $\{q_0, q_1\}$, q_1 is either reachable from p_0 or a player in Y_2. In the former case, we can satisfy p_0; in the latter case, the path from Y_2 to q_1 must be node-disjoint from the path from p_0 to q_0. We can remove a blocking edge from $\mathcal{Y}_2$ without affecting the alternating path from p_0 to q_0. But we would not do so because, as argued in [1], in order to achieve a polynomial running time, we should let $\mathcal{I}$ grow bigger so that a large progress can be made.

Since there are multiple players in Y_2 to be satisfied, we continue to look for another minimal thin edge (q_2, B_2) such that G_M contains three node-disjoint paths from $\{p_0\} \cup Y_2$ to $\{q_0, q_1, q_2\}$, $value(B_2) \geqslant \tau/\lambda$, and B_2 excludes the resources covered by thin edges in the current stack (i.e.,$\mathcal{X}_2 \cup \mathcal{Y}_2 \cup \mathcal{X}_3$) and $\mathcal{I}$. If (q_2, B_2) is blocked by thin edges in $\mathcal{E}$, we add (q_2, B_2) to $\mathcal{X}_3$; otherwise, we add it to $\mathcal{I}$. After collecting all such thin edges in $\mathcal{X}_3$ and $\mathcal{I}$, we construct the set $\mathcal{Y}_3$ of thin edges in the current partial allocation that block $\mathcal{X}_3$. Then, we add a new top layer to the stack that stores $\mathcal{X}_3$ and $\mathcal{Y}_3$ among other things for bookkeeping. Then, we turn our attention to satisfying the players in $\mathcal{Y}_3$ with new edges and so on. These repeated additions of layers to the stack constitute the build phase of the algorithm.

The build phase stops when we have enough thin edges in $\mathcal{I}$ to satisfy a predetermined fraction of players in $\mathcal{Y}_l$ for some l, and then we shrink this layer and delete all layers above it. The above is repeated until $\mathcal{I}$ is not large enough to satisfy the predetermined fraction of players in any $\mathcal{Y}_l$ in the stack. These repeated removal of layers constitute the collapse phase of the algorithm. At the end of the collapse phase, we switch back to the build phase.

The alternation of build and collapse phases continues until we succeed in satisfying player p_0, our original goal, that is stored in the bottommost layer in the stack.

A greedy strategy is also used for achieving a polynomial running time. In [1], when a blocked thin edge (q, B) is picked and added to $\mathcal{X}_l$ for some l, B is required to be a minimal set of value at least $\tau/2$, which is more than τ/λ. Intuitively, if such an edge is blocked, it must be blocked by many edges. Hence, the strategy leads to fast growth of stack. We use a more aggressive strategy: we allow the value of B to be as large as $\tau + \tau/\lambda$, and among all candidates, we pick the (q, B) with (nearly) the largest value. Our strategy leads to faster growth of the stack, and hence, a polynomial running time can be achieved for smaller λ.

3.2 Notation and definitions

A *state of the algorithm* consists of several components, namely, M, $\mathcal{E}$, a stack of layers, and a global variable $\mathcal{I}$ that stores a set of thin edge. The layers in the stack are indexed starting from 1 at the bottom. For $i \geqslant 1$, the i-th layer is a 4-tuple $(\mathcal{X}_i, \mathcal{Y}_i, d_i, z_i)$, where $\mathcal{X}_i$ and $\mathcal{Y}_i$ are sets of thin edges, and d_i and z_i are two numeric values that we will explain later. We use I, X_i and Y_i to denote the set of players covered by edges in $\mathcal{I}$, $\mathcal{X}_i$ and $\mathcal{Y}_i$, respectively. For any $k \geqslant 1$, let $\mathcal{X}_{\leqslant k}$ denote $\bigcup_{i=1}^{k} \mathcal{X}_i$, and $\mathcal{Y}_{\leqslant k}$, $X_{\leqslant k}$, and $Y_{\leqslant k}$ are similarly defined.

The sets $\mathcal{X}_i$ and $\mathcal{Y}_i$ are defined inductively. At the beginning of the algorithm, $\mathcal{X}_1 = \emptyset$, $\mathcal{Y}_1 = \{(p_0, \emptyset)\}$, $d_1 = z_1 = 0$, and $\mathcal{I} = \emptyset$. The first layer in the stack is thus $(\emptyset, \{(p_0, \emptyset)\}, 0, 0)$.

Consider the construction of the $(k+1)$-th layer in an execution of the build phase. When it first starts, $\mathcal{X}_{k+1}$ is initialized to be empty. We say that p is **addable** if $f_M(Y_{\leqslant k}, X_{\leqslant k+1} \cup I \cup \{p\}) > f_M(Y_{\leqslant k}, X_{\leqslant k+1} \cup I)$. Note that this definition depends on $X_{\leqslant k+1} \cup I$, so adding edges to $\mathcal{X}_{k+1}$ and $\mathcal{I}$ may affect the addability of players. Given an addable player p, we say that a thin edge (p, B) is **addable** if $value(B) \in [\tau/\lambda, \tau + \tau/\lambda]$ and B excludes resources currently in $\mathcal{X}_{\leqslant k+1} \cup \mathcal{Y}_{\leqslant k} \cup \mathcal{I}$. An addable thin edge (p, B) is **unblocked** if there exists a subset $B' \subseteq B$ such that $value(B') \geqslant \tau/\lambda$ and B' excludes resources used in $\mathcal{E}$. Otherwise, (p, B) is **blocked**. During the construction of the $(k+1)$th layer, the algorithm adds some blocked addable thin edges to $\mathcal{X}_{k+1}$ and some unblocked addable thin edges to $\mathcal{I}$. When the growth of $\mathcal{X}_{k+1}$ stops, the algorithm constructs $\mathcal{Y}_{k+1}$ as the set of the thin edges in $\mathcal{E}$ that share resource(s) with some edge(s) in $\mathcal{X}_{k+1}$. Edges in $\mathcal{Y}_{k+1}$ are called **blocking** edges.

After constructing $\mathcal{X}_{k+1}$ and $\mathcal{Y}_{k+1}$ and growing $\mathcal{I}$, we define $d_{k+1} := f_M(Y_{\leqslant k}, X_{\leqslant k+1} \cup I)$ and $z_{k+1} := |X_{k+1}|$. The values d_{k+1} and z_{k+1} do not change once computed unless the layer L_{k+1} is destructed in the collapse phase, although $f_M(Y_{\leqslant k}, X_{\leqslant k+1} \cup I)$ and $|X_{k+1}|$ may change subsequently. The values d_{k+1} and z_{k+1} are introduced only for the analysis.

Whenever we complete the construction of a new layer in the stack, we check whether any existing layer is collapsible. If so, we leave the build phase and enter the collapse phase, during which the stack is shrunk and the current partial allocation is updated. We stay in the collapse phase until no layer is collapsible. If the stack has become empty, we are done as the player p_0 has been satisfied. Otherwise, we reenter the build phase. We give the detailed specification of the build and collapse phases in the following.

3.3 Build phase

Assume that the stack currently contains layers $L_1, \ldots, L_k$ with L_k at the top. Let M and $\mathcal{E}$ denote the maximum matching in G and the set of thin edges in the current partial allocation, respectively. The following routine Build constructs the next layer L_{k+1}.

Build$(M, \mathcal{E}, \mathcal{I}, (L_1, \cdots, L_k))$

1. Initialize $\mathcal{X}_{k+1}$ to be the empty set.
2. If there is an addable player p and an unblocked addable edge (p, B), then:
   a. take a minimal subset $B' \subseteq B$ such that $value(B') \geqslant \tau/\lambda$ and B' excludes the resources used in $\mathcal{E}$ (we call (p, B') a *minimal unblocked addable edge*),
   b. add (p, B') to $\mathcal{I}$,
   c. go back to step 2.
3. When we come to step 3, no unblocked addable edge is left. If there are no (blocked) addable edges, go to step 4. For each addable player p who is incident to at least one addable edge, identify one *maximal blocked addable edge* (p, B) such that $B \not\subset B'$ for any blocked addable edge (p, B'). Pick the edge with the largest value among those identified, add it to $\mathcal{X}_{k+1}$, and repeat step 3.
4. At this point, the construction of $\mathcal{X}_{k+1}$ is complete. Let $\mathcal{Y}_{k+1}$ be the set of the thin edges in $\mathcal{E}$ that share resource(s) with some thin edge(s) in $\mathcal{X}_{k+1}$.
5. Compute $d_{k+1} := f_M(Y_{\leqslant k}, X_{\leqslant k+1} \cup I)$ and $z_{k+1} := \left|X_{k+1}\right|$.
6. Push the new layer $L_{k+1} = (\mathcal{X}_{k+1}, \mathcal{Y}_{k+1}, d_{k+1}, z_{k+1})$ onto the stack.

Build differs from its counterpart in [1] in several places, particularly in step 3. First, we requires blocked addable edges to be maximal while [1] only considers minimal addable edges of value at least $\tau/2$. Second, when adding addable edges to $\mathcal{X}_{k+1}$, we pick the one with (nearly) the largest value. In contrast, [1] arbitrarily picks one addable edge.

Table 1 Let ℓ denote the highest layer index in the current stack. Let M and $\mathcal{E}$ be the maximum matching and the set of thin edges in the current partial allocation.

Invariant 1	Every edge in $\mathcal{I}$ has value in $[\tau/\lambda, 2\tau/\lambda]$. Every edge in $\mathcal{X}_{\leqslant \ell}$ has value in $[\tau/\lambda, \tau+\tau/\lambda]$. No two edges from $\mathcal{X}_{\leqslant \ell}$ and $\mathcal{I}$ (both edges from either set or one edge from each set) cover the same player or share any resource.
Invariant 2	No edge in $\mathcal{E}$ shares any resource with any edge in $\mathcal{I}$.
Invariant 3	For all $i \in [1, \ell]$, every edge in $\mathcal{X}_i$ shares some resource(s) with some edge(s) in $\mathcal{Y}_i$ but not with any edge in $\mathcal{E} \setminus \mathcal{Y}_i$.
Invariant 4	$\mathcal{Y}_2, \ldots, \mathcal{Y}_\ell$ are disjoint subsets of $\mathcal{E}$. ($\mathcal{Y}_1 = \{(p_0, \emptyset)\}$ is not.)
Invariant 5	For all $i \in [1, \ell]$, no edge in $\mathcal{Y}_i$ shares any resource with any edge in $\mathcal{X}_j$ for any $j \neq i$.
Invariant 6	$f_M(Y_{\leqslant \ell-1}, I) = \lvert I \rvert$.
Invariant 7	For all $i \in [1, \ell-1]$, $f_M(Y_{\leqslant i}, X_{\leqslant i+1} \cup I) \geqslant d_{i+1}$.

Is it possible that $\mathcal{X}_{k+1} = \emptyset$ and $\mathcal{Y}_{k+1} = \emptyset$? We will establish Lemma 4.1 in Section 5.2, which implies that if $\mathcal{Y}_{k+1}$ is empty, then some layer below L_{k+1} is collapsible. As a result, the algorithm will enter the collapse phase next and L_{k+1} will be removed.

▶ **Lemma 3.1.** BUILD *runs in poly*(m, n) *time.*

▶ **Lemma 3.2.** BUILD *maintains invariants 1–7 in Table 1.*

3.4 Collapse phase

Let M be the maximum matching in the current partial allocation. Let $(L_1, L_2, \ldots, L_\ell)$ be the current stack. Deciding whether a layer can be collapsed requires a decomposition of $\mathcal{I}$.

Collapsibility. Let $\mathcal{I}_1 \cup \mathcal{I}_2 \cup \cdots \mathcal{I}_\ell$ be some partition of $\mathcal{I}$. Let I_i denote the set of players covered by $\mathcal{I}_i$. We use $\mathcal{I}_{\leqslant j}$ and $I_{\leqslant j}$ to denote $\bigcup_{i=1}^{j} \mathcal{I}_i$ and $\bigcup_{i=1}^{j} I_i$, respectively. Note that $\lvert I_i \rvert = \lvert \mathcal{I}_i \rvert$ by invariant 1. The partition $\mathcal{I}_1 \cup \mathcal{I}_2 \cup \cdots \mathcal{I}_\ell$ is a **canonical decomposition** of $\mathcal{I}$ if for all $i \in [1, \ell]$, $f_M(Y_{\leqslant i}, I_{\leqslant i}) = f_M(Y_{\leqslant i}, I) = \lvert I_{\leqslant i} \rvert = \lvert \mathcal{I}_{\leqslant i} \rvert$. [1]

▶ **Lemma 3.3** ([1]). *In poly*(ℓ, m, n) *time, one can compute a canonical decomposition* $\mathcal{I}_1 \cup \mathcal{I}_2 \cup \ldots \mathcal{I}_\ell$ *of* $\mathcal{I}$ *and a canonical solution of* $G_M(Y_{\leqslant \ell}, I)$ *which can be partitioned into a disjoint union* $\Gamma_1 \cup \Gamma_2 \cup \cdots \Gamma_\ell$ *such that for every* $i \in [1, \ell]$, Γ_i *is a set of* $\lvert I_i \rvert$ *paths from* Y_i *to* I_i.

The canonical decomposition and solution can be obtained by starting with an optimal solution of $G_M(Y_1, I)$ and successively augment it (using Claim 2.3) to optimal solutions of $G_M(Y_{\leqslant 2}, I), \ldots, G_M(Y_{\leqslant \ell}, I)$. The resulting optimal solution of $G_M(Y_{\leqslant \ell}, I)$ is an canonical solution, and also induces a canonical decomposition of $\mathcal{I}$.

Consider Γ_i. The sources (which are also sinks) of the trivial paths in Γ_i can be satisfied by a new thin edge from $\mathcal{I}_i$. Recall that the non-trivial paths in Γ_i are alternating paths of M. Their sources can be satisfied by fat resources if we flip these alternating paths and satisfy the sinks with thin edges from $\mathcal{I}_i$. If we do so, then edges in $\mathcal{Y}_i$ that cover $\mathit{source}(\Gamma_i)$ can be safely removed from $\mathcal{E}$ as the players in $\mathit{source}(\Gamma_i)$ are satisfied by new edges, and from $\mathcal{Y}_i$ since they no longer block edges in $\mathcal{X}_i$. A layer is collapsible if a certain portion of its blocking edges can be removed. More precisely, for any $i \in [0, \ell]$, L_i is **collapsible** if there is a canonical decomposition $\mathcal{I}_0 \cup \mathcal{I}_1 \cup \ldots \mathcal{I}_\ell$ of $\mathcal{I}$ such that $\lvert I_i \rvert \geqslant \mu \lvert Y_i \rvert$, where μ is a constant that will be determined later.

Collapse layers. When we find that some layer is collapsible, we run the routine Collapse below, which collapses layers in the stack until no layer is collapsible. Collapse works in the same manner as its counterpart in [1], but there are small differences in the presentation.

Collapse$(M, \mathcal{E}, \mathcal{I}, (L_1, \cdots, L_\ell))$

1. Compute a canonical decomposition $\mathcal{I}_1 \cup \mathcal{I}_2 \cup \cdots \mathcal{I}_\ell$ and a canonical solution $\Gamma_1 \cup \Gamma_2 \cup \cdots \Gamma_\ell$ of $G_M(Y_{\leqslant \ell}, I)$. If no layer is collapsible, go to build phase. Otherwise, let L_t be the collapsible layer with the smallest index t.
2. Remove all layers above L_t from the stack. Set $\mathcal{I} := \mathcal{I}_{\leqslant t-1}$.
3. Recall that $source(\Gamma_t) \subseteq Y_t$ by Lemma 3.3. Let $\mathcal{V}$ denote the set of the thin edges in $\mathcal{Y}_t$ that cover $source(\Gamma_t)$. Recall that $\mathcal{Y}_t \subseteq \mathcal{E}$.
   a. Update the maximum matching M by flipping the non-trivial paths in Γ_t, i.e., set $M := M \oplus \Gamma_t^+$. This matches the sources of non-trivial paths in Γ_t while leave their sinks unmatched.
   b. Add to $\mathcal{E}$ edges in I_t, i.e., set $\mathcal{E} := \mathcal{E} \cup \mathcal{I}_t$. Now the sinks of non-trivial paths are satisfied. Also the sources of trivial paths are satisfied by new thin edges.
   c. Now each player in $source(\Gamma_t)$ is satisfied either by a fat resource or a thin edge from $\mathcal{I}_t$. Edges in $\mathcal{V}$ can be safely removed from $\mathcal{E}$. Set $\mathcal{E} := \mathcal{E} \setminus \mathcal{V}$. Consequently, edges in $\mathcal{V}$ no longer block edges in $\mathcal{X}_t$. Set $\mathcal{Y}_t := \mathcal{Y}_t \setminus \mathcal{V}$.
4. If $t \geqslant 2$, we need to update $\mathcal{X}_t$ because the removal of $\mathcal{V}$ from $\mathcal{E}$ (and hence $\mathcal{Y}_t$) may make some edges in $\mathcal{X}_t$ unblocked. For each edge $(p, B) \in \mathcal{X}_t$ that becomes unblocked, perform the following:
   a. Remove (p, B) from $\mathcal{X}_t$.
   b. If $f_M(Y_{\leqslant t-1}, I \cup \{p\}) > f_M(Y_{\leqslant t-1}, I)$, then add (p, B') to $\mathcal{I}$, where B' is an arbitrary minimal subset of B such that $value(B') \geqslant \tau/\lambda$ and B' excludes the resources covered by $\mathcal{E}$.
5. If $t = 1$, step 3 already satisfied the player p_0 in the bottommost layer in the stack, so the algorithm terminates. Otherwise, update $\ell := t$ and go back to step 1.

▶ **Lemma 3.4.** Collapse *maintains invariants 1–7 in Table 1.*

4 Polynomial running time and binary search

Each call of Build and Collapse runs in time polynomial in ℓ, m and n. Lemma 4.1 below is the key to obtaining a bound on ℓ and the total number of calls of Build and Collapse. The proof of Lemma 4.1 is deferred to Section 5.

▶ **Lemma 4.1.** *Assume that the values τ and λ used by the algorithm satisfy the relations $\tau \leqslant \tau^*$ and $\lambda = 6 + \delta$ for an arbitrary constant $\delta \in (0, 1)$. There exists a constant $\mu \in (0, 1)$ dependent on δ such that for any state $(M, \mathcal{E}, \mathcal{I}, (L_1, \ldots, L_\ell))$ of the algorithm, if $|Y_{i+1}| < \sqrt{\mu}|Y_{\leqslant i}|$ for some $i \in [1, \ell - 1]$, then some layer below L_{i+1} must be collapsible.*

Lemma 4.1, immediately implies a logarithmic bound on the maximum number ℓ of layers. Using argument similar to that in [1, Lemmas 4.10 and 4.11], we can show that given a partial allocation, our algorithm can extend it to satisfy one more player in polynomial time. By repeating the algorithm at most n times, we can extend a maximum matching of G to an allocation of value at least τ/λ.

The remaining task is to binary search for τ^*. If we use a value τ that is at most τ^*, the algorithm terminates in polynomial time with an allocation. If we use a value $\tau > \tau^*$, there are two possible outcomes. We may be lucky and always have some collapsible layer below

L_{i+1} whenever $|Y_{i+1}| < \sqrt{\mu}|Y_{\leqslant i}|$ for some $i \in [1, \ell-1]$. In this case, the algorithm returns in polynomial time an allocation of value at least $\tau/\lambda \geqslant \tau^*/\lambda$. The second outcome is that no layer is collapsible at some point, but $|Y_{i+1}| < \sqrt{\mu}|Y_{\leqslant i}|$ for some $i \in [1, \ell-1]$. This can be detected in $O(1)$ time by maintaining $|Y_{i+1}|$ and $|Y_{\leqslant i}|$, which allows us to detect that $\tau > \tau^*$ and halt the algorithm. Since this is the first violation of this property, the running time before halting is polynomial in m and n. The last allocation returned by the algorithm during the binary search has value at least $\tau^*/\lambda = \tau^*/(6+\delta)$. We will see in Section 5.2 that a smaller δ requires a smaller μ and hence a higher running time.

▶ **Theorem 4.2.** *For any fixed constant $\delta \in (0,1)$, there is an algorithm for the restricted max-min fair allocation problem that returns a $(6+\delta)$-approximation in time polynomial in the number of players and the number of resources.*

5 Analysis

We will derive lower and upper bounds for the total value of the thin resources in the stack and show that if Lemma 4.1 does not hold, the lower bound would exceed the upper bound.

5.1 Competing players

To analyze our aggressive greedy strategy for selecting blocked addable thin edges, we need an injective map φ from the players covered by them to players who can access thin resources of high total value. The next result shows that these target players exist.

▶ **Lemma 5.1.** *Let* OPT *be an arbitrary optimal allocation. There exists a maximum matching M^* of G induced by* OPT *such that M^* matches every player who is assigned at least one fat resource in* OPT*. Hence, every player in $\bar{P}_{M^*}$ is assigned only thin resources in* OPT *that are worth a total value of τ or more, assuming that $\tau \leqslant \tau^*$.*

The domain of the injection φ is a subset of $X_{\leqslant \ell}$ and its image is a subset of $\bar{P}_{M^*}$. We call the image of φ the competing players. For any player $q \in X_{\leqslant \ell}$, $\varphi(q)$ has access to thin resources that are worth a total value of τ or more. Our goal is to prove that $\varphi(q)$ is also an addable player when q is added to $\mathcal{X}_{\leqslant \ell}$. Since the algorithm prefers q to $\varphi(q)$, either no addable edge is incident to $\varphi(q)$ or the maximal addable edge identified for $\varphi(q)$ has less value than the edge $e_q \in \mathcal{X}_{\leqslant \ell}$ that covers q. In both cases, more than $\tau - \mathit{value}(e_q)$ worth of thin resources assigned to $\varphi(q)$ in OPT are already in the stack. This will allow us to prove a good lower bound for the total value of the thin resources in the stack.

Lemma 5.2 below states the properties of competing players. We already discussed the usage of Lemma 5.2(i) and (ii). It would be ideal if the domain of φ could cover the entire $X_{\leqslant \ell}$. However, for technical reasons, when Collapse removes a player from $X_{\leqslant \ell}$, we may have to remove two players from the domain of φ in order to maintain the properties of φ. Lemma 5.2(iii) puts a lower bound on the size of the domain of φ. When deriving lower bound for the total value of the thin resources in the stack, the players in $\bar{P}_{M^*}$ that are not competing players and outside $X_{\leqslant \ell} \cup I$ also play a role. Lemma 5.2(iv) will allow us to prove that a large subset of such players are still addable after we finish adding edges to $\mathcal{X}_\ell$ during the construction of layer L_ℓ. Each of these addable players contributes a large worth of thin resources to the stack.

▶ **Lemma 5.2.** *Let M^* be a maximum matching of G induced by some optimal allocation. For any state $(M, \mathcal{E}, \mathcal{I}, (L_1, \ldots, L_\ell))$ of the algorithm, there exists an injection φ such that*

(i) *The domain* D_φ *and image* Im_φ *of* φ *are subsets of* $X_{\leqslant \ell}$ *and* $\bar{P}_{M^*}$*, respectively.*
(ii) *For every player* $p \in \mathrm{D}_\varphi$*, when* p *was added to* X_k *for some* $k \in [2, \ell]$*,* $\varphi(p)$ *was also an addable player at that time.*
(iii) $|\mathrm{D}_\varphi| \geqslant 2|X_{\leqslant \ell}| - \sum_{i=1}^{\ell} z_i$.
(iv) $f_M(\bar{P}_M, (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell}) = |\bar{P}_M|$.

Proof. Our proof is by induction on the chronological order of the build and collapse phases. In the base case, $\ell = 1$, $X_1 = \emptyset$, and $z_1 = 0$. The existence of φ is trivial as its domain $D_\varphi \subseteq X_1 = \emptyset$. So $\mathrm{Im}_\varphi = \emptyset$. Then, (i), (ii) and (iii) are satisfied trivially, and (iv) follows from Claim 2.1(i). We discuss how to update φ during the build and collapse phases.

Build phase. Suppose that Build begins to construct a new layer L_ℓ. X_ℓ is initialized to be empty. The value z_ℓ is computed only at the completion of L_ℓ. However, in this proof, we initialize $z_\ell = 0$, increment z_ℓ whenever we add an edge to X_ℓ, and show the validity of (i)–(iv) inductively.

Since $X_\ell = \emptyset$ and $z_\ell = 0$ initially, properties (i)–(iv) are satisfied by the current φ by inductive assumption.

Step 2 of Build does not change X_ℓ, and so φ needs no update.

Consider step 3 of Build. Suppose that a thin edge incident to player q_1 is added to $\mathcal{X}_\ell$. So q_1 is addable. For clarity, we use X'_ℓ, z'_ℓ, φ', $\mathrm{D}_{\varphi'}$, and $\mathrm{Im}_{\varphi'}$ to denote the updated X_ℓ, z_ℓ, φ, D_φ, and Im_φ, respectively. Clearly, $X'_\ell = X_\ell \cup \{q_1\}$ and $z'_\ell = z_\ell + 1$. We set $\mathrm{D}_{\varphi'} := \mathrm{D}_\varphi \cup \{q_1\}$. For every $p \in \mathrm{D}_{\varphi'} \setminus \{q_1\}$, we set $\varphi'(p) := \varphi(p)$. We set $\varphi'(q_1)$ as follows.

Let Π_1 be an optimal solution of $G_M(Y_{\leqslant \ell-1}, X_{\leqslant \ell} \cup I \cup \{q_1\})$. We have $q_1 \in \mathit{sink}(\Pi_1)$ since otherwise we would have $f_M(Y_{\leqslant \ell-1}, X_{\leqslant \ell} \cup I \cup \{q_1\}) = f_M(Y_{\leqslant \ell-1}, X_{\leqslant \ell} \cup I)$, contradicting the addability of q_1. Similarly, $q_1 \notin X_{\leqslant \ell}$. As $q_1 \in \mathit{sink}(\Pi_1)$, q_1 must be unmatched in $M \oplus \Pi_1^+$, i.e., $q_1 \in \bar{P}_{M \oplus \Pi_1^+}$. Let Π_2 be an optimal solution of $G_{M \oplus \Pi_1^+}(\bar{P}_{M \oplus \Pi_1^+}, (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell})$. We have $|\Pi_2| = f_{M \oplus \Pi_1^+}(\bar{P}_{M \oplus \Pi_1^+}, (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell}) = f_M(\bar{P}_M, (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell})$ by Claim 2.1(ii). Then, inductive assumption gives $|\Pi_2| = |\bar{P}_M| = |\bar{P}_{M \oplus \Pi_1^+}|$ (both M and $M \oplus \Pi_1$ are maximum matchings). So $\bar{P}_{M \oplus \Pi_1^+} = \mathit{source}(\Pi_2)$, implying that there is a path $\pi \in \Pi_2$ originating from q_1. Let $q_2 = \mathit{sink}(\pi)$.

We claim that $q_2 \notin X_{\leqslant \ell}$. If π is a trivial path, the claim is true because $q_2 = q_1 \notin X_{\leqslant \ell}$. Suppose that π is non-trivial. Suppose, for the sake of contradiction, that $q_2 \in X_{\leqslant \ell}$. This allows us to apply Claim 2.4 and use π to convert Π_1 to an equal-sized set of node-disjoint paths from $Y_{\leqslant \ell-1}$ to $X_{\leqslant \ell} \cup I$. But then $f_M(Y_{\leqslant \ell-1}, X_{\leqslant \ell} \cup I) \geqslant |\Pi_1| = f_M(Y_{\leqslant \ell-1}, X_{\leqslant \ell} \cup I \cup \{q_1\}) \geqslant f_M(Y_{\leqslant \ell-1}, X_{\leqslant \ell} \cup I)$. That is, $f_M(Y_{\leqslant \ell-1}, X_{\leqslant \ell} \cup I \cup \{q_1\}) = f_M(Y_{\leqslant \ell-1}, X_{\leqslant \ell} \cup I)$, contradicting the addability of q_1. This proves our claim that $q_2 \notin X_{\leqslant \ell}$.

Observe that $q_2 \in \bar{P}_{M^*} \setminus \mathrm{Im}_\varphi$ because $q_2 \in \mathit{sink}(\Pi_2) \subseteq (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell}$ and $q_2 \notin X_{\leqslant \ell}$. This allows us to set $\varphi'(q_1) := q_2$ and keep φ' injective.

Properties (i) and (iii) are straightforwardly satisfied by φ', z'_ℓ, $\mathrm{D}_{\varphi'}$, and X'_ℓ.

By induction assumption, (ii) holds for players in $\mathrm{D}_{\varphi'} \setminus \{q_1\} = \mathrm{D}_\varphi$. It remains to check the validity of (ii) for $\varphi'(q_1) = q_2$. If π is a trivial path, then (ii) holds because $q_2 = q_1$ and q_1 is addable. Assume that π is non-trivial. By Claim 2.4, we can use π to convert Π_1 to an equal-sized set of node-disjoint paths in G_M from $Y_{\leqslant \ell-1}$ to $X_{\leqslant \ell} \cup I \cup \{q_2\}$. Thus, $f_M(Y_{\leqslant \ell-1}, X_{\leqslant \ell} \cup I \cup \{q_2\}) \geqslant |\Pi_1| = f_M(Y_{\leqslant \ell-1}, X_{\leqslant \ell} \cup I \cup \{q_1\}) = f_M(Y_{\leqslant \ell-1}, X_{\leqslant \ell} \cup I) + 1$ as q_1 is addable. Therefore, q_2 is also an addable player at the time when $\mathcal{X}_\ell$ gains a thin edge incident to q_1.

Consider (iv). If π is a trivial path, i.e., $q_1 = q_2$, then (iv) holds because $(\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell} \subseteq (\bar{P}_{M^*} \setminus (\mathrm{Im}_\varphi \cup \{q_2\})) \cup X_{\leqslant \ell} \cup \{q_1\} = (\bar{P}_{M^*} \setminus \mathrm{Im}_{\varphi'}) \cup X'_{\leqslant \ell}$. Suppose that π is non-trivial. Recall that Π_2 is an optimal solution of $G_{M \oplus \Pi_1^+}(\bar{P}_{M \oplus \Pi_1^+}, \bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell})$,

and $|\Pi_2| = |\bar{P}_{M\oplus\Pi_1^+}|$. Take the maximum matching $M \oplus \Pi_1^+$ of G and flip the paths in $\Pi_2^+ \setminus \{\pi\}$ in G. This produces another maximum matching $M' = (M \oplus \Pi_1^+) \oplus (\Pi_2^+ \setminus \{\pi\})$. All $|\bar{P}_{M\oplus\Pi_1^+}|$ sinks of Π_2, except for q_2, are unmatched in M'. Player q_1 is also unmatched in M'. There are equally many unmatched players in M' and $M \oplus \Pi_1^+$. This implies that $(\mathit{sink}(\Pi_2) \setminus \{q_2\}) \cup \{q_1\}$ is exactly $\bar{P}_{M'}$. Since $\mathit{sink}(\Pi_2) \subseteq (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell}$, we get $\bar{P}_{M'} \subseteq \big(\big((\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_{\leqslant \ell}\big) \setminus \{q_2\}\big) \cup \{q_1\} \subseteq (\bar{P}_{M^*} \setminus (\mathrm{Im}_\varphi \cup \{q_2\})) \cup X_{\leqslant \ell} \cup \{q_1\} = (\bar{P}_{M^*} \setminus \mathrm{Im}_{\varphi'}) \cup X'_{\leqslant \ell}$. Then, we can apply Claim 2.1(i) to obtain $|\bar{P}_M| \geqslant f_M(\bar{P}_M, (\bar{P}_{M^*} \setminus \mathrm{Im}_{\varphi'}) \cup X'_{\leqslant \ell}) \geqslant f_M(\bar{P}_M, \bar{P}_{M'}) = |\bar{P}_M|$. Hence, (iv) holds.

Clearly, steps 4–6 of Build do not affect φ.

Collapse phase. Suppose that we are going to collapse the layer L_t. Since we will set $\ell := t$ at the end of collapsing L_t, we only need to prove (i)—(iv) with ℓ substituted by t.

Clearly, step 1 of Collapse has no effect on φ.

Consider step 2 of Collapse. Go back to the last time when L_t was either created by Build as the topmost layer or made by Collapse as the topmost layer. By the inductive assumption, there was an injection φ'' at that time that satisfies (i)–(iv). We set $\varphi := \varphi''$, $\mathrm{D}_\varphi := \mathrm{D}_{\varphi''}$, and $\mathrm{Im}_\varphi := \mathrm{Im}_{\varphi''}$.

In step 3 of Collapse, the maximum matching M may change, so only (iv) is affected. Nonetheless, by Claim 2.1(ii), the value of $f_M(\bar{P}_M, (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X_\ell)$ remains the same after updating M. So (iv) is satisfied afterwards.

In step 4 of Collapse, we may remove some edges from $\mathcal{X}_t$ and add some of these removed edges to $\mathcal{I}$. Adding edges to $\mathcal{I}$ does not affect φ. We need to update φ when an edge is removed from $\mathcal{X}_t$. Suppose that we are going to remove from $\mathcal{X}_t$ an edge that covers a player q_1. Recall that z_t was defined in the last construction of the layer L_t, and it has remained fixed despite possible changes to $\mathcal{X}_t$ since then. Let $X'_{\leqslant t}$, φ', $\mathrm{D}_{\varphi'}$, and $\mathrm{Im}_{\varphi'}$ denote the updated $X_{\leqslant t}$, φ, D_φ, and Im_φ, respectively. Note that $X'_{\leqslant t} = X_{\leqslant t} \setminus \{q_1\}$. We show how to define φ', $\mathrm{D}_{\varphi'}$, and $\mathrm{Im}_{\varphi'}$ appropriately.

Consider property (iv). If (iv) is not affected by the deletion of q_1, that is, $f_M(\bar{P}_M, (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X'_{\leqslant t}) = |\bar{P}_M|$, then we simply set $\mathrm{D}_{\varphi'} := \mathrm{D}_\varphi \setminus \{q_1\}$ and $\varphi'(p) := \varphi(p)$ for all $p \in \mathrm{D}_{\varphi'}$. It is easy to verify that φ' satisfies (iv). Suppose that property (iv) is affected, and therefore, $f_M(\bar{P}_M, (\bar{P}_{M^*} \setminus \mathrm{Im}_\varphi) \cup X'_{\leqslant t}) = |\bar{P}_M| - 1$. Since $f_M(\bar{P}_M, \bar{P}_{M^*}) = |\bar{P}_M|$, we have that $f_M(\bar{P}_M, \bar{P}_{M^*} \cup X'_{\leqslant t}) = |\bar{P}_M|$. Comparing the two equations above, we conclude that there must a player $q_2 \in \mathrm{D}_\varphi$ such that $f_M(\bar{P}_M, (\bar{P}_{M^*} \setminus (\mathrm{Im}_\varphi \setminus \{\varphi(q_2)\}) \cup X'_{\leqslant t}) = |\bar{P}_M|$. So we set $\mathrm{D}_{\varphi'} := \mathrm{D}_\varphi \setminus \{q_1, q_2\}$, and $\varphi'(p) := \varphi(p)$ for all $p \in \mathrm{D}_{\varphi'}$. Property (iv) is satisfied afterwards.

Irrespective of which definition of φ' above is chosen, properties (i) and (ii) trivially hold. Property (iii) holds because the left hand side decreases by at most 2 and the right hand side decreases by exactly 2. ◀

5.2 Proof of Lemma 4.1

Suppose, for the sake of contradiction, that there exists an index $k \in [1, \ell - 1]$ such that $|Y_{k+1}| < \sqrt{\mu}\,|Y_{\leqslant k}|$ but no layer below L_{k+1} is collapsible. Let k be the smallest such index. So $|Y_{i+1}| \geqslant \sqrt{\mu}\,|Y_{\leqslant i}|$ for every $i \in [1, k-1]$.

Consider the moment immediately after the last construction of the $(k+1)$-th layer. Let $(M', \mathcal{E}', \mathcal{I}', (L'_1, \ldots, L'_{k+1}))$ be the state of the algorithm at that moment. No layer below L'_{k+1} is collapsible immediately after the construction of L'_{k+1} since this is the last construction of the $(k+1)$-th layer. We will derive a few inequalities that hold given the existence of k. Then we will obtain a contradiction by showing that the system made up of these inequalities is infeasible.

We first define some notations. Let $\mathcal{X}'_i$ and $\mathcal{Y}'_i$ denote the set of blocked addable (thin) edges and the set of blocking (thin) edges associated with L'_i. Then, X'_i, Y'_i, $\mathcal{X}'_{\leqslant i}$, $\mathcal{Y}'_{\leqslant i}$, $X'_{\leqslant i}$, and $Y'_{\leqslant i}$ are correspondingly defined. Let M^* be a maximum matching induced by an optimal allocation OPT. Let φ' and $\mathrm{D}_{\varphi'}$ be the injection and its domain associated with $X'_{\leqslant t+1}$ as defined in Lemma 5.2 with respect to M^*. For all $p \in \mathrm{D}_{\varphi'}$, define $w_p := \mathit{value}(B)$, where (p, B) is the thin edge for p in $\mathcal{X}'_{\leqslant k+1}$. By invariant 1 in Table 1, w_p is well defined (as $\mathrm{D}_{\varphi'} \subseteq X'_{\leqslant k+1}$ by Lemma 5.2(i) and no player is covered by two edges in $\mathcal{X}'_{\leqslant k+1}$) and $w_p \in [\tau/\lambda, \tau + \tau/\lambda]$.

By the definition above, we already have two easy inequalities. Recall that given a set $\mathcal{S}$ of thin edges, $\mathit{value}(\mathcal{S})$ is the total value of the thin resources covered by $\mathcal{S}$.

$$\mathit{value}(\mathcal{X}'_{\leqslant k+1} \cup \mathcal{Y}'_{\leqslant k}) \geqslant \mathit{value}(\mathcal{X}'_{\leqslant k+1}) \geqslant \sum_{p \in \mathrm{D}_{\varphi'}} w_p, \quad \frac{\tau}{\lambda}|\mathrm{D}_{\varphi'}| \leqslant \sum_{p \in \mathrm{D}_{\varphi'}} w_p \leqslant (\tau + \frac{\tau}{\lambda})|\mathrm{D}_{\varphi'}|.$$

▶ **Claim 5.3.** $|\mathrm{D}_{\varphi'}| \leqslant |Y'_{\leqslant k}|$.

▶ **Claim 5.4.** $\mathit{value}(\mathcal{X}'_{\leqslant k+1} \cup \mathcal{Y}'_{\leqslant k}) \leqslant \frac{\tau}{\lambda}|\mathrm{D}_{\varphi'}| + \frac{2\tau}{\lambda}|Y'_{\leqslant k}| + \frac{\delta_1\tau}{\lambda}|Y'_{\leqslant k}|$, *where* $\delta_1 = \lambda\mu + 2\mu + 2\sqrt{\mu}$.

▶ **Claim 5.5.** $\mathit{value}(\mathcal{X}'_{\leqslant k+1} \cup \mathcal{Y}'_{\leqslant k}) \geqslant (\tau - \frac{\tau}{\lambda})(|Y'_{\leqslant k}| - |\mathrm{D}_{\varphi'}|) + \sum_{p \in \mathrm{D}_{\varphi'}}(\tau - w_p) - \frac{\delta_2\tau}{\lambda}|Y'_{\leqslant k}|$, *where* $\delta_2 = 2\lambda\mu + 2\lambda\sqrt{\mu} + 6\sqrt{\mu}$.

The proofs of above claims use Lemma 5.2. In particular, Lemma 5.2 plays a key role in the proof of Claim 5.5. Here we give a rough idea. Consider the moment we just finish adding edges to $\mathcal{X}'_{k+1}$. Lemma 5.2(iv) ensures that roughly $(|Y'_{\leqslant k}| - |\mathrm{D}_{\varphi'}|)$ players in $\bar{P}_{M^*} \setminus \mathrm{Im}_{\varphi'}$ are still addable. Since there are no more addable edges (otherwise they will be added to $\mathcal{X}'_{k+1}$), each of these addable players can access less than τ/λ worth of thin resources that are not in the stack, and hence each of them contribute at least $\tau - \tau/\lambda$ worth of thin resources to the stack. This gives the first term $(\tau - \frac{\tau}{\lambda})(|Y'_{\leqslant k}| - |\mathrm{D}_{\varphi'}|)$. For each player $\varphi'(p) \in \mathrm{Im}_{\varphi'}$, as we explained in section 5.1, it contributed at least $\tau - w_p$ worth of thin resources to the stack at the time player p was picked. This gives the second term $\sum_{p \in \mathrm{D}_{\varphi'}}(\tau - w_p)$. The third term is just slack in the analysis.

Putting all the inequalities together gives the following system.

- $\mathit{value}(\mathcal{X}_{\leqslant k+1} \cup \mathcal{Y}_{\leqslant k}) \geqslant \sum_{p \in \mathrm{D}_{\varphi'}} w_p$,
- $\frac{\tau}{\lambda}|\mathrm{D}_{\varphi'}| \leqslant \sum_{p \in \mathrm{D}_{\varphi'}} w_p \leqslant (\tau + \tau/\lambda)|\mathrm{D}_{\varphi'}|$,
- $|\mathrm{D}_{\varphi'}| \leqslant |Y'_{\leqslant k}|$,
- $\mathit{value}(\mathcal{X}'_{\leqslant k+1} \cup \mathcal{Y}'_{\leqslant k}) \leqslant \frac{\tau}{\lambda}|\mathrm{D}_{\varphi'}| + \frac{2\tau}{\lambda}|Y'_{\leqslant k}| + \frac{\delta_1\tau}{\lambda}|Y'_{\leqslant k}|$,
- $\mathit{value}(\mathcal{X}'_{\leqslant k+1} \cup \mathcal{Y}'_{\leqslant k}) \geqslant (\tau - \tau/\lambda)(|Y'_{\leqslant k}| - |\mathrm{D}_{\varphi'}|) + \sum_{p \in \mathrm{D}_{\varphi'}}(\tau - w_p) - \frac{\delta_2\tau}{\lambda}|Y'_{\leqslant k}|$.

Divide the above system by $\frac{\tau}{\lambda}|\mathrm{D}_{\varphi'}|$. To simplify the notation, define the variables $B_1 := \mathit{value}(\mathcal{X}_{\leqslant k+1} \cup \mathcal{Y}_{\leqslant k})/(\frac{\tau}{\lambda}|\mathrm{D}_{\varphi'}|)$, $B_2 := |Y'_{\leqslant k}|/|\mathrm{D}_{\varphi'}|$, and $B_3 := \sum_{p \in \mathrm{D}_{\varphi'}} w_p/(\frac{\tau}{\lambda}|\mathrm{D}_{\varphi'}|)$. Then we can write the above system equivalently as follows.

$$B_1 \geqslant B_3, \quad 1 \leqslant B_3 \leqslant \lambda + 1, \quad 1 \leqslant B_2, \quad B_1 \leqslant 1 + 2B_2 + \delta_1 B_2,$$
$$B_1 \geqslant (\lambda - 1)(B_2 - 1) + \lambda - B_3 - \delta_2 B_2.$$

The first, fourth, and fifth inequalities give $2(1 + 2B_2 + \delta_1 B_2) \geqslant B_1 + B_3 \geqslant (\lambda - 1)(B_2 - 1) + \lambda - \delta_2 B_2 \Rightarrow 2 + (4 + 2\delta_1)B_2 \geqslant (\lambda - 1 - \delta_2)B_2 + 1 \Rightarrow (\lambda - 5 - 2\delta_1 - \delta_2)B_2 \leqslant 1$. On the other hand, $\lambda - 5 - 2\delta_1 - \delta_2 > 1$ for a sufficiently small μ because when μ tends to zero, both δ_1 and δ_2 tend to 0. Hence, $(\lambda - 5 - 2\delta_1 - \delta_2)B_2 > 1$ as $B_2 \geqslant 1$ by the third inequality. But it is impossible that $(\lambda - 5 - 2\delta_1 - \delta_2)B_2 \leqslant 1$ and $(\lambda - 5 - 2\delta_1 - \delta_2)B_2 > 1$ simultaneously.

References

1 Chidambaram Annamalai, Christos Kalaitzis, and Ola Svensson. Combinatorial algorithm for restricted max-min fair allocation. *ACM Trans. Algorithms*, 13(3):37:1–37:28, 2017.

2 Arash Asadpour, Uriel Feige, and Amin Saberi. Santa claus meets hypergraph matchings. *ACM Trans. Algorithms*, 8(3):24:1–24:9, 2012.

3 Arash Asadpour and Amin Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In *Proc. 39th ACM Symposium on Theory of Computing*, pages 114–121, 2007.

4 Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *Proc. 38th ACM Symposium on Theory of Computing*, pages 31–40, 2006.

5 Ivona Bezáková and Varsha Dani. Allocating indivisible goods. *SIGecom Exchanges*, 5(3):11–18, 2005.

6 Deeparnab Chakrabarty, Julia Chuzhoy, and Sanjeev Khanna. On allocating goods to maximize fairness. In *Proc. 50th IEEE Symposium on Foundations of Computer Science*, pages 107–116, 2009.

7 T.-H. Hubert Chan, Zhihao Gavin Tang, and Xiaowei Wu. On (1, epsilon)-restricted max-min fair allocation problem. In *Proc. 27th International Symposium on Algorithms and Computation*, volume 64, pages 23:1–23:13, 2016.

8 Uriel Feige. On allocations that maximize fairness. In *Proc. 19th ACM-SIAM Symposium on Discrete Algorithms*, pages 287–293, 2008.

9 Daniel Golovin. Max-min fair allocation of indivisible good. *Technical report, Carnegie Mellon University*, 2005.

10 Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. New constructive aspects of the lovász local lemma. *Journal of the ACM*, 58(6):28:1–28:28, 2011.

11 László Lovász and Michael D. Plummer. *Matching Theory*. American mathematical society, 2009.

12 Barna Saha and Aravind Srinivasan. A new approximation technique for resource-allocation problems. In *Proc. 1st Symposium on Innovations in Computer Science*, pages 342–357, 2010.

Improved Approximation for Node-Disjoint Paths in Grids with Sources on the Boundary

Julia Chuzhoy[1]
Toyota Technological Institute at Chicago, 6045 S. Kenwood Ave., Chicago, Illinois 60637, USA
cjulia@ttic.edu

David H. K. Kim[2]
Computer Science Department, University of Chicago, 1100 East 58th Street, Chicago, Illinois 60637, USA
hongk@cs.uchicago.edu

Rachit Nimavat[3]
Toyota Technological Institute at Chicago, 6045 S. Kenwood Ave., Chicago, Illinois 60637, USA
nimavat@ttic.edu

Abstract

We study the classical Node-Disjoint Paths (NDP) problem: given an undirected n-vertex graph G, together with a set $\{(s_1,t_1),\ldots,(s_k,t_k)\}$ of pairs of its vertices, called source-destination, or demand pairs, find a maximum-cardinality set $\mathcal{P}$ of mutually node-disjoint paths that connect the demand pairs. The best current approximation for the problem is achieved by a simple greedy $O(\sqrt{n})$-approximation algorithm. Until recently, the best negative result was an $\Omega(\log^{1/2-\epsilon} n)$-hardness of approximation, for any fixed ϵ, under standard complexity assumptions.

A special case of the problem, where the underlying graph is a grid, has been studied extensively. The best current approximation algorithm for this special case achieves an $\tilde{O}(n^{1/4})$-approximation factor. On the negative side, a recent result by the authors shows that NDP is hard to approximate to within factor $2^{\Omega(\sqrt{\log n})}$, even if the underlying graph is a subgraph of a grid, and all source vertices lie on the grid boundary. In a very recent follow-up work, the authors further show that NDP in grid graphs is hard to approximate to within factor $\Omega(2^{\log^{1-\epsilon} n})$ for any constant ϵ under standard complexity assumptions, and to within factor $n^{\Omega(1/(\log\log n)^2)}$ under randomized ETH.

In this paper we study the NDP problem in grid graphs, where all source vertices $\{s_1,\ldots,s_k\}$ appear on the grid boundary. Our main result is an efficient randomized $2^{O(\sqrt{\log n}\cdot\log\log n)}$-approximation algorithm for this problem. Our result in a sense complements the $2^{\Omega(\sqrt{\log n})}$-hardness of approximation for sub-graphs of grids with sources lying on the grid boundary, and should be contrasted with the above-mentioned almost polynomial hardness of approximation of NDP in grid graphs (where the sources and the destinations may lie anywhere in the grid).

Much of the work on approximation algorithms for NDP relies on the multicommodity flow relaxation of the problem, which is known to have an $\Omega(\sqrt{n})$ integrality gap, even in grid graphs, with all source and destination vertices lying on the grid boundary. Our work departs from this paradigm, and uses a (completely different) linear program only to select the pairs to be routed, while the routing itself is computed by other methods.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Node-disjoint paths, approximation algorithms, routing and layout

[1] Supported in part by NSF grants CCF-1318242 and CCF-1616584.
[2] Supported in part by NSF grants CCF-1318242 and CCF-1616584.
[3] Supported in part by NSF grant CCF-1318242.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 38; pp. 38:1–38:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.38

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.09956`.

1 Introduction

We study the classical Node-Disjoint Paths (NDP) problem, where the input consists of an undirected n-vertex graph G and a collection $\mathcal{M} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of pairs of its vertices, called *source-destination* or *demand* pairs. We say that a path P *routes* a demand pair (s_i, t_i) iff the endpoints of P are s_i and t_i. The goal is to compute a maximum-cardinality set $\mathcal{P}$ of node-disjoint paths, where each path $P \in \mathcal{P}$ routes a distinct demand pair in $\mathcal{M}$. We denote by NDP-Planar the special case of the problem when the underlying graph G is planar, and by NDP-Grid the special case where G is a square grid[4]. We refer to the vertices in set $S = \{s_1, \ldots, s_k\}$ as *source vertices*; to the vertices in set $T = \{t_1, \ldots, t_k\}$ as *destination vertices*, and to the vertices in set $S \cup T$ as *terminals*.

NDP is a fundamental graph routing problem that has been studied extensively in both graph theory and theoretical computer science communities. Robertson and Seymour [31, 33] explored the problem in their Graph Minor series, providing an efficient algorithm for NDP when the number k of the demand pairs is bounded by a constant. But when k is a part of input, the problem becomes NP-hard [20, 18], even in planar graphs [27], and even in grid graphs [26]. The best current approximation factor of $O(\sqrt{n})$ for NDP is achieved by a simple greedy algorithm [25]. Until recently, this was also the best approximation algorithm for NDP-Planar and NDP-Grid. A natural way to design approximation algorithms for NDP is via the multicommodity flow relaxation: instead of connecting each routed demand pair with a path, send maximum possible amount of (possibly fractional) flow between them. The optimal solution to this relaxation can be computed via a standard linear program. The $O(\sqrt{n})$-approximation algorithm of [25] can be cast as an LP-rounding algorithm of this relaxation. Unfortunately, it is well-known that the integrality gap of this relaxation is $\Omega(\sqrt{n})$, even when the underlying graph is a grid, with all terminals lying on its boundary. In a recent work, Chuzhoy and Kim [12] designed an $\tilde{O}(n^{1/4})$-approximation for NDP-Grid, thus bypassing this integrality gap barrier. Their main observation is that, if all terminals lie close to the grid boundary (say within distance $O(n^{1/4})$), then a simple dynamic programming-based algorithm yields an $O(n^{1/4})$-approximation. On the other hand, if, for every demand pair, either the source or the destination lies at a distance at least $\Omega(n^{1/4})$ from the grid boundary, then the integrality gap of the multicommodity flow relaxation improves, and one can obtain an $\tilde{O}(n^{1/4})$-approximation via LP-rounding. A natural question is whether the integrality gap improves even further, if all terminals lie further away from the grid boundary. Unfortunately, the authors show in [12] that the integrality gap remains at least $\Omega(n^{1/8})$, even if all terminals lie within distance $\Omega(\sqrt{n})$ from the grid boundary. The $\tilde{O}(n^{1/4})$-approximation algorithm for NDP-Grid was later extended and generalized to an $\tilde{O}(n^{9/19})$-approximation algorithm for NDP-Planar [13].

On the negative side, until recently, only an $\Omega(\log^{1/2-\epsilon} n)$-hardness of approximation was known for the general version of NDP, for any constant ϵ, unless NP $\subseteq$ ZPTIME$(n^{\text{poly}\log n})$ [4, 3], and only APX-hardness was known for NDP-Planar and NDP-Grid [12]. In a recent

[4] We use the standard convention of denoting $n = |V(G)|$, and so the grid has dimensions $(\sqrt{n} \times \sqrt{n})$; we assume that $\sqrt{n}$ is an integer.

work [15], the authors have shown that NDP is hard to approximate to within a $2^{\Omega(\sqrt{\log n})}$ factor unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log n)})$, even if the underlying graph is a planar graph with maximum vertex degree at most 3, and all source vertices lie on the boundary of a single face. The result holds even when the input graph G is a vertex-induced subgraph of a grid, with all sources lying on the grid boundary. In a very recent work [14], the authors show that NDP-Grid is $2^{\Omega(\log^{1-\epsilon} n)}$-hard to approximate for any constant ϵ assuming $\mathsf{NP} \not\subseteq \mathsf{BPTIME}(n^{\mathrm{poly}\log n})$, and moreover, assuming randomized ETH, the hardness of approximation factor becomes $n^{\Omega(1/(\log\log n)^2)}$. We note that the instances constructed in these latter hardness proofs require all terminals to lie far from the grid boundary.

In this paper we explore NDP-Grid. This important special case of NDP was initially motivated by applications in VLSI design, and has received a lot of attention since the 1960's. We focus on a restricted version of NDP-Grid, that we call Restricted NDP-Grid: here, in addition to the graph G being a square grid, we also require that all source vertices $\{s_1, \ldots, s_k\}$ lie on the grid boundary. We do not make any assumptions about the locations of the destination vertices, that may appear anywhere in the grid. The best current approximation algorithm for Restricted NDP-Grid is the same as that for the general NDP-Grid, and achieves a $\tilde{O}(n^{1/4})$-approximation [12]. Our main result is summarized in the following theorem.

▶ **Theorem 1.** *There is an efficient randomized $2^{O(\sqrt{\log n}\cdot\log\log n)}$-approximation algorithm for* Restricted NDP-Grid.

This result in a sense complements the $2^{\Omega(\sqrt{\log n})}$-hardness of approximation of NDP on sub-graphs of grids with all sources lying on the grid boundary of [15][5], and should be contrasted with the recent almost polynomial hardness of approximation of [14] for NDP-Grid mentioned above. Our algorithm departs from previous work on NDP in that it does not use the multicommodity flow relaxation. Instead, we define sufficient conditions that allow us to route a subset $\mathcal{M}'$ of demand pairs via disjoint paths, and show that there exists a subset of demand pairs satisfying these conditions, whose cardinality is at least $\mathsf{OPT}/2^{O(\sqrt{\log n}\cdot\log\log n)}$, where OPT is the value of the optimal solution. It is then enough to compute a maximum-cardinality subset of the demand pairs satisfying these conditions. We write an LP-relaxation for this problem and design a $2^{O(\sqrt{\log n}\cdot\log\log n)}$-approximation LP-rounding algorithm for it. We emphasize that the linear program is only used to select the demand pairs to be routed, and not to compute the routing itself.

We then generalize this result to instances where the source vertices lie within a prescribed distance from the grid boundary.

▶ **Theorem 2.** *For every integer $\delta \geq 1$, there is an efficient randomized $\left(\delta \cdot 2^{O(\sqrt{\log n}\cdot\log\log n)}\right)$-approximation algorithm for the special case of* NDP-Grid *where all source vertices lie within distance at most δ from the grid boundary.*

We note that for instances of NDP-Grid where both the sources and the destinations are within distance at most δ from the grid boundary, it is easy to obtain an efficient $O(\delta)$-approximation algorithm (see, e.g. [12]).

A problem closely related to NDP is the Edge-Disjoint Paths (EDP) problem. It is defined similarly, except that now the paths chosen to route the demand pairs may share vertices,

[5] Note that the two results are not strictly complementary: our algorithm only applies to grid graphs, while the hardness result is only valid for sub-graphs of grids.

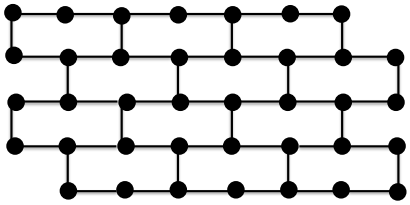


Figure 1 A wall graph.

and are only required to be edge-disjoint. The approximability status of EDP is very similar to that of NDP: there is an $O(\sqrt{n})$-approximation algorithm [10], and an $\Omega(\log^{1/2-\epsilon} n)$-hardness of approximation for any constant ϵ, unless NP $\subseteq$ ZPTIME$(n^{\text{poly}\log n})$ [4, 3]. As in the NDP problem, we can use the standard multicommodity flow LP-relaxation of the problem, in order to obtain the $O(\sqrt{n})$-approximation algorithm, and the integrality gap of the LP-relaxation is $\Omega(\sqrt{n})$ even in planar graphs. Recently, Fleszar et al. [19] designed an $O(\sqrt{r}\cdot\log(kr))$-approximation algorithm for EDP, where r is the feedback vertex set number of the input graph $G=(V,E)$ — the smallest number of vertices that need to be deleted from G in order to turn it into a forest.

Several special cases of EDP have better approximation algorithms: an $O(\log^2 n)$-approximation is known for even-degree planar graphs [9, 8, 22], and an $O(\log n)$-approximation is known for nearly-Eulerian uniformly high-diameter planar graphs, and nearly-Eulerian densely embedded graphs, including grid graphs [5, 24, 23]. Furthermore, an $O(\log n)$-approximation algorithm is known for EDP on 4-edge-connected planar, and Eulerian planar graphs [21]. It appears that the restriction of the graph G to be Eulerian, or near-Eulerian, makes the EDP problem on planar graphs significantly simpler, and in particular improves the integrality gap of the standard multicommodity flow LP-relaxation.

The analogue of the grid graph for the EDP problem is the wall graph (see Figure 1): the integrality gap of the multicommodity flow relaxation for EDP on wall graphs is $\Omega(\sqrt{n})$. The $\tilde{O}(n^{1/4})$-approximation algorithm of [12] for NDP-Grid extends to EDP on wall graphs, and the $2^{\Omega(\sqrt{\log n})}$-hardness of approximation of [15] for NDP-Planar also extends to EDP on sub-graphs of walls, with all sources lying on the top boundary of the wall. The recent hardness result of [14] for NDP-Grid also extends to an $2^{\Omega(\log^{1-\epsilon} n)}$-hardness of EDP on wall graphs, assuming NP $\not\subseteq$ BPTIME$(n^{\text{poly}\log n})$, and to $n^{\Omega(1/(\log\log n)^2)}$-hardness assuming randomized ETH. We extend our results to EDP and NDP on wall graphs:

▶ **Theorem 3.** *There is an efficient randomized $2^{O(\sqrt{\log n}\cdot\log\log n)}$-approximation algorithm for* EDP *and for* NDP *on wall graphs, when all source vertices lie on the wall boundary.*

Other related work

Cutler and Shiloach [17] studied an even more restricted version of NDP-Grid, where all source vertices lie on the top row R^* of the grid, and all destination vertices lie on a single row R' of the grid, far enough from its top and bottom boundaries. They considered three different settings of this special case. In the packed-packed setting, all sources appear consecutively on R^*, and all destinations appear consecutively on R' (but both sets may appear in an arbitrary order). They show a necessary and a sufficient condition for all demand pairs to be routable via node-disjoint paths in this setting. The second setting is the packed-spaced setting. Here, the sources again appear consecutively on R^*, but all destinations are at a distance at least

d from each other. For this setting, the authors show that if $d \geq k$, then all demand pairs can be routed. We note that [12] extended their algorithm to a more general setting, where the destination vertices may appear anywhere in the grid, as long as the distance between any pair of the destination vertices, and any destination vertex and the boundary of the grid, is at least $\Omega(k)$. Robertson and Seymour [32] provided sufficient conditions for the existence of node-disjoint routing of a given set of demand pairs in the more general setting of graphs drawn on surfaces, and they designed an algorithm whose running time is $\text{poly}(n) \cdot f(k)$ for finding the routing, where $f(k)$ is at least exponential in k. Their result implies the existence of the routing in grids, when the destination vertices are sufficiently far from each other and from the grid boundaries, but it does not provide an efficient algorithm to compute such a routing. The third setting studied by Cutler and Shiloach is the spaced-spaced setting, where the distances between every pair of source vertices, and every pair of destination vertices are at least d. The authors note that they could not come up with a better algorithm for this setting, than the one provided for the packed-spaced case. Aggarwal, Kleinberg, and Williamson [1] considered a special case of NDP-Grid, where the set of the demand pairs is a permutation: that is, every vertex of the grid participates in exactly one demand pair. They show that $\Omega(\sqrt{n}/\log n)$ demand pairs are routable in this case via node-disjoint paths. They further show that if all terminals are at a distance at least d from each other, then at least $\Omega(\sqrt{nd}/\log n)$ pairs are routable.

A variation of the NPD and EDP problems, where small congestion is allowed, has been a subject of extensive study, starting with the classical paper of Raghavan and Thompson [29] that introduced the randomized rounding technique. We say that a set $\mathcal{P}$ of paths causes congestion c, if at most c paths share the same vertex or the same edge, for the NDP and the EDP settings respectively. A recent line of work [9, 28, 2, 30, 11, 16, 7, 6] has lead to an $O(\text{poly}\log k)$-approximation for both NDP and EDP problems with congestion 2. For planar graphs, a constant-factor approximation with congestion 2 is known [34].

Organization

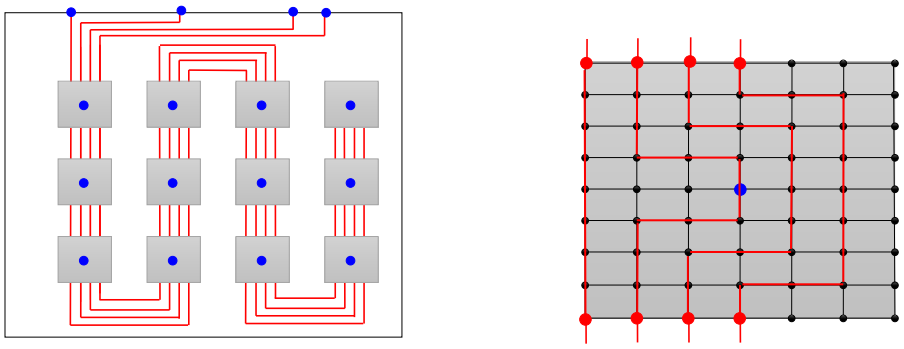
The majority of this extended abstract is dedicated to a detailed but informal overview of the proofs of Theorem 1 and Theorem 2. The formal proofs, as well as the extension to EDP and NDP on wall graphs, are deferred to the full version of the paper.

2 High-Level Overview of the Algorithm

The goal of this section is to provide an informal high-level overview of the main result of the paper – the proof of Theorem 1. With this goal in mind, the values of various parameters are given imprecisely in this section, in a way that best conveys the intuition. The full version of the paper contains a formal description of the algorithm and the precise settings of all parameters.

We first consider an even more restricted special case of NDP-Grid, where all source vertices appear on the top boundary of the grid, and all destination vertices appear far enough from the grid boundary, and design an efficient randomized $2^{O(\sqrt{\log n \cdot \log\log n})}$-approximation algorithm $\mathcal{A}$ for this problem. We later show how to reduce Restricted NDP-Grid to this special case of the problem; we focus on the description of the algorithm $\mathcal{A}$ for now.

We assume that our input graph G is the $(\ell \times \ell)$-grid, and we denote by $n = \ell^2$ the number of its vertices. We further assume that the set of the demand pairs is $\mathcal{M} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, with the vertices in set $S = \{s_1, \ldots, s_k\}$ called source vertices; the vertices in set $T = \{t_1, \ldots, t_k\}$ called destination vertices; and the vertices in $S \cup T$ called



(a) Global routing. In this figure, the sub-grids B_i are aligned vertically and horizontally. A similar (but somewhat more complicated) routing can be performed even if they are not aligned. For convenience we did not include all source vertices and all paths.

(b) Local routing inside B_i

Figure 2 Schematic view of routing of spaced-out instances.

terminals. Let OPT denote the value of the optimal solution to the NDP instance $(G, \mathcal{M})$. We assume that the vertices of S lie on the top boundary of the grid, that we denote by R^*, and the vertices of T lie sufficiently far from the grid boundary – say, at a distance at least OPT from it. For a subset $\mathcal{M}' \subseteq \mathcal{M}$ of the demand pairs, we denote by $S(\mathcal{M}')$ and $T(\mathcal{M}')$ the sets of the source and the destination vertices of the demand pairs in $\mathcal{M}'$, respectively. As our starting point, we consider a simple observation of Chuzhoy and Kim [12], that generalizes the results of Cutler and Shiloach [17]. Suppose we are given an instance of NDP-Grid with k demand pairs, where the sources lie on the top boundary of the grid, and the destination vertices may appear anywhere in the grid, but the distance between every pair of the destination vertices, and every destination vertex and the boundary of the grid, is at least $(8k + 8)$ – we call such instances *spaced-out instances*. In this case, all demand pairs in $\mathcal{M}$ can be efficiently routed via node-disjoint paths, as follows. Consider, for every destination vertex $t_i \in T$, a square sub-grid B_i of G, of size $(2k \times 2k)$, such that t_i lies roughly at the center of B_i. We construct a set $\mathcal{P}$ of k node-disjoint paths, that originate at the vertices of S, and traverse the sub-grids B_i one-by-one in a snake-like fashion (see a schematic view on Figure 2a). We call this part of the routing *global routing*. The *local routing* needs to specify how the paths in $\mathcal{P}$ traverse each box B_i. This is done in a straightforward manner, while ensuring that the unique path originating at vertex s_i visits the vertex t_i (see Figure 2b). By suitably truncating the final set $\mathcal{P}$ of paths, we obtain a routing of all demand pairs in $\mathcal{M}$ via node-disjoint paths.

Unfortunately, in our input instance $(G, \mathcal{M})$, the destination vertices may not be located sufficiently far from each other. We can try to select a large subset $\mathcal{M}' \subseteq \mathcal{M}$ of the demand pairs, so that every pair of destination vertices in $T(\mathcal{M}')$ appear at a distance at least $\Omega(|\mathcal{M}'|)$ from each other; but in some cases the largest such set $\mathcal{M}'$ may only contain $O(\mathsf{OPT}/\sqrt{k})$ demand pairs (for example, suppose all destination vertices lie consecutively on a single row of the grid). One of our main ideas is to generalize this simple algorithm to a number of recursive levels.

For simplicity, let us first describe the algorithm with just two recursive levels. Suppose we partition the top row of the grid into z disjoint intervals, $I_1, \ldots, I_z$. Let $\mathcal{M}' \subseteq \mathcal{M}$ be a set of demand pairs that we would like to route. Denote $|\mathcal{M}'| = k'$, and assume that we are given a collection $\mathcal{Q}$ of square sub-grids of G, of size $(4k' \times 4k')$ each (that we call *squares*),

such that every pair $Q, Q' \in \mathcal{Q}$ of distinct squares is at a distance at least $4k'$ from each other. Assume further that each such sub-grid $Q \in \mathcal{Q}$ is assigned a color $\chi(Q) \in \{c_1, \ldots, c_z\}$, such that, if Q is assigned the color c_j, then all demand pairs $(s,t) \in \mathcal{M}'$ whose destination t lies in Q have their source $s \in I_j$ (so intuitively, each color c_j represents an interval I_j). Let $\mathcal{M}'_j \subseteq \mathcal{M}'$ be the set of all demand pairs $(s,t) \in \mathcal{M}'$ with $s \in I_j$. We would like to ensure that $|\mathcal{M}'_j|$ is roughly k'/z, and that all destination vertices of $T(\mathcal{M}'_j)$ are at a distance at least $|\mathcal{M}'_j|$ from each other. We claim that if we could find the collection $\{I_1, \ldots, I_z\}$ of the intervals of the first row, a collection $\mathcal{Q}$ of sub-grids of G, a coloring $\chi : \mathcal{Q} \to \{c_1, \ldots, c_z\}$, and a subset $\mathcal{M}' \subseteq \mathcal{M}$ of the demand pairs with these properties, then we would be able to route all demand pairs in $\mathcal{M}'$.

In order to do so, for each square $Q \in \mathcal{Q}$, we construct an augmented square Q^+, by adding a margin of k' rows and columns around Q. Our goal is to construct a collection $\mathcal{P}$ of node-disjoint paths routing the demand pairs in $\mathcal{M}'$. We start by constructing a global routing, where all paths in $\mathcal{P}$ originate from the vertices of $S(\mathcal{M}')$ and then visit the squares in $\{Q^+ \mid Q \in \mathcal{Q}\}$ in a snake-like fashion, just like we did for the spaced-out instances described above (see Figure 2a). Consider now some square $Q \in \mathcal{Q}$ and the corresponding augmented square Q^+. Assume that $\chi(Q) = c_j$, and let $\mathcal{P}_j \subseteq \mathcal{P}$ be the set of paths originating at the source vertices that lie in I_j. While traversing the square Q^+, we ensure that only the paths in $\mathcal{P}_j$ enter the square Q; the remaining paths use the margins on the left and on the right of Q in order to traverse Q^+. This can be done because the sources of the paths in $\mathcal{P}_j$ appear consecutively on R^*, relatively to the sources of all paths in $\mathcal{P}$. In order to complete the local routing inside the square Q, observe that the destination vertices appear far enough from each other, and so we can employ the simple algorithm for spaced-out instances inside Q.

In order to optimize the approximation factor that we achieve, we extend this approach to $\rho = O(\sqrt{\log n})$ recursive levels. Let $\eta = 2^{\lceil \sqrt{\log n} \rceil}$. We define auxiliary parameters $d_1 > d_2 > \cdots > d_\rho > d_{\rho+1}$. Roughly speaking, we can think of $d_{\rho+1}$ as being a constant (say 16), of d_1 as being comparable to OPT, and for all $1 \leq h \leq \rho$, $d_{h+1} = d_h/\eta$. The setup for the algorithm consists of three ingredients: (i) a hierarchical decomposition $\tilde{\mathcal{H}}$ of the grid into square sub-grids (that we refer to as squares); (ii) a hierarchical partition $\mathfrak{I}$ of the first row R^* of the grid into intervals; and (iii) a hierarchical coloring f of the squares in $\tilde{\mathcal{H}}$ with colors that correspond to the intervals of $\mathfrak{I}$, together with a selection of a subset $\mathcal{M}' \subseteq \mathcal{M}$ of the demand pairs to route. We define sufficient conditions on the hierarchical system $\tilde{\mathcal{H}}$ of squares, the hierarchical partition $\mathfrak{I}$ of R^* into intervals, the coloring f and the subset $\mathcal{M}'$ of the demand pairs, under which a routing of all pairs in $\mathcal{M}'$ exists and can be found efficiently. For a fixed hierarchical system $\tilde{\mathcal{H}}$ of squares, a triple $(\mathfrak{I}, f, \mathcal{M}')$ satisfying these conditions is called a *good ensemble*. We show that a good ensemble with a large enough set $\mathcal{M}'$ of demand pairs exists, and then design an approximation algorithm for computing a good ensemble maximizing $|\mathcal{M}'|$. We now describe each of these ingredients in turn.

2.1 A Hierarchical System of Squares.

A hierarchical system $\tilde{\mathcal{H}}$ of squares consists of a sequence $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_\rho$ of sets of sub-grids of G. For each $1 \leq h \leq \rho$, $\mathcal{Q}_h$ is a collection of disjoint sub-grids of G (that we refer to as *level-h squares*); every such square $Q \in \mathcal{Q}_h$ has size $(d_h \times d_h)$, and every pair of distinct squares $Q, Q' \in \mathcal{Q}_h$ are within distance at least d_h from each other (see Figure 3). We require that for each $1 < h \leq \rho$, for every square $Q \in \mathcal{Q}_h$, there is a unique square $Q' \in \mathcal{Q}_{h-1}$ (called the *parent-square* of Q) that contains Q. We say that a demand pair (s,t) *belongs* to the hierarchical system $\tilde{\mathcal{H}} = (\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_\rho)$ of squares iff $t \in \bigcup_{Q \in \mathcal{Q}_\rho} Q$. We show a

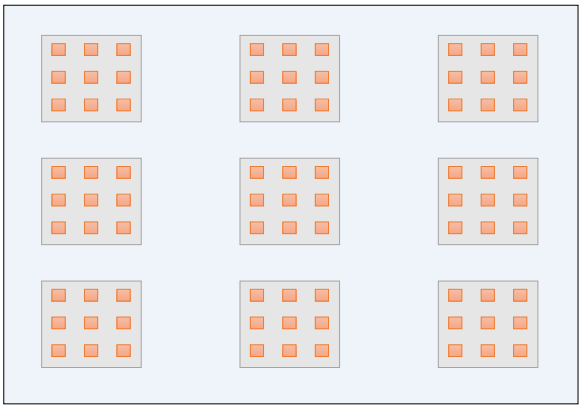


Figure 3 A schematic view of a hierarchical system of squares with 2 levels.

simple efficient algorithm to construct $2^{O(\sqrt{\log n})}$ such hierarchical systems of squares, so that every demand pair belongs to at least one of them. Each such system $\tilde{\mathcal{H}}$ of squares induces an instance of NDP — the instance is defined over the same graph G, and the set $\tilde{\mathcal{M}} \subseteq \mathcal{M}$ of demand pairs that belong to the system $\tilde{\mathcal{H}}$. It is then enough to obtain a factor $2^{O(\sqrt{\log n} \cdot \log\log n)}$-approximation algorithm for each resulting instance $(G, \tilde{\mathcal{M}})$ separately. From now on we fix one such hierarchical system $\tilde{\mathcal{H}} = (\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_\rho)$ of squares, together with the set $\tilde{\mathcal{M}} \subseteq \mathcal{M}$ of demand pairs, containing all pairs (s,t) that belong to $\tilde{\mathcal{H}}$, and focus on designing a $2^{O(\sqrt{\log n} \cdot \log\log n)}$-approximation algorithm for instance $(G, \tilde{\mathcal{M}})$.

2.2 A Hierarchical Partition of the Top Grid Boundary

Recall that R^* denotes the first row of the grid. A hierarchical partition $\mathfrak{I}$ of R^* is a sequence $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_\rho$ of sets of sub-paths of R^*, such that for each $1 \leq h \leq \rho$, the paths in $\mathcal{I}_h$ (that we refer to as *level-h intervals*) partition the vertices of R^*. We also require that for all $1 < h \leq \rho$, every level-h interval $I \in \mathcal{I}_h$ is contained in a unique level-$(h-1)$ interval $I' \in \mathcal{I}_{h-1}$, that we refer to as the *parent-interval* of I. For every level $1 \leq h \leq \rho$, we define a collection χ_h of colors, containing one color $c_h(I)$ for each level-h interval $I \in \mathcal{I}_h$. If $I' \in \mathcal{I}_h$ is a parent-interval of $I \in \mathcal{I}_{h+1}$, then we say that color $c_h(I')$ is a *parent-color* of $c_{h+1}(I)$.

2.3 Coloring the Squares and Selecting Demand Pairs to Route

The third ingredient of our algorithm is an assignment f of colors to the squares, and a selection of a subset of the demand pairs to be routed. For every level $1 \leq h \leq \rho$, for every level-h square $Q \in \mathcal{Q}_h$, we would like to assign a single level-h color $c_h(I) \in \chi_h$ to Q, denoting $f(Q) = c_h(I)$. Intuitively, if color $c_h(I)$ is assigned to Q, then the only demand pairs (s,t) with $t \in Q$ that we may route are those whose source vertex s lies on the level-h interval I. We require that the coloring is consistent across levels: that is, for all $1 < h \leq \rho$, if a level-h square is assigned a level-h color c_h, and its parent-square is assigned a level-$(h-1)$ color c_{h-1}, then c_{h-1} must be a parent-color of c_h. We call such a coloring f a *valid coloring* of $\tilde{\mathcal{H}}$ with respect to $\mathfrak{I}$.

Finally, we would like to select a subset $\mathcal{M}' \subseteq \tilde{\mathcal{M}}$ of the demand pairs to route. Consider some demand pair (s,t) and some level $1 \leq h \leq \rho$. Let I_h be the level-h interval to which s belongs. Then we say that s has the level-h color $c_h(I_h)$. Therefore, for each level $1 \leq h \leq \rho$,

vertex s is assigned the unique level-h color $c_h(I_h)$, and for $1 \leq h < \rho$, $c_h(I_h)$ is the parent-color of $c_{h+1}(I_{h+1})$. Let $Q_\rho \in \mathcal{Q}_\rho$ be the level-ρ square to which t belongs. We may only add (s,t) to $\mathcal{M}'$ if the level-ρ color of Q_ρ is $c_\rho(I_\rho)$ (that is, it is the same as the level-ρ color of s). Notice that in particular, this means that for every level $1 \leq h \leq \rho$, if Q_h is the level-h square containing t, and it is assigned the color $c_h(I_h)$, then s is assigned the same level-h color, and so $s \in I_h$. Finally, we require that for all $1 \leq h \leq \rho$, for every level-h color c_h, the total number of all demand pairs $(s,t) \in \mathcal{M}'$, such that the level-h color of s is c_h, is no more than $d_{h+1}/16$ (if $h = \rho$, then the number is no more than 1). If $\mathcal{M}'$ has all these properties, then we say that it *respects the coloring* f. We say that $(\mathfrak{I}, f, \mathcal{M}')$ is a *good ensemble* iff $\mathfrak{I}$ is a hierarchical partition of R^* into intervals; f is a valid coloring of the squares in $\tilde{\mathcal{H}}$ with respect to $\mathfrak{I}$; and $\mathcal{M}' \subseteq \tilde{\mathcal{M}}$ is a subset of the demand pairs that respects the coloring f. The *size* of the ensemble is $|\mathcal{M}'|$.

2.4 The Routing

We show that, if we are given a good ensemble $(\mathfrak{I}, f, \mathcal{M}')$, then we can route all demand pairs in $\mathcal{M}'$. The routing itself follows the high-level idea outlined above. We gradually construct a collection $\mathcal{P}$ of node-disjoint paths routing the demand pairs in $\mathcal{M}'$. At the highest level, all these paths depart from their sources and then visit the level-1 squares one-by-one, in a snake-like fashion, as in Figure 2a. Consider now some level-1 square Q, and assume that its level-1 color is $c_1(I)$, where $I \in \mathcal{I}_1$ is some level-1 interval of R^*. Then only the paths $P \in \mathcal{P}$ that originate at the vertices of I will enter the square Q; the remaining paths will exploit the spacing between the level-1 squares in order to bypass it; the spacing between the level-1 squares is sufficient to allow this. Once we have defined this global routing, we need to specify how the routing is carried out inside each square. We employ the same procedure recursively. Consider some level-1 square Q, and let $\mathcal{P}' \subseteq \mathcal{P}$ be the set of all paths that visit Q. Assume further that the level-1 color of Q is $c_1(I)$. Since we are only allowed to have at most $d_2/16$ demand pairs in $\mathcal{M}'$ whose level-1 color is $c_1(I)$, $|\mathcal{P}'| \leq d_2/16$. Let $\mathcal{Q}' \subseteq \mathcal{Q}_2$ be the set of all level-2 squares contained in Q. The paths in $\mathcal{P}'$ will visit the squares of $\mathcal{Q}'$ one-by-one in a snake-like fashion (but this part of the routing is performed inside Q). As before, for every level-2 square $Q' \subseteq Q$, if the level-2 color of Q' is $c_2(I')$, then only those paths of $\mathcal{P}'$ that originate at the vertices of I' will enter Q'; the remaining paths will use the spacing between the level-2 squares to bypass Q'. Since $|\mathcal{P}'| \leq d_2/16$, and all level-2 squares are at distance at least d_2 from each other, there is a sufficient spacing to allow this routing. We continue this process recursively, until, at the last level of the recursion, we route at most one path per color, to its destination vertex.

In order to complete the proof of the theorem, we need to show that there exists a good ensemble $(\mathfrak{I}, f, \mathcal{M}')$ of size $|\mathcal{M}'| \geq |\mathsf{OPT}|/2^{O(\sqrt{\log n \cdot \log\log n})}$, and that we can find such an ensemble efficiently.

2.5 The Existence of the Ensemble

The key notion that we use in order to show that a large good ensemble $(\mathfrak{I}, f, \mathcal{M}')$ exists is that of a *shadow property*. Suppose Q is some $(d \times d)$ sub-grid of G, and let $\hat{\mathcal{M}} \subseteq \mathcal{M}$ be some subset of the demand pairs. Among all demand pairs $(s,t) \in \hat{\mathcal{M}}$ with $t \in Q$, let (s_1, t_1) be the one with s_1 appearing earliest on the first row R^* of G, and let (s_2, t_2) be the one with s_2 appearing latest on R^*. The *shadow of* Q *with respect to* $\hat{\mathcal{M}}$ is the sub-path of R^* between s_1 and s_2. Let $N_{\hat{\mathcal{M}}}(Q)$ be the number of all demand pairs $(s,t) \in \hat{\mathcal{M}}$ with s lying in the shadow of Q (that is, s lies between s_1 and s_2 on R^*). We say that $\hat{\mathcal{M}}$ has the

shadow property with respect to Q iff $N_{\hat{\mathcal{M}}}(Q) \leq d$. We say that $\hat{\mathcal{M}}$ has the *shadow property with respect to the hierarchical system* $\tilde{\mathcal{H}} = (\mathcal{Q}_1, \ldots, \mathcal{Q}_\rho)$ *of squares*, iff $\hat{\mathcal{M}}$ has the shadow property with respect to every square in $\bigcup_{h=1}^{\rho} \mathcal{Q}_h$. Let $\mathcal{P}^*$ be the optimal solution to the instance $(G, \tilde{\mathcal{M}})$ of NDP, where $\tilde{\mathcal{M}}$ only includes the demand pairs that belong to $\tilde{\mathcal{H}}$. Let $\mathcal{M}^* \subseteq \tilde{\mathcal{M}}$ be the set of the demand pairs routed by $\mathcal{P}^*$. For every demand pair $(s,t) \in \mathcal{M}^*$, let $P(s,t) \in \mathcal{P}^*$ be the path routing this demand pair. Intuitively, it feels like $\mathcal{M}^*$ should have the shadow property. Indeed, let $Q \in \bigcup_{h=1}^{\rho} \mathcal{Q}_h$ be some square of size $(d_h \times d_h)$, and let $(s_1,t_1), (s_2,t_2) \in \mathcal{M}^*$ be defined for Q as before, so that the shadow of Q with respect to $\mathcal{M}^*$ is the sub-path of R^* between s_1 and s_2. Let P be any path of length at most $2d_h$ connecting t_1 to t_2 in Q, and let γ be the closed curve consisting of the union of $P(s_1,t_1)$, P, $P(s_2,t_2)$, and the shadow of Q. Consider the disc D whose boundary is γ. The intuition is that, if $(s,t) \in \mathcal{M}^*$ is a demand pair whose source lies in the shadow of Q, and destination lies outside of D, then $P(s,t)$ must cross the path P, as it needs to escape the disc D. Since path P is relatively short, only a small number of such demand pairs may exist. The main difficulty with this argument is that we may have a large number of demand pairs (s,t), whose source lies in the shadow of Q, and the destination lies in the disc D. Intuitively, this can only happen if $P(s_1,t_1)$ and $P(s_2,t_2)$ "capture" a large area of the grid. We show that, in a sense, this cannot happen too often, and that there is a subset $\mathcal{M}^{**} \subseteq \mathcal{M}^*$ of at least $|\mathcal{M}^*|/2^{O(\sqrt{\log n} \cdot \log\log n)}$ demand pairs, such that $\mathcal{M}^{**}$ has the shadow property with respect to $\tilde{\mathcal{H}}$.

Finally, we show that there exists a good ensemble $(\mathfrak{I}, f, \mathcal{M}')$ with $|\mathcal{M}'| \geq |\mathcal{M}^{**}|/2^{O(\sqrt{\log n} \cdot \log\log n)}$. We construct the ensemble over the course of ρ iterations, starting with $\mathcal{M}' = \mathcal{M}^{**}$. In the hth iteration we construct the set $\mathcal{I}_h$ of the level-h intervals of R^*, assign level-h colors to all level-h squares of $\tilde{\mathcal{H}}$, and discard some demand pairs from $\mathcal{M}'$. Recall that $\eta = 2^{\lceil \sqrt{\log n} \rceil}$. In the first iteration, we let $\mathcal{I}_1$ be a partition of the row R^* into intervals, each of which contains roughly $\frac{d_1}{16\eta} = \frac{d_2}{16} \leq \frac{|\mathcal{M}^*|}{\eta}$ vertices of $S(\mathcal{M}')$. Assume that these intervals are $I_1, \ldots, I_r$, and that they appear in this left-to-right order on R^*. We call all intervals I_j where j is odd *interesting intervals*, and the remaining intervals I_j *uninteresting intervals*. We discard from $\mathcal{M}'$ all demand pairs (s,t), where s lies on an uninteresting interval. Consider now some level-1 square Q, and let $\mathcal{M}(Q) \subseteq \mathcal{M}'$ be the set of all demand pairs whose destination lies in Q. Since the original set $\mathcal{M}^{**}$ of demand pairs had the shadow property with respect to Q, it is easy to verify that all source vertices of the demand pairs in $\mathcal{M}(Q)$ must belong to a single interesting interval of $\mathcal{I}_1$. Let I be that interval. Then we color the square Q with the level-1 color $c_1(I)$ corresponding to the interval I. This completes the first iteration. Notice that for each level-1 color $c_1(I)$, at most $d_2/16$ demand pairs $(s,t) \in \mathcal{M}'$ have $s \in I$. In the following iteration, we similarly partition every interesting level-1 interval into level-2 intervals that contain roughly $d_3/16 \leq |\mathcal{M}^*|/\eta^2$ source vertices of $\mathcal{M}'$ each, and then define a coloring of all level-2 squares similarly, while suitably updating the set $\mathcal{M}'$ of the demand pairs. We continue this process for ρ iterations, eventually obtaining a good ensemble $(\mathfrak{I}, f, \mathcal{M}')$. Since we only discard a constant fraction of the demand pairs of $\mathcal{M}'$ in every iteration, at the end, $|\mathcal{M}'| \geq |\mathcal{M}^{**}|/2^{\rho} = |\mathcal{M}^{**}|/2^{O(\sqrt{\log n})} \geq |\mathcal{M}^*|/2^{O(\sqrt{\log n} \cdot \log\log n)}$.

2.6 Finding the Good Ensemble

In our final step, our goal is to find a good ensemble $(\mathfrak{I}, f, \mathcal{M}')$ maximizing $|\mathcal{M}'|$. We show an efficient randomized $2^{O(\sqrt{\log n} \cdot \log\log n)}$-approximation algorithm for this problem. First, we show that, at the cost of losing a small factor in the approximation ratio, we can restrict our attention to a small collection $\mathfrak{I}_1, \mathfrak{I}_2, \ldots, \mathfrak{I}_z$ of hierarchical partitions of R^* into intervals,

and that it is enough to obtain a $2^{O(\sqrt{\log n}\cdot\log\log n)}$-approximate solution for the problem of finding the largest ensemble $(\mathfrak{I}_j, f, \mathcal{M}')$ for each such partition $\mathfrak{I}_j$ separately.

We then fix one such hierarchical partition $\mathfrak{I}_j$, and design an LP-relaxation for the problem of computing a coloring f of $\tilde{\mathcal{H}}$ and a collection $\mathcal{M}'$ of demand pairs, such that $(\mathfrak{I}_j, f, \mathcal{M}')$ is a good ensemble, while maximizing $|\mathcal{M}'|$. Finally, we design an efficient randomized LP-rounding $2^{O(\sqrt{\log n}\cdot\log\log n)}$-approximation algorithm for the problem.

2.7 Completing the Proof of Theorem 1

So far we have assumed that all source vertices lie on the top boundary of the grid, and all destination vertices are at a distance at least $\Omega(\mathsf{OPT})$ from the grid boundary. Let $\mathcal{A}$ be the randomized efficient $2^{O(\sqrt{\log n}\cdot\log\log n)}$-approximation algorithm for this special case. We now extend it to the general Restricted NDP-Grid problem. For every destination vertex t, we identify the closest vertex $\tilde{t}$ that lies on the grid boundary. Using standard grouping techniques, at the cost of losing an additional $O(\log n)$ factor in the approximation ratio, we can assume that all source vertices lie on the top boundary of the grid, all vertices in $\left\{\tilde{t} \mid t \in T(\mathcal{M})\right\}$ lie on a single boundary edge of the grid (assume for simplicity that it is the bottom boundary), and that there is some integer d, such that for every destination vertex $t \in T(\mathcal{M})$, $d \leq d(t,\tilde{t}) < 2d$. We show that we can define a collection $\mathcal{Z} = \{Z_1, \ldots, Z_r\}$ of disjoint square sub-grids of G, and a collection $\mathcal{I} = \{I_1, \ldots, I_r\}$ of disjoint sub-intervals of R^*, such that the bottom boundary of each sub-grid Z_i is contained in the bottom boundary of G, the top boundary of Z_i is within distance at least OPT from R^*, $Z_1, \ldots, Z_r$ appear in this left-to-right order in G, and $I_1, \ldots, I_r$ appear in this left-to-right order on R^*. For each $1 \leq j \leq r$, we let $\mathcal{M}_j$ denote the set of all demand pairs with the sources lying on I_j and the destinations lying in Z_j. For each $1 \leq j \leq r$, we then obtain a new instance $(G, \mathcal{M}_j)$ of the NDP problem. We show that there exist a collection $\mathcal{Z}$ of squares and a collection $\mathcal{I}$ of intervals, such that the value of the optimal solution to each instance $(G, \mathcal{M}_j)$, that we denote by OPT_j, is at most d, while $\sum_{j=1}^{r} \mathsf{OPT}_j \geq \mathsf{OPT}/2^{O(\sqrt{\log n}\cdot\log\log n)}$. Moreover, it is not hard to show that, if we can compute, for each $1 \leq j \leq r$, a routing of some subset $\mathcal{M}'_j \subseteq \mathcal{M}_j$ of demand pairs in G, then we can also route all demand pairs in $\bigcup_{j=1}^{r} \mathcal{M}'_j$ simultaneously in G.

There are two problems with this approach. First, we do not know the set $\mathcal{Z}$ of sub-grids of G and the set $\mathcal{I}$ of intervals of R^*. Second, it is not clear how to solve each resulting problem $(G, \mathcal{M}_j)$. To address the latter problem, we define a simple mapping of all source vertices in $S(\mathcal{M}_j)$ to the top boundary of grid Z_j, obtaining an instance of Restricted NDP-Grid, where all source vertices lie on the top boundary of the grid Z_j, and all destination vertices lie at a distance at least $\mathsf{OPT}_j \leq d$ from its boundary. We can then use algorithm $\mathcal{A}$ in order to solve this problem efficiently. It is easy to see that, if we can route some subset $\mathcal{M}'_j$ of the demand pairs via node-disjoint paths in Z_j, then we can extend this routing to the corresponding set of original demand pairs, whose sources lie on R^*.

Finally, we employ dynamic programming in order to find the set $\mathcal{Z}$ of sub-grids of G and the set $\mathcal{I}$ of intervals of I. For each such potential sub-grid Z and interval I, we use algorithm $\mathcal{A}$ in order to find a routing of a large set of demand pairs of the corresponding instance defined inside Z, and then exploit the resulting solution values for each such pair (I, Z) in a simple dynamic program, that allows us to compute the set $\mathcal{Z}$ of sub-grids of G, the set $\mathcal{I}$ of intervals of I, and the final routing.

3 Approximation Algorithm for the Special Case with Sources Close to the Grid Boundary

In this section we provide a sketch of the proof of Theorem 2. We assume that we are given an instance $(G, \mathcal{M})$ of NDP-Grid and an integer $\delta > 0$, such that every source vertex is at a distance at most δ from the grid boundary. Our goal is to design an efficient randomized factor-$(\delta \cdot 2^{O(\sqrt{\log n} \cdot \log\log n)})$-approximation algorithm for this special case. For every terminal $v \in S(\mathcal{M}) \cup T(\mathcal{M})$, let $\tilde{v}$ be the vertex lying closest to v on the boundary of the grid G. Using standard grouping techniques, at the cost of losing an $O(\log n)$-factor in the approximation ratio, we can assume that there is some integer d, such that for all $t \in T(\mathcal{M})$, $d \leq d(t, \tilde{t}) < 2d$.

Assume first that $d \leq \delta \cdot 2^{O(\sqrt{\log n} \log\log n)}$. Let $\hat{\mathcal{M}} = \left\{(\tilde{s}, \tilde{t}) \mid (s,t) \in \mathcal{M}\right\}$ be a new set of demand pairs, so that all vertices participating in these demand pairs lie on the boundary of G. We can efficiently find an optimal solution to the NDP problem instance $(G, \hat{\mathcal{M}})$ using standard dynamic programming. We then show that $\mathsf{OPT}(G, \hat{\mathcal{M}}) = \Omega(\mathsf{OPT}(G, \mathcal{M})/(\delta \cdot 2^{O(\sqrt{\log n} \cdot \log\log n)}))$, obtaining an $(\delta \cdot 2^{O(\sqrt{\log n} \cdot \log\log n)})$-approximation algorithm.

From now on we assume that $d > \delta \cdot 2^{\Omega(\sqrt{\log n} \log\log n)}$. Next, we define a new set $\tilde{\mathcal{M}}$ of demand pairs: $\tilde{\mathcal{M}} = \{(\tilde{s}, t) \mid (s,t) \in \mathcal{M}\}$, so all source vertices of the demand pairs in $\tilde{\mathcal{M}}$ lie on the boundary of G, obtaining an instance of Restricted NDP-Grid. Let OPT' be the value of the optimal solution to problem $(G, \tilde{\mathcal{M}})$. We show that $\mathsf{OPT}' \geq \Omega(\mathsf{OPT}(G, \mathcal{M})/\delta)$.

We then focus on instance $(G, \tilde{\mathcal{M}})$ of Restricted NDP-Grid. We say that a path P routing a demand pair $(\tilde{s}, t) \in \tilde{\mathcal{M}}$ is *canonical* iff it contains the original source s. The crux of the proof is to show that we can modify the routing produced by the $2^{O(\sqrt{\log n} \cdot \log\log n)}$-approximation algorithm to instance $(G, \tilde{\mathcal{M}})$, so that in the resulting routing all paths are canonical. In order to do so, we utilize the fact that the destination vertices lie much further from the grid boundaries than the source vertices. This creates sufficient margins around the grid boundaries that allow us to modify the routing to turn it a canonical one.

References

1. Alok Aggarwal, Jon Kleinberg, and David P. Williamson. Node-disjoint paths on the mesh and a new trade-off in VLSI layout. *SIAM J. Comput.*, 29(4):1321–1333, 2000. doi:10.1137/S0097539796312733.
2. Matthew Andrews. Approximation algorithms for the edge-disjoint paths problem via Raecke decompositions. In *Proceedings of IEEE FOCS*, pages 277–286, 2010. doi:10.1109/FOCS.2010.33.
3. Matthew Andrews, Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, Kunal Talwar, and Lisa Zhang. Inapproximability of edge-disjoint paths and low congestion routing on undirected graphs. *Combinatorica*, 30(5):485–520, 2010. doi:10.1007/s00493-010-2455-9.
4. Matthew Andrews and Lisa Zhang. Logarithmic hardness of the undirected edge-disjoint paths problem. *J. ACM*, 53(5):745–761, sep 2006. doi:10.1145/1183907.1183910.
5. Yonatan Aumann and Yuval Rabani. Improved bounds for all optical routing. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, SODA '95, pages 567–576, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=313651.313820.
6. Chandra Chekuri and Julia Chuzhoy. Half-integral all-or-nothing flow. Unpublished Manuscript.
7. Chandra Chekuri and Alina Ene. Poly-logarithmic approximation for maximum node disjoint paths with constant congestion. In *Proc. of ACM-SIAM SODA*, 2013.

8 Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. Edge-disjoint paths in planar graphs. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 71–80. IEEE, 2004.

9 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. Multicommodity flow, well-linked terminals, and routing problems. In *Proc. of ACM STOC*, pages 183–192, 2005.

10 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. An $O(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow. *Theory of Computing*, 2(1):137–146, 2006. doi:10.4086/toc.2006.v002a007.

11 Julia Chuzhoy. Routing in undirected graphs with constant congestion. *SIAM J. Comput.*, 45(4):1490–1532, 2016. doi:10.1137/130910464.

12 Julia Chuzhoy and David H. K. Kim. On approximating node-disjoint paths in grids. In Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, volume 40 of *LIPIcs*, pages 187–211. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. URL: http://www.dagstuhl.de/dagpub/978-3-939897-89-7, doi:10.4230/LIPIcs.APPROX-RANDOM.2015.187.

13 Julia Chuzhoy, David H. K. Kim, and Shi Li. Improved approximation for node-disjoint paths in planar graphs. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2016, pages 556–569, New York, NY, USA, 2016. ACM. doi:10.1145/2897518.2897538.

14 Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat. Almost polynomial hardness of node-disjoint paths in grids. Unpublished Manuscript, 2017.

15 Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat. New hardness results for routing on disjoint paths. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 86–99. ACM, 2017. doi:10.1145/3055399.3055411.

16 Julia Chuzhoy and Shi Li. A polylogarithmic approximation algorithm for edge-disjoint paths with congestion 2. *J. ACM*, 63(5):45:1–45:51, 2016. URL: http://dl.acm.org/citation.cfm?id=2893472, doi:10.1145/2893472.

17 M. Cutler and Y. Shiloach. Permutation layout. *Networks*, 8:253–278, 1978. doi:10.1002/net.3230080308.

18 Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976. doi:10.1137/0205048.

19 Krzysztof Fleszar, Matthias Mnich, and Joachim Spoerhase. New Algorithms for Maximum Disjoint Paths Based on Tree-Likeness. In Piotr Sankowski and Christos Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:17, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2016.42.

20 R. Karp. On the complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

21 Ken-Ichi Kawarabayashi and Yusuke Kobayashi. An O(log n)-approximation algorithm for the edge-disjoint paths problem in Eulerian planar graphs. *ACM Trans. Algorithms*, 9(2):16:1–16:13, 2013. doi:10.1145/2438645.2438648.

22 Jon Kleinberg. An approximation algorithm for the disjoint paths problem in even-degree planar graphs. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '05, pages 627–636, Washington, DC, USA, 2005. IEEE Computer Society. doi:10.1109/SFCS.2005.18.

23 Jon M. Kleinberg and Éva Tardos. Disjoint paths in densely embedded graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 52–61, 1995.

24 Jon M. Kleinberg and Éva Tardos. Approximations for the disjoint paths problem in high-diameter planar networks. *J. Comput. Syst. Sci.*, 57(1):61–73, 1998. doi:10.1006/jcss.1998.1579.

25 Stavros G. Kolliopoulos and Clifford Stein. Approximating disjoint-path problems using packing integer programs. *Mathematical Programming*, 99:63–87, 2004. doi:10.1007/s10107-002-0370-6.

26 MR Kramer and Jan van Leeuwen. The complexity of wire-routing and finding minimum area layouts for arbitrary vlsi circuits. *Advances in computing research*, 2:129–146, 1984.

27 James F. Lynch. The equivalence of theorem proving and the interconnection problem. *SIGDA Newsl.*, 5(3):31–36, 1975. doi:10.1145/1061425.1061430.

28 Harald Räcke. Minimizing congestion in general networks. In *Proc. of IEEE FOCS*, pages 43–52, 2002.

29 Prabhakar Raghavan and Clark D. Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, December 1987. doi:10.1007/BF02579324.

30 Satish Rao and Shuheng Zhou. Edge disjoint paths in moderately connected graphs. *SIAM J. Comput.*, 39(5):1856–1887, 2010. doi:10.1137/080715093.

31 N. Robertson and P. D. Seymour. Outline of a disjoint paths algorithm. In *Paths, Flows and VLSI-Layout*. Springer-Verlag, 1990.

32 Neil Robertson and Paul D. Seymour. Graph minors. VII. disjoint paths on a surface. *J. Comb. Theory, Ser. B*, 45(2):212–254, 1988. doi:10.1016/0095-8956(88)90070-6.

33 Neil Robertson and Paul D Seymour. Graph minors. XIII. the disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.

34 Loïc Seguin-Charbonneau and F. Bruce Shepherd. Maximum edge-disjoint paths in planar graphs with congestion 2. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 200–209, Washington, DC, USA, 2011. IEEE Computer Society. doi:10.1109/FOCS.2011.30.

Optimal Hashing in External Memory

Alex Conway

Rutgers University, New Brunswick, NJ, USA
alexander.conway@rutgers.edu

Martín Farach-Colton

Rutgers University, New Brunswick, NJ, USA
farach@rutgers.edu

Philip Shilane

Dell EMC, Newtown, PA, USA
shilane@dell.com

Abstract

Hash tables are a ubiquitous class of dictionary data structures. However, standard hash table implementations do not translate well into the external memory model, because they do not incorporate locality for insertions.

Iacono and Pătraşu established an update/query tradeoff curve for external-hash tables: a hash table that performs insertions in $O(\lambda/B)$ amortized IOs requires $\Omega(\log_\lambda N)$ expected IOs for queries, where N is the number of items that can be stored in the data structure, B is the size of a memory transfer, M is the size of memory, and λ is a tuning parameter. They provide a complicated hashing data structure, which we call the **IP hash table**, that meets this curve for λ that is $\Omega(\log\log M + \log_M N)$.

In this paper, we present a simpler external-memory hash table, the **Bundle of Arrays Hash Table** (BOA), that is optimal for a narrower range of λ. The simplicity of BOAs allows them to be readily modified to achieve the following results:

- A new external-memory data structure, the **Bundle of Trees Hash Table** (BOT), that matches the performance of the IP hash table, while retaining some of the simplicity of the BOAs.
- The **Cache-Oblivious Bundle of Trees Hash Table** (COBOT), the first cache-oblivious hash table. This data structure matches the optimality of BOTs and IP hash tables over the same range of λ.

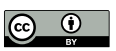
2012 ACM Subject Classification Theory of computation → Sorting and searching

Keywords and phrases hash tables, external memory algorthims, cache-oblivious algorithms, asymmetric data structures

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.39

Related Version A full version of the paper is available at [**?**], `http://arxiv.org/abs/1805.09423`.

Funding Supported by NSF CCF 1637458, NIH 1 U01 CA198952-01, a NetAPP Faculty Fellowship and a gift from Dell/EMC.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 39; pp. 39:1–39:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Dictionaries are among the most heavily used data structures. A dictionary maintains a collection of key-value pairs $\mathcal{S} \subseteq \mathcal{U} \times \mathcal{V}$, under operations[1] INSERT$(x, v, \mathcal{S})$, DELETE$(x, \mathcal{S})$, and QUERY$(x, \mathcal{S})$, which returns the value corresponding to x when $x \in S$. When data fits in memory, there are many solutions to the dictionary problem.

When data is too large to fit in memory, comparison-based dictionaries can be quite varied. They include the B$^\varepsilon$-tree [8], the write-optimized skip list [6], and the cache-optimized look-ahead array (COLA) [2,3,5]. These are optimal in the **external-memory comparison model** in that they match the bound established by Brodal and Fagerberg [8] who showed that for any dictionary in this model, if insertions can be performed in $O\left(\frac{\lambda \log_\lambda N}{B}\right)$ amortized IOs, then there exists a query that requires at least $\Omega(\log_\lambda N)$ IOs, where N is the number of items that can be stored in the data structure, B is the size of a memory transfer, and λ is a tuning parameter. In the following M will be the size of memory, and $B = \Omega(\log n)$. This trade off has since been extended in several ways [1, 4].

Iacono and Pătraşcu showed that in the DAM model, in which operations beyond comparisons are allowed on keys, that a better tradeoff exists:

▶ **Theorem 1** ([11]). *If insertions into an external memory dictionary can be performed in $O(\lambda/B)$ amortized IOs, then queries require an expected $\Omega(\log_\lambda N)$ IOs.*

They further describe an external-memory hashing algorithm, which we refer to here as the **IP hash table**, that performs insertions in $O\left(\frac{1}{B}\left(\lambda + \log_{\frac{M}{B}} N + \log\log N\right)\right)$ IOs and queries in $O(\log_\lambda N)$ IOs w.h.p. Therefore, for $\lambda = \Omega\left(\log_{M/B} N + \log\log N\right)$, the IP hash table meets the tradeoff curve of Theorem 1 and is thus optimal.

In dictionaries that do not support successors and predecessors, we can assume that keys are hashed, that is, that they are uniformly distributed and satisfy some independence properties. The IP hash table and the following results hash all keys before insertion and query in the dictionary by a $\Theta(\log N)$-independent hash function.

The base result of this paper is a simple external-memory hashing scheme, the **Bundle of Arrays Hash Table** (BOA), that meets the optimal Theorem 1 trade off curve for large enough λ. Specifically, we show:

▶ **Theorem 2.** *A BOA supports N insertions and deletions with amortized per entry cost of $O\left(\left(\lambda + \log_{\frac{M}{B}} N + \log_\lambda N\right)/B\right)$ IOs, for any $\lambda > 1$. A query for a key K costs $O(D_K \log_\lambda N)$ IOs w.h.p., where D_K is the number of times K has been inserted or deleted.*

Thus BOAs are optimal for $\lambda = \Omega(\log_{\frac{M}{B}} N + \log_\lambda N)$. They are readily modified to provide several variations, notably the **Bundle of Trees Hash Table** (BOT). BOTs are optimal for the same range of λ as the IP hash table:

▶ **Theorem 3.** *A BOT supports N insertions and deletions with amortized per entry cost of $O\left(\left(\lambda + \log_{\frac{M}{B}} N + \log\log M\right)/B\right)$ IOs for any $\lambda > 1$. A query for a key K costs $O(D_K \log_\lambda N)$ IOs w.h.p., where D_K is the number of times K has been inserted or deleted.*

[1] We do not consider dictionaries that also support the SUCC$(x, \mathcal{S})$ and PRED$(x, \mathcal{S})$. SUCC$(x, \mathcal{S})$ return $\min\{y | y > x \wedge y \in \mathcal{S}\}$ and PRED$(x, \mathcal{S})$ is defined symmetrically.

We further introduce the first cache-oblivious hash table, the **Cache-Oblivious Bundle of Trees Hash Table** (COBOT), which matches the IO performance of BOTs and IP hash tables.

The BOT can also be adapted to models in which disk reads and writes incur different costs. The β-asymmetric BOT adjusts the merging schedule of a regular BOT to trade some writes for more reads. We leave the details for the full version [**?**].

▶ **Theorem 4.** *A β-asymmetric BOT supports N insertions and deletions with amortized per entry cost of $O\left(\frac{1}{B}\left(\lambda + \frac{1}{\beta}\log_\lambda N\right)\right)$ writes and $O\left(\frac{1}{B}(\lambda+\beta)\right)$ reads for any $\lambda > 1$ and $\beta \leq \left\lfloor \log_\lambda \frac{M}{B \log_\lambda N} \right\rfloor$. A query for a key K performs $O(D_K \log_\lambda N)$ reads, where D_K is the number of times K has been inserted or deleted.*

2 Preliminaries

Fingerprints and Hashing. In order to achieve our bounds, we need $\Theta(\log N)$-wise independent hash functions, which, once again matches IP hash tables. We note that a k-wise independent hash function is also k-wise independent on individual bits. Furthermore, the following Chernoff-type bound holds:

▶ **Lemma 5** ([14]). *Let $X_1, X_2, \ldots, X_N$ be $\lceil \mu\delta \rceil$-wise independent binary random variables, $X = \sum_{i=1}^{N} X_i$ and $\mu = \mathrm{E}[X]$. Then $\mathrm{P}(X > \mu\delta) = O\left(1/\delta^{\mu\delta}\right)$, for sufficiently large δ.*

In the following, we use **fingerprint** to refer to any key that has been hashed using a $\Theta(\log N)$-wise independent hash function. Such hash functions have a compact representation and can be specified using $\Theta(\log N)$ words. The universe that is hashed into is assumed to have size $\Theta(N^k)$ for $k \geq 2$. We ignore collisions, but these can be handled as in [12].

For a fingerprint K, it will be convenient to interpret the bits of K as a string of $\log \lambda$ (where lambda is a given parameter) bit characters, $K = K_0 K_1 K_2 \cdots$.

Delta Encoding. We will frequently encounter sorted lists of fingerprint prefixes (possibly with duplicates), together with some data about each. When the size of the list is dense in the space of prefixes, we can compress it using **delta encoding**, where the difference between prefixes is stored rather than the prefixes themselves.

▶ **Lemma 6.** *A list of delta-encoded prefixes with density D, that is there are D prefixes in the list for every possible prefix, requires $O(-\log_\lambda D)$ characters per prefix.*

Proof. The average difference between consecutive prefixes is $1/D$. Because logarithms are convex, the average number of characters required to represent this difference is therefore $O(-\log_\lambda D)$. ◀

Log-structured Merge Trees. Log-structured merge trees (LSMs) are (a family of) external-memory dictionary data structures. They come in two varieties: **level-tiered LSMs** (LT-LSMs) and **size-tiered LSMs** (ST-LSMs). Both kinds are suboptimal in that they do not meet the optimal insertion-query tradeoff [8], although the COLA [3] is an optimal variant of the LT-LSMs.

An LSM consists of sets of either B-trees or sorted arrays called **runs**. In this paper, we describe them in terms of runs, since we use runs below.

An LT-LSM consists of a cascade of levels, where each level consists of at most one run. Each level has a **capacity** that is λ times greater than the level below it, where λ is called

the **growth factor**.[2] When a level reaches capacity, it is merged into the next level (perhaps causing a merge cascade). The amortized IO cost for insertions is small because sequential merging is fast, although each item will participate in $\lambda/2$ merges on average. The IO cost for a query is high because a query must be performed independently on each of $O(\log_\lambda N)$ levels (although Bloom filters [5, 7] are sometimes used to mitigate this cost).

An ST-LSM further improves insertion IOs at the expense of queries. Each level contains fewer than λ runs. Every run on a given level has the same size, which is λ times larger than the runs on the level beneath it. When λ runs are present at a level, they are merged into one run and placed at the next level. There are therefore $O(\log_\lambda N)$ levels. Insertions are faster than in LT-LSMs because each item is only merged once on each level. Queries are slower because each query must be perform $O(\lambda)$ times at each level.

In LSMs, deletions can be implemented by the use of **upsert messages** [9, 13], which are a type of insertion with a message that indicates that the key has been deleted. A query for a key K then fetches all the matching key-value pairs and if the last one (temporally) is a deletion upsert, it returns false. To this end, the merges must maintain the temporal order of key-value pairs with the same key. Because a query for a key K must fetch every instance of K, the cost of a query is proportional to the number of times the key has been inserted and deleted, which we refer to as the **duplication count**, D_K of K. When $N/2$ deletions have been made, the structure is rebuilt to reclaim space. In what follows, deletions will be implemented using the same mechanism.

3 Bundle of Arrays Hashing

A **Bundle of Arrays Hash Table** (BOA) is an external-memory dictionary based on ST-LSMs. In this section, we describe a simple version which is optimal in the sense of Theorem 1, but where the query cost meets the bound only in expectation, not w.h.p. In Section 4, we give a version that satifies Theorem 2.

As a first step, we show that runs with uniformly distributed, $\Theta(\log N)$-wise independent fingerprints can be searched more quickly than in an ST-LSMs. In this section

▶ **Lemma 7.** *Let A be a sorted array of N uniformly distributed $\Theta(\log N)$-wise independent keys in the range $[0, K)$, and assume $B = \Omega(\log N)$. Then A can be written to external memory using $O(N)$ space and $O(N/B)$ IOs so that membership in A can be determined in $O(1)$ IOs with high probability.*

Proof. First note that, by Lemma 5 and Bonferroni's inequality, if N balls are thrown into $\Theta(N/\log N)$ bins uniformly and $\Theta(\log N)$-wise independently, then every bin has $\Theta(\log N)$ balls with high probability.

Divide the range of keys into N/B uniformly sized buckets; that is, bucket i contains keys in the range $[(i-1)KB/N, iKB/N)$. Because the keys in A are distributed uniformly, and $B = \Omega(\log N)$, every bucket contains $\Theta(B)$ keys with high probability. Let F be the number of items in the fullest bucket, and write the keys in each bucket to disk in order using F space for each. Because $F = \Theta(B)$, this takes the desired space and IOs.

Now, to find a key, compute which bucket it belongs to. A constant number of IOs will fetch that bucket, whose address is known because all buckets have the same size. ◀

[2] Sometimes this and related structures are analyzed with a growth factor of B^ϵ. The two are equivalent. We use λ rather than ϵ as the tuning parameter for consistency with the external-memory hashing literature.

▶ **Corollary 8.** *If an ST-LSM contains uniformly distributed and $\Theta(\log N)$-wise independent fingerprints and has growth factor λ, then a query for K can be performed in $O(D_K \lambda \log_\lambda N)$ IOs by writing the levels as in Lemma 7. The insertion/deletion cost is unchanged: $O\left(\frac{1}{B}\left(\log_\lambda N + \log_{\frac{M}{B}} N\right)\right)$ amortized IOs.*

While the query performance improves by a factor of $\log N$, the ST-LSM is still off the optimal tradeoff curve of Theorem 1. In particular, queries can be at least exponentially slower than optimal. The BOA use additional structure in order to reduce this query cost.

3.1 Routing Filters

The main result of this section is an auxiliary data structure, the **routing filter**, that improves the query cost of an ST-LSM by a factor of λ by further exploiting the log-wise independence of fingerprints. Combining these routing filters with fast interpolation search will yield the BOA, a hashing data structure that is optimal for large enough λ.

The purpose of the routing filter is to indicate probabilistically, at each level, which run contains the fingerprint we are looking for. Each level will have its own routing filter, defined as follows. For each level ℓ, let h_ℓ be some number, to be specified below. Let $P_\ell(K)$ be the prefix consisting of the first h_ℓ characters of K. The routing filter F_ℓ for level ℓ is a λ^{h_ℓ}-character array, where $F_\ell[i] = j$ if the jth run $R_{\ell,j}$ contains a fingerprint K such that the $P_\ell(K) = i$, and no later run $R_{\ell,j'}$ (i.e. with $j' > j$) contains such a fingerprint.

We also modify each run $R_{\ell,j}$ during the merge so that each fingerprint-value pair contains a **previous field** of 1 additional character used to specify the previous run containing a fingerprint with the same prefix, or j to indicate no such run exists. Thus these fingerprint-value pairs now form a singly linked list whose fingerprint share the same prefix, and the routing filter points to the run containing the head.

During a query for a fingerprint K, first $F_\ell[P_\ell(K)]$ is checked to find the latest run containing a fingerprint with a matching prefix. Once that fingerprint-value pair is found, its previous field indicates the next run which needs to be checked and so on until all fingerprints with matching prefix in the level are found. Each fingerprint $K' \neq K$ that matches K's prefix is a false positive.

Such routing filters induce a space/cost tradeoff. The greater h_ℓ is, the more space the table takes but the less likely it is that many runs will have false positives. The rest of this section shows that when $h_\ell = \log_\lambda B + \ell$, in other words, when prefixes grow by a character per level, the BOA lies on the optimal tradeoff curve of Theorem 1.

Define β, the **routing table ratio**, to be the ratio of the number of buckets in the routing filter to the size of a run. The number of entries in a run on level ℓ is $B\lambda^{\ell-1}$, so $\beta = \lambda^{h_\ell}/B\lambda^{\ell-1}$. We first analyze the per-level insertion/deletion cost, and then we compute the expected number of false positives in order to analyze the overall query cost.

▶ **Lemma 9.** *For a BOA with growth factor λ and routing table ratio β, merging a level incurs $\Theta\left(\frac{1}{B}\left(1 + \log_{\frac{M}{B}} \lambda + \beta \log_N \lambda\right)\right)$ IOs per fingerprint.*

Proof. Merging a level requires merging its runs as well as updating the next level's routing filter. Merging λ sorted arrays takes $\Theta\left(\frac{1}{B}\left(1 + \log_{M/B} \lambda\right)\right)$ IOs per fingerprint.

The routing filter is updated by iterating through it and the new run sequentially. For each fingerprint K appearing in the run, $F_{\ell+1}[P_{h_{\ell+1}}(K)]$ is copied to the previous field in the run, and $F_{\ell+1}[P_{h_{\ell+1}}(K)]$ is set to the number of the current run. Each entry in the routing filter is a character, and the routing filter has β entries for each new fingerprint. Thus, it requires $\Theta\left(\frac{\beta}{B} \log_N \lambda\right)$ IOs per fingerprint to update sequentially. ◀

▶ **Lemma 10.** *For a BOA with growth factor λ and routing table ratio β, querying a fingerprint K on a given level incurs at most $\frac{\lambda}{\beta}$ false positives in expectation.*

Proof. Given some enumeration of the fingerprints in level ℓ, which are not equal to K, denote the ith such fingerprint by K_i. Some of these may be the duplicates. Let X_i be the indicator random variable, which is 1 if $P_\ell(K) = P_\ell(K_i)$ and 0 otherwise. K and K_i are uniformly distributed and their bits are pairwise independent. Thus $\mathrm{E}[X_i] \leq \frac{1}{\lambda^{h_\ell}}$. The expected number of fingerprints (excluding K) in the level with prefix $P_\ell(K)$ is thus at most $\sum_{i=1}^{B\lambda^\ell} \mathrm{E}[X_i] \leq \frac{B\lambda^\ell}{\lambda^{h_\ell}} = \frac{\lambda}{\beta}$. ◀

▶ **Lemma 11.** *A BOA with growth factor λ and routing table ratio β has insertion/deletion cost $O\left(\frac{1}{B}\left(\beta + \log_{\frac{M}{B}} N + \log_\lambda N\right)\right)$. A query for fingerprint K has expected cost $O\left(\frac{\lambda}{\beta} D_K \log_\lambda N\right)$, where D_K is the duplication count of K.*

Proof. Because a BOA has $\log_\lambda N$ levels, the insertion cost follows from Lemma 9.

To query for a fingerprint K, the routing filter on each level is checked, which incurs $O(\log_\lambda N)$ IOs. These routing filters return a collection of runs which contain up to D_K true positives and an expected $O\left(\frac{\lambda}{\beta} \log_\lambda N\right)$ false positives, by Lemma 10. By Lemma 7, each run can be checked in $O(1)$ IOs. ◀

So for a fixed λ, there is no advantage to choosing $\beta = \omega(\lambda)$. On the other hand, $\beta = o(\lambda)$ is suboptimal, because then choosing $\beta' = \lambda' = \beta$ changes a linear factor in the query cost to a logarithmic one. Therefore, it is optimal to choose $\beta = \Theta(\lambda)$, and in what follows we will fix $\beta = \lambda$. Thus,

▶ **Lemma 12.** *A BOA supports N insertions and deletions with amortized per entry cost of $O\left(\left(\lambda + \log_{\frac{M}{B}} N + \log_\lambda N\right)/B\right)$ IOs, for any $\lambda > 1$. A query for a key K costs $O(D_K \log_\lambda N)$ IOs in expectation, where D_K is the duplication count of K.*

4 Refined Bundle of Arrays Hashing

In order to obtain high probability bounds for a BOA, we need a stronger guarantee on the number of false positives. This is achieved by including an additional character, the **check character** from each fingerprint in the routing filter, which is also checked during queries and thus eliminates most false positives. To support this, we will need to refine the routing filter so it can maintain check characters even when there are collisions.

The ith check character $C_i(K)$ of a fingerprint K is the ith character from the end of the string representation of K. As described in Section 2, we assume that the fingerprints are taken from a universe of size at least N^2 so that the check characters do not overlap with the characters used in the prefixes of the routing filters, and by $\Theta(\log N)$-wise independence, the check characters of $O(1)$ fingerprints are independent. Now each fingerprint in the filter has a check character and a array pointer, and we refer to this data as the **sketch** of the fingerprint.

When level i of the BOA is queried for a fingerprint K, the refined routing filter (described below) returns a list of sketches, one for each fingerprint in the level with prefix $P_i(K)$. The array indicated in the sketch is only checked if the check character matches $C_i(K)$, which reduces the number of false positives by a factor of λ.

Refined Routing Filter. The routing filter described in Section 3.1 handles prefix collisions by returning only the last run containing the queried fingerprint and then chaining in the

runs. Whereas, to support check characters, we need to return a list instead, while having the same performance guarantees.

The idea behind the refined routing filter is to keep the prefix-sketch pairs in a sorted list and use a hash table on prefixes to point queries to the appropriate place. Each pointer may require as many as $\Omega(\log N)$ bits, and we require the routing filter to have $O(1)$ characters per fingerprint. Therefore the hash table must use shorter prefixes so as to reduce the number of buckets and thus reduce its footprint. In particular, it uses prefixes which are $\log_\lambda \log_\lambda N$ characters shorter, which we refer to as **pivot prefixes**.

The list delta encodes the prefix for each fingerprint K, together with its sketch. In addition, the first entry following each pivot prefix contains the full prefix, rather then just the difference. Otherwise, when the hash table routes a query to that place in the list, the full prefix wouldn't be immediately computable.

▶ **Lemma 13.** *A refined routing filter can be updated using $O\left(\frac{\lambda \log \lambda}{B \log N}\right)$ IOs per new entry, and performs lookups in $O(D^*_K)$ IOs w.h.p., where D^*_K is the number of times K appears in the level.*

Proof. We prove first the update bound and then the query bound.

Let C be the capacity of the level. There are at most $\frac{C}{\log_\lambda N}$ pivot prefixes. For each pivot prefix, the hash table stores the bit position in a list with at most C entries, where $C \leq N$. Each entry is at most $\log N$ bits, so this position can be written using $O(\log N)$ bits.

For each fingerprint in the node, the list contains $O(1)$ characters by Lemma 6, or $O(\log \lambda)$ bits. Additionally, each pivot prefix has to an initial entry of length $O(\log N)$ bits, so the list all together uses $O(C \log \lambda + \frac{C}{\log_\lambda N} \cdot \log N) = O(C \log \lambda)$ bits.

When the refined routing filter is updated, the old version is read sequentially and the new version is written out sequentially. C/λ fingerprints are added at a time, so this incurs $O\left(\frac{\lambda \log \lambda}{B \log N}\right)$ IOs per entry.

During a query, the pivot bit string of a fingerprint and its successor are accessed from the hash table in $O(1)$ IOs. This returns the beginning and ending bit positions in the list. Because the fingerprints are distributed uniformly and are pairwise independent to K, there are $O(\log_\lambda N + D^*_K)$ fingerprints matching the pivot prefix in expectation. From Lemma 5 with $\delta = \log \lambda$, there are $O(\log N + D^*_K)$ fingerprints matching the pivot prefix w.h.p. The encoding of each fingerprint is less than a word, and $B = \Omega(\log N)$ by assumption, so this is $O(D^*_K)$ IOs. ◀

BOA Performance. We now can show:

▶ **Theorem 2.** *A BOA supports N insertions and deletions with amortized per entry cost of $O\left(\left(\lambda + \log_{\frac{M}{B}} N + \log_\lambda N\right)/B\right)$ IOs, for any $\lambda > 1$. A query for a key K costs $O(D_K \log_\lambda N)$ IOs w.h.p., where D_K is the number of times K has been inserted or deleted.*

Proof. The insertion/deletion cost is given by Lemma 13 and Theorem 12.

During a query for a fingerprint K, the expected number of false positives on level i (fingerprints which match the prefix $P_i(K)$ and check character $C_i(K)$ but are not K) is $O\left(\frac{1}{\lambda}\right)$. Thus, the number of false positives across levels is $O\left(\frac{\log_\lambda N}{\lambda}\right)$, so by Lemma 5, the number of false positives is $O\left(\log_\lambda N\right)$ w.h.p. ◀

Thus, a BOA is optimal for large enough λ:

▶ **Corollary 14.** *Let $\mathcal{B}$ be a BOA with growth factor λ containing N entries. If $\lambda = \Omega\left(\log_{\frac{M}{B}} N + \frac{\log N}{\log \log N}\right)$, then $\mathcal{B}$ is an optimal unsorted dictionary.*

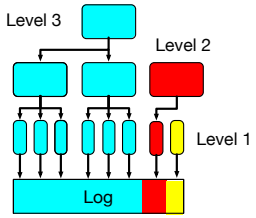


Figure 1 The routing trees in a 3 level BOT. The trees cover contiguous portions of the log. The highest level covers the beginning of the log, the next level the beginning of the remainder of the log, and so on.

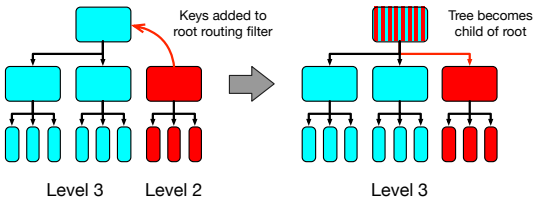


Figure 2 When the routing tree on level i fills, it is merged into the routing tree on level $i+1$. The now-full routing tree from level $i+1$ becomes a child of the root on level $i+1$. Its fingerprints are added to the root routing filter. Note that the tree is not moved.

5 Bundle of Trees Hashing

In order for a BOA to be an optimal dictionary, its growth factor λ must be $\Omega(\log N/\log\log N)$. Otherwise, the cost of insertion is dominated by the cost of merging, which in slow because it effectively sorts the fingerprints using a λ-ary merge sort. In this section, we present the **Bundle of Trees Hash Table** (BOT), which is a BOA-like structure. A BOT stores the fingerprints in a log in the order in which they arrive. Each level of the BOT is like a level of a BOA, where the bundle of arrays on each level is replaced by an search structure on the log (the **routing tree**) and a data structure needed to merge routing trees (the **character queue**). The character queue performs a delayed sort on the characters needed at each level, thus increasing the arity of the sort and decreasing the IOs.

A BOT has $s = \lceil \log_\lambda N/B \rceil$ levels, each of which contains a routing tree. The root of the routing tree has degree less than λ and all internal nodes have degree λ. Each node of a routing tree contains a routing filter. As in Section 4, each routing filter takes as input a fingerprint K and outputs a list of sketches corresponding to fingerprints with the same prefix as K. Each sketch consists of a pointer to a child, a check character and some auxiliary information discussed below.

Each leaf points to a block of B fingerprints in the log. The deepest level s uses a height-s tree to index the beginning of the fingerprint log, the next level then indexes the next section and so forth, as shown in Figure 1. Insertions and deletions (as upsert messages) are appended to the log until they form a block, at which point they are added to the tree in the 1st level of the BOT.

When a level i in the BOT fills, its routing tree is merged into the routing tree of level $i+1$, thus increasing the degree of the target routing tree by 1 (and perhaps filling level $i+1$, which triggers a merge of level $i+1$ into $i+2$, and so on). The merge of level i into level $i+1$ consists of adding the prefix-sketch pairs of the fingerprints from level i to the routing filter of the root on level $i+1$. The child pointers of these pairs will point to the root of the formerly level-i routing tree, so it becomes a child of the root of the level $i+1$ routing tree, although it isn't moved or copied. See Figure 2. In this way, a BOT resembles an LT-LSM, described in Section 2.

In order to add a fingerprint K from level i to the root routing filter on level $i+1$, the prefix $P_{i+1}(K)$ must be known. However, the root routing filter on level i only stores the prefix $P_i(K)$ for each fingerprint K it contains, so that in particular the last character of $P_{i+1}(K)$ is missing. As described in Section 5.2, each level has a character queue, which provides this character, as well as the check characters, in order to merge the routing trees efficiently.

5.1 Queries in a BOT

A query to the BOT for a fingerprint K is performed independently at each level, beginning at the root of each routing tree. When a node is queried, its routing filter returns a list of sketches. The sketches whose check characters match the queried fingerprint indicate to which children the query is passed. This process continues until the query reaches a block of the log, which is then searched in full. In this way queries are "routed" down the tree on each level to the part of log where the fingerprint and its associated value are. Note that as queries descend the routing tree, they may generate false positives which are likewise routed down towards the log.

In this section, we refine routing trees so that they offer two guarantees about false positives. The first is that at each level, the probability that a given false positive is not elinimated is at most $\frac{1}{\lambda}$. The second is that false positives can only be created in the root, so that as the query descends the tree, the number of false positives cannot increase.

During a query to a node of height h for a fingerprint K, the routing filter returns a list of sketches corresponding to fingerprints which match K's prefix. The query only proceeds on those children whose check characters also match the check character $C_h(K)$. Since the characters of the fingerprint are uniformly distributed and $\Theta(\log N)$-wise independent, the check character of each false positive matches with probability $\frac{1}{\lambda}$. Moreover, the characters of each level are non-overlapping, so for fingerprints K, K' the event that $V_h(K) = V_h(K')$ is independent of the event that $V_{h-1}(K) = V_{h-1}(K')$.

To prevent new false positives from being generated when a query passes from a parent to a child, the **next character** of each fingerprint is also kept in its sketch in the routing filter. For a fingerprint K in a node of height h, the next character is just the next character that follows the prefix, $P_h(K)$, so that its prefix in the parent, $P_{h+1}(K)$, can be obtained. A false positive in a child which is not in the parent will not match this next character and can be eliminated.

When there are multiple prefix-matching fingerprints in both a parent and its child, we would like to be able to align the lists returned by the routing filters so that known false positives in the parent (either from check or next characters) can be eliminated in the child. Otherwise the check character in the child of a known false positive in the parent may match the queried fingerprint, and therefore more than $\frac{1}{\lambda}$ of the false positives may survive. To this end, we require the routing filter to return the list of sketches in the order their fingerprint-value pairs appear in the log. Then after the sketches in the child list whose next characters do not match the parent are eliminated, the remaining phrases will be in the same order as in the parent. In this way, known false positives can also be eliminated in the child.

Now we can show:

▶ **Lemma 15.** *During a query to a routing tree, the following are true:*

1. *A false positive can only be generated in the root.*
2. *At each level, a given false positive survives with probability at most* $\frac{1}{\lambda}$.

Proof. Because of the next characters, false positives may only be created in the root of the routing tree. Each false positive in the root corresponds to a fingerprint K' in the level. At each node on the path to K''s location in the log, we use the ordering to determine which returned sketch corresponds to K', so that the false positive corresponding to K' is eliminated with probability $\frac{1}{\lambda}$. ◀

5.2 Character Queue

The purpose of the character queue is to store all the sketches of fingerprints contained in a level i that will be needed during a merge in the future. When level i is merged into level $i+1$, the character queue outputs a sorted list of the delta-encoded prefix-sketch pairs of all the fingerprints, which is used to update the root routing tree. The character queue is then merged into the character queue on level $i+1$.

The character queue effectively performs a merge sort on the sketches. If it were to merge all the sketches as soon as they are available, this would consist of λ-ary merges. In order to increase the arity of the merges, it defers merging sketches which are not needed immediately. The sketches are stored collection of **series**, by which we mean a collection of sorted runs. Each series stores a continuous range of sketches $S_i(K), S_{i+1}(K), \ldots, S_{i+j}(K)$ for each fingerprint K, together with the prefix up to the first sketch, $P_{i-1}(K)$. These prefixes are delta encoded in their run. Thus the size of an entry is determined by the number of sketches in the range and the length of the prefix relative to the size of the run (by Lemma 6).

The character queue tradeoff. We are faced with the following tradeoff. If the character queue merges a series frequently, the delta encoding is more efficient, which decreases the cost of the merging. However the arity is lower, which increases it. The character queue uses a merging schedule which balances this tradeoff and thus achieves optimal insertions.

The character queue merging schedule. The character queue on level i (here we consider blocks of the log to be level 0) contains the sketches $S_{i+1}(K), S_{i+2}(K), \ldots S_s(K)$ of each fingerprint K in the level. These characters are stored in a collection of series $\{\sigma_{j_q}\}$, where j_q is the smallest multiple of 2^q greater than i. Series σ_{j_q} contains the sketches $S_{j_q}(K), \ldots, S_{j_{q+1}-1}(K)$. Each series consists of a collection of sorted runs each of which stores the delta encoded prefix of each fingerprint together with its sketches.

Initially, when a block of the log is written, all the series σ_{2^q} for $q = 1, 2, 3, \ldots$ are created. When level i fills, the runs in the series σ_{i+1} are merged, and the character queue outputs the delta encoded prefix-sketch pairs, $(P_{i+1}(K), S_{i+1}(K))$ to update the root routing filter on level $i+1$. If $2^{\rho(i+q)}$ is the greatest power of 2 dividing $i+1$ (ρ is sometimes referred to as the **ruler function** [15]), then σ_{i+1} also contains the next $2^{\rho(i+1)}-1$ sketches of each fingerprint. These are batched and delta encoded to become runs in the series σ_{j_q} for $q = [0, \rho(i+1)]$. The runs in the remaining series of level i becomes runs of their respective series on level $i+1$.

Note that for the lower levels, some runs in may be shorter than B due to the delta encoding. For a run in a series σ_q, this is handled by buffering them with the runs σ_q of higher levels and writing them out once they are of size B. Note that this requires $O(B \log\log N)$ memory.

This leads to the following merging pattern: σ_j batches $2^{\rho(j)}$ sketches, and has delta encoded prefixes of $2^{\rho(j)}$ characters on average, by Lemma 6. Therefore,

▶ **Lemma 16.** *A series σ_j in a character queue contains $O(2^{\rho(j)})$ characters per fingerprint.*

This leads to a merging schedule where the characters per item merged on the jth level is $O(2^{\rho(j)})$. Starting from 1 this is $1, 2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2, 1, 16, \ldots$, which resemble the tick marks of a ruler, hence the name ruler function.

We now analyze the cost of maintaining the character queues.

▶ **Lemma 17.** *The total per-insertion/deletion cost to update the character queues in a BOT is $\Theta\left(\frac{1}{B}\left(\log_{\frac{M}{B}} N + \log\log M\right)\right)$.*

Proof. When σ_j is merged, $\lambda^{2^{\rho(j)}}$ runs are merged, which has a cost of $O\left(\frac{2^{\rho(j)}}{B}\left\lceil \log_{M/B}\left(\lambda^{2^{\rho(j)}}\right)\right\rceil\right)$ characters per fingerprint.

There are $\log_\lambda \frac{N}{B} = O(\log_\lambda N)$ levels, so this leads to the following total cost in terms of characters:

$$\begin{aligned} O\left(\sum_{i=1}^{\log_\lambda N} 2^{\rho(j)}\left\lceil \log_{\frac{M}{B}}\left(\lambda^{2^{\rho(j)}}\right)\right\rceil\right) &= O\left(\sum_{k=0}^{\log\log_\lambda N} \frac{\log_\lambda N}{2^k}\cdot 2^k\left\lceil \log_{\frac{M}{B}}\left(\lambda^{2^k}\right)\right\rceil\right) \\ &= O\left(\log_\lambda N\left(\log\log M + \sum_{k=\log\log M}^{\log\log_\lambda N} 2^k \log_{\frac{M}{B}}\lambda\right)\right) \\ &= O\left(\log_\lambda N\left(\log\log M + \log_{\frac{M}{B}} N\right)\right), \end{aligned}$$

where the last equality is because the RHS sum is dominated by its last term. Because there are $\log_\lambda N$ characters in a word, and all reads and writes are performed sequentially in runs of size at lease B, the result follows. ◀

5.3 Performance of the BOT

We can now prove Theorem 3:

▶ **Theorem 3.** *A BOT supports N insertions and deletions with amortized per entry cost of $O\left(\left(\lambda + \log_{\frac{M}{B}} N + \log\log M\right)/B\right)$ IOs for any $\lambda > 1$. A query for a key K costs $O(D_K \log_\lambda N)$ IOs w.h.p., where D_K is the number of times K has been inserted or deleted.*

Proof. By Lemma 13, the cost of updating the routing filters is $O\left(\frac{\lambda}{B}\right)$, since there are $O(\log_\lambda N)$ levels. This, together with the cost of updating the character queues, given by Lemma 17, is the insertion cost.

By Lemma 15, a query for fingerprint K on level i incurs $O\left(\frac{1}{\lambda}\right)$ false positives in the root, and $O(1)$ nodes are accessed along each of their root-to-leaf paths. By Lemma 13, each false positive thus incurs $O(D_K)$ IOs.

There are an expected $O\left(\frac{\log_\lambda N}{\lambda}\right)$ false positives across all levels, so, using Lemma 5 with $\delta = \lambda$, $O(D_K \log_\lambda N)$ nodes are accessed due to false positives w.h.p. For each time K appears in the BOT, $O(\log_\lambda N)$ nodes are accessed on its root-to-leaf path. By Lemma 13 the node accesses along each path incur $O(D_K \log_\lambda)$ IOs w.h.p., so accessing the nodes incurs $O(\log_\lambda N)$ IOs w.h.p.

A block of the log is scanned at most D_K times for true positives and also whenever a false positive from the level-i root survives i times. The expected number of such false positives for level i is $1/\lambda^i$, so the expected number across levels is $O\left(\frac{1}{\lambda}\right)$. Therefore by Lemma 5, the number of blocks scanned is $O(D_K \log_\lambda N)$ w.h.p. ◀

▶ **Corollary 18.** *Let $\mathcal{B}$ be a BOT with growth factor λ containing N entries. If $\lambda = \Omega\left(\log_{\frac{M}{B}} N + \log\log M\right)$, then $\mathcal{B}$ is an optimal dictionary.*

6 Cache-Oblivious BOTs

In this section, we show how to modify a BOT to be cache oblivious. We call the resulting structure a cache-oblivious hash tree (COBOT).

Much of the structure of the BOT translates directly into the cache-oblivious model. However, some changes are necessary. In particular, when the series of character queues are merged, this merge must be performed cache-obliviously using funnels [10], rather than with an (up to) M/B-way merge. Also, the log cannot be buffered into sections of size $O(B)$, and so instead they are buffered into sections of constant size, items are immediately added to routing filter, and the extra IOs are eliminated by optimal caching.

When an insertion is made into a COBOT, its fingerprint-value pair is appended to the log, and it is immediately inserted into level 1. Thus, the leaves of the routing trees point to single entries in the log.

The series of the character queues must be placed more carefully as well. In particular the runs of series σ_j must be laid out back-to-back for all j (rather than just small j as in Section 5.2), so that the caching algorithm can buffer them appropriately.

The series are merged using a **partial funnelsort**. Funnelsort is a cache-oblivious sorting algorithm that makes use of K-funnels [10]. A K-funnel is a CO data structure that merges K sorted lists of total length N. We make use of the the following lemma.

▶ **Lemma 19** ([10]). *A K-funnel merges K sorted lists of total length $N \geq K^3$ in $O\left(\frac{N}{B}\log_{M/B}\frac{N}{B} + K + \frac{N}{B}\log_K\frac{N}{B}\right)$ IOs, provided the tall cache assumption that $M = \Omega(B^2)$ holds.*

The partial funnelsort used to merge K runs of a series with total length L (in words) performs a single merge with a K-funnel if $L \geq K^3$ and recursively merges the run in groups of $K^{1/3}$ runs otherwise.

▶ **Corollary 20.** *A partial funnelsort merges K runs of total word length L in $O\left(\frac{L}{B}\log_{M/B}\frac{L}{B} + \frac{L}{B}\log_K\frac{L}{B}\right)$ IOs, provided the tall cache assumption that $M = \Omega(B^2)$ holds.*

Proof. The base case of the recursion occurs either when there is only 1 list remaining or the remaining lists fit in memory. In any other case of the recursion, since $L = \Omega(B^2)$ by the tall cache assumption, the K term in Lemma 19 is dominated.

The recurrence is dominated by the cost of the funnel merges, which yields the result. ◀

▶ **Theorem 21.** *If $M = \Omega(B^2)$, then a COBOT with N entries and growth factor λ has amortized insertion/deletion cost $\Theta\left(\frac{1}{B}\left(\lambda + \log\log M + \log_{M/B} N/B\right)\right)$. A query for key K has cost $\Theta\left(D_K \log_\lambda N\right)$, w.h.p., where D_K is the duplication count of K.*

Proof. We may assume that the caching algorithm sets aside enough memory that the last B items in the log, together with the subtree rooted at their least common ancestor, are cached. Thus the log is updated at a per-item cost of $O(1/B)$.

The proof of Theorem 3 now carries over to the COBOT. The routing filters are updated the same way, and the cost of updating the character queues is unchanged, by Corollary 20.

Queries are performed as in Section 5.1, except that now the level 1 nodes cover $O(1)$ fingerprints, but the depth of the tree is unchanged, so the cost is the same. ◀

References

1 Peyman Afshani, Michael A. Bender, Martin Farach-Colton, Jeremy T. Fineman, Mayank Goswami, and Meng-Tsung Tsai. Cross-referenced dictionaries and the limits of write optimization. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1523–1532. SIAM, 2017. doi:10.1137/1.9781611974782.99.

2 Michael A. Bender, Richard Cole, Erik D. Demaine, and Martin Farach-Colton. Scanning and traversing: Maintaining data for traversals in a memory hierarchy. In Rolf H. Möhring and Rajeev Raman, editors, *Algorithms - ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17-21, 2002, Proceedings*, volume 2461 of *Lecture Notes in Computer Science*, pages 139–151. Springer, 2002. doi:10.1007/3-540-45749-6_16.

3 Michael A. Bender, Martin Farach-Colton, Jeremy T. Fineman, Yonatan R. Fogel, Bradley C. Kuszmaul, and Jelani Nelson. Cache-oblivious streaming b-trees. In Phillip B. Gibbons and Christian Scheideler, editors, *SPAA 2007: Proceedings of the 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Diego, California, USA, June 9-11, 2007*, pages 81–92. ACM, 2007. doi:10.1145/1248377.1248393.

4 Michael A. Bender, Martin Farach-Colton, Mayank Goswami, Dzejla Medjedovic, Pablo Montes, and Meng-Tsung Tsai. The batched predecessor problem in external memory. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 112–124. Springer, 2014. doi:10.1007/978-3-662-44777-2_10.

5 Michael A. Bender, Martin Farach-Colton, Rob Johnson, Russell Kraner, Bradley C. Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P. Spillane, and Erez Zadok. Don't thrash: How to cache your hash on flash. *PVLDB*, 5(11):1627–1637, 2012. URL: http://vldb.org/pvldb/vol5/p1627_michaelabender_vldb2012.pdf.

6 Michael A. Bender, Martin Farach-Colton, Rob Johnson, Simon Mauras, Tyler Mayer, Cynthia A. Phillips, and Helen Xu. Write-optimized skip lists. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 69–78. ACM, 2017. URL: http://dl.acm.org/citation.cfm?id=3034786, doi:10.1145/3034786.3056117.

7 Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970. doi:10.1145/362686.362692.

8 Gerth Stølting Brodal and Rolf Fagerberg. Lower bounds for external memory dictionaries. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 546–554. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644201.

9 John Esmet, Michael A. Bender, Martin Farach-Colton, and Bradley C. Kuszmaul. The tokufs streaming file system. In *4th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage'12, Boston, MA, USA, June 13-14, 2012*, 2012. URL: https://www.usenix.org/conference/hotstorage12/workshop-program/presentation/esmet.

10 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 285–298. IEEE Computer Society, 1999. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6604, doi:10.1109/SFFCS.1999.814600.

11 John Iacono and Mihai Patrascu. Using hashing to solve the dictionary problem (in external memory). *CoRR*, abs/1104.2799, 2011. arXiv:1104.2799.

12 John Iacono and Mihai Patrascu. Using hashing to solve the dictionary problem. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 570–582. SIAM, 2012. URL: http://portal.acm.org/citation.cfm?id=2095164&CFID=63838676&CFTOKEN=79617016, doi:10.1137/1.9781611973099.

13 William Jannen, Michael A. Bender, Martin Farach-Colton, Rob Johnson, Bradley C. Kuszmaul, and Donald E. Porter. Lazy analytics: Let other queries do the work for you. In

8th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2016, Denver, CO, June 20-21, 2016., 2016. URL: https://www.usenix.org/conference/hotstorage16/workshop-program/presentation/jannen.

14 Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM J. Discrete Math.*, 8(2):223–250, 1995. doi:10.1137/S089548019223872X.

15 Wikipedia. Thomae's function — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Thomae's%20function&oldid=837510765, 2018. [Online; accessed 28-April-2018].

Lovász Meets Weisfeiler and Leman

Holger Dell
Saarland University and Cluster of Excellence (MMCI), Saarbrücken, Germany
hdell@mmci.uni-saarland.de
https://orcid.org/0000-0001-8955-0786

Martin Grohe
RWTH Aachen University, Aachen, Germany
grohe@informatik.rwth-aachen.de
https://orcid.org/0000-0002-0292-9142

Gaurav Rattan
RWTH Aachen University, Aachen, Germany
grohe@informatik.rwth-aachen.de
https://orcid.org/0000-0002-5095-860X

Abstract

In this paper, we relate a beautiful theory by Lovász with a popular heuristic algorithm for the graph isomorphism problem, namely the color refinement algorithm and its k-dimensional generalization known as the Weisfeiler-Leman algorithm. We prove that two graphs G and H are indistinguishable by the color refinement algorithm if and only if, for all trees T, the number $\mathsf{Hom}(T, G)$ of homomorphisms from T to G equals the corresponding number $\mathsf{Hom}(T, H)$ for H.

There is a natural system of linear equations whose nonnegative integer solutions correspond to the isomorphisms between two graphs. The nonnegative real solutions to this system are called fractional isomorphisms, and two graphs are fractionally isomorphic if and only if the color refinement algorithm cannot distinguish them (Tinhofer 1986, 1991). We show that, if we drop the nonnegativity constraints, that is, if we look for arbitrary real solutions, then a solution to the linear system exists if and only if, for all t, the two graphs have the same number of length-t walks.

We lift the results for trees to an equivalence between numbers of homomorphisms from graphs of tree width k, the k-dimensional Weisfeiler-Leman algorithm, and the level-k Sherali-Adams relaxation of our linear program. We also obtain a partial result for graphs of bounded path width and solutions to our system where we drop the nonnegativity constraints.

A consequence of our results is a quasi-linear time algorithm to decide whether, for two given graphs G and H, there is a tree T with $\mathsf{Hom}(T, G) \neq \mathsf{Hom}(T, H)$.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis, Mathematics of computing → Graph theory

Keywords and phrases graph isomorphism, graph homomorphism numbers, tree width

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.40

Related Version A full version of the paper is available at https://arxiv.org/abs/1802.08876.

1 Introduction

An old result due to Lovász [16] states a graph G can be characterized by counting homomorphisms from all graphs F to G. That is, two graphs G and H are isomorphic if and only if, for all F, the number $\mathsf{Hom}(F, G)$ of homomorphisms from F to G equals the number $\mathsf{Hom}(F, H)$ of homomorphism from F to H. This simple result has far reaching consequences,

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 40; pp. 40:1–40:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

because mapping graphs G to their *homomorphism vectors* $\mathsf{HOM}(G) := \big(\mathsf{Hom}(F,G)\big)_{F \text{ graph}}$ (or suitably scaled versions of these infinite vectors) allows us to apply tools from functional analysis in graph theory. This is the foundation of the beautiful theory of graph limits, developed by Lovász and others over the last 15 years (see [17]).

However, from a computational perspective, representing graphs by their homomorphism vectors has the disadvantage that the problem of computing the entries of these vectors is NP-complete. To avoid this difficulty, we may want to restrict the homomorphism vectors to entries from a class of graphs for which counting homomorphisms is tractable. That is, instead of considering the full homomorphism vector $\mathsf{HOM}(G)$ we consider the vector $\mathsf{HOM}_{\mathcal{F}}(G) := \big(\mathsf{Hom}(F,G)\big)_{F\in\mathcal{F}}$ for a class $\mathcal{F}$ of graphs such that the problem of computing $\mathsf{Hom}(F,G)$ for given graphs $F \in \mathcal{F}$ and G is in polynomial time. Arguably the most natural example of such a class $\mathcal{F}$ is the class of all trees. More generally, computing $\mathsf{Hom}(F,G)$ for given graphs $F \in \mathcal{F}$ and G is in polynomial time for all classes $\mathcal{F}$ of bounded tree width, and under a natural assumption from parameterized complexity theory, it is not in polynomial time for any class $\mathcal{F}$ of unbounded tree width [10]. This immediately raises the question what the vector $\mathsf{HOM}_{\mathcal{F}}(G)$, for a class $\mathcal{F}$ of bounded tree width, tells us about the graph G.

A first nice example (Proposition 9) is that the vector $\mathsf{HOM}_{\mathcal{C}}(G)$ for the class $\mathcal{C}$ of all cycles characterizes the spectrum of a graph, that is, for graphs G, H we have $\mathsf{HOM}_{\mathcal{C}}(G) = \mathsf{HOM}_{\mathcal{C}}(H)$ if and only if the adjacency matrices of G and H have the same eigenvalues with the same multiplicities. This equivalence is a basic observation in spectral graph theory (see [23, Lemma 1]). Before we state deeper results along these lines, let us describe a different (though related) motivation for this research.

Determining the similarity between two graphs is an important problem with many applications, mainly in machine learning, where it is known as "graph matching" (e.g. [9]). But how can the similarity between graphs be measured? An obvious idea is to use the *edit distance*, which simply counts how many edges and vertices have to be deleted from or added to one graph to obtain the other. However, two graphs that have a small edit distance can nevertheless be structurally quite dissimilar (e.g. [17, Section 1.5.1]). The edit distance is also very hard to compute as it is closely related to the notoriously difficult quadratic assignment problem (e.g. [3, 19]).

Homomorphism vectors offer an alternative, more structurally oriented approach to measuring graph similarity. After suitably scaling the vectors, we can can compare them using standard vector norms. This idea is reminiscent of the "graph kernels" used in machine learning (e.g. [24]). Like the homomorphism vectors, many graph kernels are based on the idea of counting certain patterns in graphs, such as paths, walks, cycles or subtrees, and in fact any inner product on the homomorphism vectors yields a graph kernel.

A slightly different type of graph kernel is the so-called Weisfeiler-Leman (subtree) kernel [20]. This kernel is derived from the *color refinement* algorithm (a.k.a. the *1-dimensional Weisfeiler-Leman algorithm*), which is a simple and efficient heuristic to test whether two graphs are isomorphic (e.g. [11]). The algorithm computes a coloring of the vertices of a graph based on the iterated degree sequences, we give the details in Section 3. To use it as an isomorphism test, we compare the color patterns of two graphs. If they are different, we say that color refinement *distinguishes* the graphs. If the color patterns of the two graphs turn out to be the same, the graphs may still be non-isomorphic, but the algorithm fails to detect this.

Whether color refinement is able to distinguish two graphs G and H has a very nice linear-algebraic characterization due to Tinhofer [21, 22]. Let V and W be the vertex sets and let $A \in \{0,1\}^{V\times V}$ and $B \in \{0,1\}^{W\times W}$ be the adjacency matrices of G and H, respectively.

Now consider the system $\mathsf{F}_{\text{iso}}(G,H)$ of linear equations:

$$\mathsf{F}_{\text{iso}}(G,H): \quad \begin{cases} AX = XB & \text{(F1)} \\ X\mathbf{1}_W = \mathbf{1}_V & \text{(F2)} \\ \mathbf{1}_V^T X = \mathbf{1}_W^T & \text{(F3)} \end{cases}$$

In these equations, X denotes a $(V \times W)$-matrix of variables and $\mathbf{1}_U$ denotes the all-1 vector over the index set U. Equations (F2) and (F3) simply state that all row and column sums of X are supposed to be 1. Thus the nonnegative integer solutions to $\mathsf{F}_{\text{iso}}(G,H)$ are permutation matrices, which due to (F1) describe isomorphisms between G and H. The nonnegative real solutions to $\mathsf{F}_{\text{iso}}(G,H)$, which in fact are always rational, are called *fractional isomorphisms* between G and H. Tinhofer proved that two graphs are fractionally isomorphic if and only if color refinement does not distinguish them.

For every $k \geq 2$, color refinement has a generalization, known as the *k-dimensional Weisfeiler-Leman algorithm (k-WL)*, which colors not the vertices of the given graph but k-tuples of vertices. Atserias and Maneva [4] (also see [18]) generalized Tinhofer's theorem by establishing a close correspondence between k-WL and the level-k Sherali-Adams relaxation of $\mathsf{F}_{\text{iso}}(G,H)$.

Our results

How expressive are homomorphism vectors $\mathsf{HOM}_{\mathcal{F}}(G)$ for restricted graph classes $\mathcal{F}$? We consider the class $\mathcal{T}$ of trees first, where the answer is surprisingly clean.

▶ **Theorem 1.** *For all graphs G and H, the following are equivalent:*

i $\mathsf{HOM}_{\mathcal{T}}(G) = \mathsf{HOM}_{\mathcal{T}}(H)$.
ii *Color refinement does not distinguish G and H.*
iii *G and H are fractionally isomorphic, that is, the system $\mathsf{F}_{\text{iso}}(G,H)$ of linear equations has a nonnegative real solution.*

As mentioned before, the equivalence between ii and iii is due to Tinhofer [21, 22]. An unexpected consequence of our theorem is that we can decide in time $O((n+m)\log n)$ whether $\mathsf{HOM}_{\mathcal{T}}(G) = \mathsf{HOM}_{\mathcal{T}}(H)$ holds for two given graphs G and H with n vertices and m edges. (If two graphs have a different number of vertices or edges, then their homomorphism counts already differ on the 1-vertex or 2-vertex trees.) This is remarkable, because every known algorithm for computing the entry $\mathsf{Hom}(T,G)$ of the vector $\mathsf{HOM}_{\mathcal{T}}(G)$ requires quadratic time when T and G are given as input.

It is a consequence of the proof of Theorem 1 that, in order to characterize an n-vertex graph G up to fractional isomorphisms, it suffices to restrict the homomorphism vector $\mathsf{HOM}_{\mathcal{T}}(G)$ to trees of height at most $n-1$. What happens if we restrict the structure of trees even further? In particular, let us restrict the homomorphism vector to its path entries, that is, consider $\mathsf{HOM}_{\mathcal{P}}(G)$ for the class $\mathcal{P}$ of all paths. Figure 1 shows an example of two graphs G and H with $\mathsf{HOM}_{\mathcal{P}}(G) = \mathsf{HOM}_{\mathcal{P}}(H)$ and $\mathsf{HOM}_{\mathcal{T}}(G) \neq \mathsf{HOM}_{\mathcal{T}}(H)$.

Despite their weaker distinguishing capabilities, the vectors $\mathsf{HOM}_{\mathcal{P}}(G)$ are quite interesting. They are related to graph kernels based on counting walks, and they have a clean algebraic description: it is easy to see that $\mathsf{Hom}(P_k,G)$, the number of homomorphisms from the path P_k of length k to G, is equal to the number of length-k walks in G, which in turn is equal to $\mathbf{1}^T A^k \mathbf{1}$, where A is the adjacency matrix of G and $\mathbf{1}$ is the all-1 vector of appropriate length.

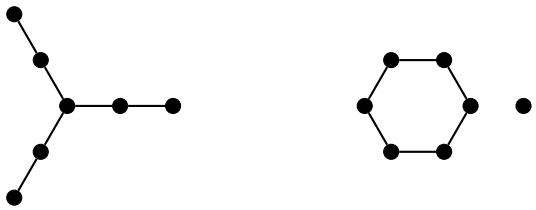


Figure 1 Two fractionally non-isomorphic graphs with the same path homomorphism counts.

▶ **Theorem 2.** *For all graphs G and H, the following are equivalent:*

i $\mathsf{HOM}_{\mathcal{P}}(G) = \mathsf{HOM}_{\mathcal{P}}(H)$.

ii *The system $\mathsf{F}_{\text{iso}}(G,H)$ of linear equations has a real solution.*

While the proof of Theorem 1 is mainly graph-theoretic—we establish the equivalence between the assertions i and ii by expressing the "colors" of color refinement in terms of specific tree homomorphisms—the proof of Theorem 2 is purely algebraic. We use spectral techniques, but with a twist, because neither does the spectrum of a graph G determine the vector $\mathsf{HOM}_{\mathcal{P}}(G)$ nor does the vector determine the spectrum. This is in contrast with $\mathsf{HOM}_{\mathcal{C}}(G)$ for the class $\mathcal{C}$ of all cycles, which, as we already mentioned, distinguishes two graphs if and only if they have the same spectrum.

Let us now turn to homomorphism vectors $\mathsf{HOM}_{\mathcal{T}_k}(G)$ for the class $\mathcal{T}_k$ of all graphs of tree width at most k. We will relate these to k-WL, the k-dimensional generalization of color refinement. We also obtain a corresponding system of linear equations. Let G and H be graphs with vertex sets V and W, respectively. Instead of variables X_{vw} for vertex pairs $(v,w) \in V \times W$, as in the system $\mathsf{F}_{\text{iso}}(G,H)$, the new system has variables X_π for $\pi \subseteq V \times W$ of size $|\pi| \leq k$. We call $\pi = \{(v_1,w_1),\dots,(v_\ell,w_\ell)\} \subseteq V \times W$ a *partial bijection* if $v_i = v_j \iff w_i = w_j$ holds for all i,j, and we call it a *partial isomorphism* if in addition $v_iv_j \in E(G) \iff w_iw_j \in E(H)$ holds for all i,j. Now consider the following system $\mathsf{L}^k_{\text{iso}}(G,H)$ of linear equations:

$$\mathsf{L}^k_{\text{iso}}(G,H): \begin{cases} \displaystyle\sum_{v\in V} X_{\pi\cup\{(v,w)\}} = X_\pi & \text{for all } \pi \subseteq V\times W \text{ of size } |\pi| \leq k-1 \text{ and all } w \in W \quad \text{(L1)} \\ \displaystyle\sum_{w\in W} X_{\pi\cup\{(v,w)\}} = X_\pi & \text{for all } \pi \subseteq V\times W \text{ of size } |\pi| \leq k-1 \text{ and all } v \in V \quad \text{(L2)} \\ X_\pi = 0 & \text{for all } \pi \subseteq V\times W \text{ of size } |\pi| \leq k \text{ such that } \pi \text{ is not a partial isomorphism from } G \text{ to } H \quad \text{(L3)} \\ X_\emptyset = 1 & \quad \text{(L4)} \end{cases}$$

This system is closely related to the Sherali-Adams relaxations of $\mathsf{F}_{\text{iso}}(G,H)$: Every solution for the level-k Sherali-Adams relaxation of $\mathsf{F}_{\text{iso}}(G,H)$ yields a solution to $\mathsf{L}^k_{\text{iso}}(G,H)$, and every solution to $\mathsf{L}^k_{\text{iso}}(G,H)$ yields a solution to the level $k-1$ Sherali-Adams relaxation of $\mathsf{F}_{\text{iso}}(G,H)$ [4, 12]. Our result is this:

▶ **Theorem 3.** *For all $k \geq 1$ and for all graphs G and H, the following are equivalent:*

i $\mathsf{HOM}_{\mathcal{T}_k}(G) = \mathsf{HOM}_{\mathcal{T}_k}(H)$.

ii *k-WL does not distinguish G and H.*

iii $\mathsf{L}^{k+1}_{\text{iso}}(G,H)$ *has a nonnegative real solution.*

The equivalence between ii and iii is implicit in previous work [14, 4, 12]. The system $\mathsf{L}^k_{\mathrm{iso}}(G,H)$ has another nice interpretation related to the proof complexity of graph isomorphism: it is shown in [7] that $\mathsf{L}^k_{\mathrm{iso}}(G,H)$ has a real solution if and only if a natural system of polynomial equations encoding the isomorphisms between G and H has a degree-k solution in the Hilbert Nullstellensatz proof system [6, 8]. In view of Theorem 2, it is tempting to conjecture that the solvability of $\mathsf{L}^{k+1}_{\mathrm{iso}}(G,H)$ characterizes the expressiveness of the homomorphism vectors $\mathsf{HOM}_{\mathcal{P}_k}(G)$ for the class $\mathcal{P}_k$ of all graphs of path width k. Unfortunately, we only prove one direction of this conjecture.

▶ **Theorem 4.** *Let k be an integer with $k \geq 2$ and let G, H be graphs. If $\mathsf{L}^{k+1}_{\mathrm{iso}}(G,H)$ has a real solution, then $\mathsf{HOM}_{\mathcal{P}_k}(G) = \mathsf{HOM}_{\mathcal{P}_k}(H)$.*

Combining this theorem with a recent result from [13] separating the nonnegative from arbitrary real solutions of our systems of equations, we obtain the following corollary.

▶ **Corollary 5.** *For every k, there are graphs G and H with $\mathsf{HOM}_{\mathcal{P}_k}(G) = \mathsf{HOM}_{\mathcal{P}_k}(H)$ and $\mathsf{HOM}_{\mathcal{T}_2}(G) \neq \mathsf{HOM}_{\mathcal{T}_2}(H)$.*

2 Preliminaries

Basics. Graphs in this paper are simple, undirected, and finite (even though our results transfer to directed graphs and even to weighted graphs). For a graph G, we write $V(G)$ for its vertex set and $E(G)$ for its edge set. For $v \in V(G)$, the set of neighbors of v are denoted with $N_G(v)$. For $S \subseteq V(G)$, we denote with $G[S]$ the subgraph of G induced by the vertices of S. A *rooted graph* is a graph G together with a designated root vertex $r(G) \in V(G)$. We write multisets using the notation $\{\!\{1,1,6,2\}\!\}$.

Matrices. An LU-decomposition of a matrix A consists of a lower triangular matrix L and an upper triangular matrix U such that $A = LU$ holds. Every finite matrix A over $\mathbf{R}$ has an LU-decomposition. We also use *infinite matrices* over $\mathbf{R}$, which are functions $A : I \times J \to \mathbf{R}$ where I and J are locally finite posets and countable. The matrix product AB is defined in the natural way via $(AB)_{ij} = \sum_k A_{ik}B_{kj}$ if all of these inner products are finite sums, and otherwise we leave it undefined. An $n \times n$ real symmetric matrix has real eigenvalues and a corresponding set of orthogonal eigenspaces. The spectral decomposition of a real symmetric matrix M is of the form $M = \lambda_1 P_1 + \cdots + \lambda_l P_l$ where $\lambda_1, \ldots, \lambda_l$ are the eigenvalues of M with corresponding eigenspaces $W_1, \ldots, W_l$. Moreover, each P_j is the projection matrix corresponding to the projection onto the eigenspace W_j. Usually, P_j is expressed as $P_j = UU^T$ for a matrix U whose columns form an orthonormal basis of W_j.

Homomorphism numbers. Recall that a mapping $h : V(F) \to V(G)$ is a homomorphism if $h(e) \in E(G)$ holds for all $e \in E(F)$ and that $\mathsf{Hom}(F,G)$ is the number of homomorphisms from F to G. Let $\mathsf{Surj}(F,G)$ be the number of homomorphisms from F to G that are surjective on both the vertices and edges of G. Let $\mathsf{Inj}(F,G)$ be the number of injective homomorphisms from F to G. Let $\mathsf{Sub}(F,G) = \mathsf{Inj}(F,G)/\mathsf{Aut}(F)$, where $\mathsf{Aut}(F)$ is the number of automorphisms of F. Observe that $\mathsf{Sub}(F,G)$ is the number of subgraphs of G that are isomorphic to F. Where convenient, we view the objects Hom, Surj, and Inj as infinite matrices; the matrix indices are all unlabeled graphs, sorted by their size. However, we only use one representative of each isomorphism class, called the *isomorphism type* of the graphs in the class, as an index in the matrix. Then Surj is lower triangular and Inj is upper triangular, so $\mathsf{Hom} = \mathsf{Surj} \cdot \mathsf{Sub}$ is an LU-decomposition of Hom. Finally, $\mathsf{Ind}(F,G)$ is

the number of times F occurs as an induced subgraph in G. Similarly to the homomorphism vectors $\mathsf{HOM}_{\mathcal{F}}(G)$ we define vectors $\mathsf{INJ}_{\mathcal{F}}(G)$ and $\mathsf{IND}_{\mathcal{F}}(G)$. Finally, let G, H be rooted graphs. A homomorphism from G to H is a graph homomorphism that maps the root of G to the root of H. Moreover, two rooted graphs are isomorphic if there is an isomorphism mapping the root to the root.

3 Homomorphisms from trees

3.1 Color refinement and tree unfolding

Color refinement iteratively colors the vertices of a graph in a sequence of *refinement rounds*. Initially, all vertices get the same color. In each refinement round, any two vertices v and w that still have the same color get different colors if there is some color c such that v and w have a different number of neighbors of color c; otherwise they keep the same color. We stop the refinement process if the vertex partition that is induced by the colors does not change anymore, that is, all pairs of vertices that have the same color before the refinement round still have the same color after the round. More formally, we define the sequence $C_0^G, C_1^G, C_2^G, \dots$ of colorings as follows. We let $C_0^G(v) = 1$ for all $v \in V(G)$, and for $i \geq 0$ we let $C_{i+1}^G(v) = \{\!\{ C_i^G(u) : u \in N_G(v) \}\!\}$. We say that color refinement *distinguishes* two graphs G and H if there is an $i \geq 0$ with

$$\{\!\{ C_i^G(v) : v \in V(G) \}\!\} \neq \{\!\{ C_i^H(v) : v \in V(H) \}\!\}. \tag{1}$$

We argue now that the color refinement algorithm implicitly constructs a tree at v obtained by simultaneously taking all possible walks starting at v (and not remembering nodes visited in the past). For a rooted tree T with root r, a graph G, and a vertex $v \in V(G)$, we say that T is *a tree at* v if there is a homomorphism f from T to G such that $f(r) = v$ and, for all non-leaves $t \in V(T)$, the function f induces a bijection between the set of children of t in in T and the set of neighbors of $f(t)$ in G. In other words, f is a homomorphism from T to G that is *locally bijective*. If T is an infinite tree at v and does not have any leaves, then T is uniquely determined up to isomorphisms, and we call this *the infinite tree at* v (or the *tree unfolding* of G at v), denoted with $T(G, v)$. For an infinite rooted tree T, let $T_{\leq d}$ be the finite rooted subtree of T where all leaves are at depth exactly d. For all finite trees T of depth d, define $\mathsf{Cr}(T, G) \in \{0, \dots, |V(G)|\}$ to be the number of vertices $v \in V(G)$ for which T is isomorphic to $T(G, v)_{\leq d}$. Note that this number is zero if not all leaves of T are at the same depth d or if some node of T has more than $n - 1$ children. The CR-vector of G is the vector $\mathsf{CR}(G) = (\mathsf{Cr}(T, G))_{T \in \mathcal{T}_r}$, where $\mathcal{T}_r$ denotes the family of all rooted trees. The following connection between the color refinement algorithm and the CR-vector is known.

▶ **Lemma 6** (Angluin [2], also see Krebs and Verbitsky [15, Lemma 2.5])**.** *For all graphs G and H, color refinement distinguishes G and H if and only if $\mathsf{CR}(G) \neq \mathsf{CR}(H)$ holds.*

3.2 Proof of Theorem 1

Throughout this section, we work with rooted trees. For a rooted tree T and an (unrooted) graph G, we simply let $\mathsf{Hom}(T, G)$ be the number of homomorphisms of the plain tree underlying T to G, ignoring the root.

Let T and T' be rooted trees. A homomorphism h from T to T' is *depth-preserving* if, for all vertices $v \in V(T)$, the depth of v in T is equal to the depth of $h(v)$ in T'. Moreover, a homomorphism h from T to T' is *depth-surjective* if the image of T under h contains vertices

at every depth present in T'. We define $\overrightarrow{\mathsf{Hom}}(T,T')$ as the number of homomorphisms from T to T' that are both depth-preserving and depth-surjective. Note that $\overrightarrow{\mathsf{Hom}}(T,T') = 0$ holds if and only if T and T' have different depths.

▶ **Lemma 7.** *Let T be a rooted tree and let G be a graph. We have*

$$\mathsf{Hom}(T,G) = \sum_{T'} \overrightarrow{\mathsf{Hom}}(T,T') \cdot \mathsf{Cr}(T',G)\,, \tag{2}$$

where the sum is over all unlabeled rooted trees T'. In other words, the matrix identity $\mathsf{Hom} = \overrightarrow{\mathsf{Hom}} \cdot \mathsf{Cr}$ holds.

Proof. Let d be the depth of T and let r be the root of T. Every T' with $\overrightarrow{\mathsf{Hom}}(T,T') \neq 0$ has depth d too and there are at most n non-isomorphic rooted trees T' of depth d with $\mathsf{Cr}(T',G) \neq 0$. Thus the sum in (2) has only finitely many non-zero terms and is well-defined.

For a rooted tree T' and a vertex $v \in V(G)$, let $H(T',v)$ be the set of all homomorphisms h from T to G such that $h(r) = v$ holds and the tree unfolding $T(G,v)_{\leq d}$ is isomorphic to T'. Let $H(T') = \bigcup_{v \in V(G)} H(T',v)$ and observe $|H(T',v)| = \overrightarrow{\mathsf{Hom}}(T,T')$. Since $\mathsf{Cr}(T',G)$ is the number of $v \in V(G)$ with $T(G,v)_{\leq d} \cong T'$, we thus have $|H(T')| = \overrightarrow{\mathsf{Hom}}(T,T') \cdot \mathsf{Cr}(T',G)$. Since each homomorphism from T to G is contained in exactly one set $H(T')$, we obtain the desired equality (2). ◀

For rooted trees T and T', let $\overrightarrow{\mathsf{Surj}}(T,T')$ be the number of depth-preserving and surjective homomorphisms from T to T'. In particular, not only do these homomorphisms have to be depth-surjective, but they should hit every vertex of T'. For rooted trees T and T' of the same depth, let $\overrightarrow{\mathsf{Sub}}(T,T')$ be the number of subgraphs of T' that are isomorphic to T (under an isomorphism that maps the root to the root); if T and T' have different depths, we set $\overrightarrow{\mathsf{Sub}}(T,T') = 0$.

▶ **Lemma 8.** $\overrightarrow{\mathsf{Hom}} = \overrightarrow{\mathsf{Surj}} \cdot \overrightarrow{\mathsf{Sub}}$ *is an LU-decomposition of* $\overrightarrow{\mathsf{Hom}}$*, and* $\overrightarrow{\mathsf{Surj}}$ *and* $\overrightarrow{\mathsf{Sub}}$ *are invertible.*

As is the case for finite matrices, the inverse of a lower (upper) triangular matrix is lower (upper) triangular. As the matrix $\overrightarrow{\mathsf{Surj}}$ is lower triangular and the matrix $\overrightarrow{\mathsf{Sub}}$ is upper triangular, their inverses are as well. We are ready to prove our first main theorem.

Proof of Theorem 1. We only need to prove the equivalence between assertions i and ii. For every graph G, let $\mathsf{HOM}_r(G) := \big(\mathsf{Hom}(T,G)\big)_{T \in \mathcal{T}_r}$. By our convention that for a rooted tree T and an unrooted graph G we let $\mathsf{Hom}(T,G)$ be the number of homomorphisms of the plain tree underlying T to G, for all G and H we have $\mathsf{HOM}_r(G) = \mathsf{HOM}_r(H) \iff \mathsf{HOM}(G) = \mathsf{HOM}(H)$. By Lemma 6, it suffices to prove for all graph G, H that

$$\mathsf{CR}(G) = \mathsf{CR}(H) \iff \mathsf{HOM}_r(G) = \mathsf{HOM}_r(H)\,. \tag{3}$$

We view the vectors $\mathsf{HOM}_r(G)$ and $\mathsf{CR}(G)$ as infinite column vectors. By Lemma 7, we have

$$\mathsf{HOM}_r(G) = \overrightarrow{\mathsf{Hom}} \cdot \mathsf{CR}(G) \text{ and } \mathsf{HOM}_r(H) = \overrightarrow{\mathsf{Hom}} \cdot \mathsf{CR}(H)\,. \tag{4}$$

The forward direction of (3) now follows immediately.

It remains to prove the backward direction. Since $\overrightarrow{\mathsf{Hom}} = \overrightarrow{\mathsf{Surj}} \cdot \overrightarrow{\mathsf{Sub}}$ holds by Lemma 8 for two invertible matrices $\overrightarrow{\mathsf{Surj}}$ and $\overrightarrow{\mathsf{Sub}}$, we can first left-multiply with $\overrightarrow{\mathsf{Surj}}^{-1}$ to obtain the equivalent identities

$$\overrightarrow{\mathsf{Surj}}^{-1} \cdot \mathsf{HOM}_r(G) = \overrightarrow{\mathsf{Sub}} \cdot \mathsf{CR}(G) \text{ and } \overrightarrow{\mathsf{Surj}}^{-1} \cdot \mathsf{HOM}_r(H) = \overrightarrow{\mathsf{Sub}} \cdot \mathsf{CR}(H)\,. \tag{5}$$

Now suppose $\mathsf{HOM}_r(G) = \mathsf{HOM}_r(H)$ holds, and set $\boldsymbol{v} = \mathsf{HOM}_r(G)$. Then $\overrightarrow{\mathsf{Surj}}^{-1} \cdot \boldsymbol{v}$ is well-defined, because $\overrightarrow{\mathsf{Surj}}$ and its inverse are lower triangular. Thus we obtain $\overrightarrow{\mathsf{Sub}} \cdot \mathsf{CR}(G) = \overrightarrow{\mathsf{Sub}} \cdot \mathsf{CR}(H)$ and set $\boldsymbol{w} = \mathsf{CR}(G)$. Unfortunately, $\overrightarrow{\mathsf{Sub}}^{-1} \cdot \boldsymbol{w}$ may be undefined, since $\overrightarrow{\mathsf{Sub}}^{-1}$ is upper triangular. While we can still use a matrix inverse, the argument becomes a bit subtle. The crucial observation is that $\mathsf{Cr}(T', G)$ is non-zero for at most n different trees T', and all such trees have maximum degree at most $n-1$. Thus we do not need to look at *all* trees but only those with maximum degree n. Let $\widetilde{\mathcal{T}}$ be the set of all unlabeled rooted trees of maximum degree at most n. Let $\mathsf{CR}' = \mathsf{CR}|_{\widetilde{\mathcal{T}}}$, let $\boldsymbol{w}' = \boldsymbol{w}|_{\widetilde{\mathcal{T}}}$, and let $\overrightarrow{\mathsf{Sub}}' = \overrightarrow{\mathsf{Sub}}|_{\widetilde{\mathcal{T}} \times \widetilde{\mathcal{T}}}$. Then we still have the following for all $T \in \widetilde{\mathcal{T}}$ and G:

$$\boldsymbol{w}'_T = \sum_{T' \in \widetilde{\mathcal{T}}} \overrightarrow{\mathsf{Sub}}'(T, T') \cdot \mathsf{Cr}'(T', G)\,. \tag{6}$$

The new matrix $\overrightarrow{\mathsf{Sub}}'$ is a principal minor of $\overrightarrow{\mathsf{Sub}}$ and thus remains invertible. Moreover, $\overrightarrow{\mathsf{Sub}}'^{-1} \cdot \boldsymbol{w}'$ is well-defined, since

$$\sum_{T' \in \widetilde{\mathcal{T}}} \overrightarrow{\mathsf{Sub}}'^{-1}(T, T') \cdot \boldsymbol{w}'_{T'} \tag{7}$$

is a finite sum for each T: The number of (unlabeled) trees $T' \in \widetilde{\mathcal{T}}$ that have the same depth d as T is bounded by a function in n and d. Thus $\overrightarrow{\mathsf{Sub}}'^{-1} \cdot \boldsymbol{w}' = \mathsf{CR}'(G)$. By a similar argument, we obtain $\overrightarrow{\mathsf{Sub}}'^{-1} \cdot \boldsymbol{w}' = \mathsf{CR}'(H)$. This implies $\mathsf{CR}'(G) = \mathsf{CR}'(H)$ and thus $\mathsf{CR}(G) = \mathsf{CR}(H)$. ◀

4 Homomorphisms from cycles and paths

While the arguments we saw in the proof of Theorem 1 are mainly graph-theoretic, the proof of Theorem 2 uses spectral techniques. To introduce the techniques, we first prove a simple, known result already mentioned in the introduction. We call two square matrices *co-spectral* if they have the same eigenvalues with the same multiplicities, and we call two graphs *co-spectral* if their adjacency matrices are co-spectral.

▶ **Proposition 9** (e.g. [23, Lemma 1])**.** *Let $\mathcal{C}$ be the class of all cycles (including the degenerate cycle of length 0, which is just a single vertex). For all graphs G and H, we have $\mathsf{HOM}_{\mathcal{C}}(G) = \mathsf{HOM}_{\mathcal{C}}(H)$ if and only if G and H are co-spectral.*

For the proof, we review a few simple facts from linear algebra. The trace $\operatorname{tr}(A)$ of a square matrix $A \in \mathbb{R}^{n \times n}$ is the sum of the diagonal entries. If the eigenvalues of A are $\lambda_1, \dots, \lambda_n$, then $\operatorname{tr}(A) = \sum_{i=1}^n \lambda_i$. Moreover, for each $\ell \geq 0$ the eigenvalues of the matrix A^ℓ are $\lambda_1^\ell, \dots, \lambda_n^\ell$, and thus $\operatorname{tr}(A^\ell) = \sum_{i=1}^n \lambda_i^\ell$. The following technical lemma encapsulates the fact that the information $\operatorname{tr}(A^\ell)$ for all $\ell \in \mathbf{N}$ suffices to reconstruct the spectrum of A with multiplicities. We use the same lemma to prove Theorem 2, but for Proposition 9 a less general version would suffice.

▶ **Lemma 10.** *Let $X, Y \subseteq \mathbf{R}$ be two finite sets and let $c \in \mathbf{R}^X_{\neq 0}$ and $d \in \mathbf{R}^Y_{\neq 0}$ be two vectors. If the equation*

$$\sum_{x \in X} c_x x^\ell = \sum_{y \in Y} d_y y^\ell \tag{8}$$

holds for all $\ell \in \mathbf{N}$, then $X = Y$ and $c = d$.

Proof. We prove the claim by induction on $k := |X| + |Y|$. For $k = 0$, the claim is trivially true since both sums in (8) are equal to zero by convention.

Let $\hat{x} = \arg\max\{\, |x| : x \in X \cup Y \,\}$ and let $\hat{x} \in X$ without loss of generality. If $\hat{x} = 0$, then $X = \{0\}$ and we claim that $Y = \{0\}$ holds. Clearly (8) for $\ell = 0$ yields $0 \neq c_0 = \sum_{y \in Y} d_y$. In particular, $Y \neq \emptyset$ holds. Since $\hat{x} = 0$ is the maximum of $X \cup Y$ in absolute value, we have $Y = \{0\}$ and thus also $c = d$.

Now suppose that $\hat{x} \neq 0$ holds. We consider the sequences $(a_\ell)_{\ell \in \mathbf{N}}$ and $(b_\ell)_{\ell \in \mathbf{N}}$ with

$$a_\ell = \frac{1}{\hat{x}^\ell} \cdot \sum_{x \in X} c_x x^\ell \quad \text{and} \quad b_\ell = \frac{1}{\hat{x}^\ell} \cdot \sum_{y \in Y} d_y y^\ell \,. \tag{9}$$

Note that $a_\ell = b_\ell$ holds for all $\ell \in \mathbf{N}$ by assumption. Observe the following simple facts:
1) If $-\hat{x} \notin X$, then $\lim_{\ell \to \infty} a_\ell = c_{\hat{x}}$.
2) If $-\hat{x} \in X$, then $\lim_{\ell \to \infty} a_{2\ell} = c_{\hat{x}} + c_{-\hat{x}}$ and $\lim_{\ell \to \infty} a_{2\ell+1} = c_{\hat{x}} - c_{-\hat{x}}$.
As well as the following exhaustive case distinction for Y:
a) If $\hat{x}, -\hat{x} \notin Y$, then $\lim_{\ell \to \infty} b_\ell = 0$.
b) If $\hat{x} \in Y$ and $-\hat{x} \notin Y$, then $\lim_{\ell \to \infty} b_\ell = d_{\hat{x}}$.
c) If $\hat{x} \notin Y$ and $-\hat{x} \in Y$, then $\lim_{\ell \to \infty} b_{2\ell} = d_{-\hat{x}}$ and $\lim_{\ell \to \infty} b_{2\ell+1} = -d_{-\hat{x}}$.
d) If $\hat{x}, -\hat{x} \in Y$, then $\lim_{\ell \to \infty} b_{2\ell} = d_{\hat{x}} + d_{-\hat{x}}$ and $\lim_{\ell \to \infty} b_{2\ell+1} = d_{\hat{x}} - d_{-\hat{x}}$.
If $-\hat{x} \notin X$ holds, we see from 1) that a_ℓ converges to the non-zero value $c_{\hat{x}}$. Since the two sequences are equal, the sequence b_ℓ also converges to a non-zero value. The only case for Y where this happens is b), and we get $\hat{x} \in Y$, $-\hat{x} \notin Y$, and $c_{\hat{x}} = d_{\hat{x}}$. On the other hand, if $-\hat{x} \in X$, we see from 2) that a_ℓ does not converge, but the even and odd subsequences do. The only cases for Y where this happens for b_ℓ too are c) and d). We cannot be in case c), since the two accumulation points of b_ℓ just differ in their sign, while the two accumulation points of a_ℓ do not have the same absolute value. Thus we must be in case d) and obtain $x, \hat{x} \in Y$ as well as

$$c_{\hat{x}} + c_{-\hat{x}} = d_{\hat{x}} + d_{-\hat{x}} \quad \text{and} \quad c_{\hat{x}} - c_{-\hat{x}} = d_{\hat{x}} - d_{-\hat{x}} \,.$$

This linear system has full rank and implies $c_{\hat{x}} = d_{\hat{x}}$ and $c_{-\hat{x}} = d_{-\hat{x}}$.

Either way, we can remove $\{\hat{x}\}$ or $\{\hat{x}, -\hat{x}\}$ from both X and Y and apply the induction hypothesis on the resulting instance X', Y', c', d'. Then $(X, c) = (Y, d)$ follows as claimed. ◀

Proof of Proposition 9. For all $\ell \geq 0$, the number of homomorphisms from the cycle C_ℓ of length ℓ to a graph G with adjacency matrix A is equal to the number of closed length-ℓ walks in G, which in turn is equal to the trace of A^ℓ. Thus for graphs G, H with adjacency matrices A, B, we have $\mathsf{HOM}_{\mathcal{C}}(G) = \mathsf{HOM}_{\mathcal{C}}(H)$ if and only if $\operatorname{tr}(A^\ell) = \operatorname{tr}(B^\ell)$ holds for all $\ell \geq 0$.

If A and B have the same spectrum $\lambda_1, \ldots, \lambda_n$, then $\operatorname{tr}(A^\ell) = \lambda_1^\ell + \cdots + \lambda_n^\ell = \operatorname{tr}(B^\ell)$ holds for all $\ell \in \mathbf{N}$. For the reverse direction, suppose $\operatorname{tr}(A^\ell) = \operatorname{tr}(B^\ell)$ for all $\ell \in \mathbf{N}$. Let $X \subseteq \mathbf{R}$ be the set of eigenvalues of A and for each $\lambda \in X$, let $c_\lambda \in \{1, \ldots, n\}$ be the multiplicity of the eigenvalue λ. Let $Y \subseteq \mathbf{R}$ and d_λ for $\lambda \in Y$ be the corresponding eigenvalues and multiplicities for B. Then for all $\ell \in \mathbf{N}$, we have

$$\sum_{\lambda \in X} c_\lambda \lambda^\ell = \operatorname{tr}(A^\ell) = \operatorname{tr}(B^\ell) = \sum_{\lambda \in Y} d_\lambda \lambda^\ell \,.$$

By Lemma 10, this implies $(X, c) = (Y, d)$, that is, the spectra of A and B are identical. ◀

In the following example, we show that the vectors $\mathsf{HOM}_{\mathcal{C}}$ for the class $\mathcal{C}$ of cycles and $\mathsf{HOM}_{\mathcal{T}}$ for the class $\mathcal{T}$ of trees are incomparable in their expressiveness.

Figure 2 Two co-spectral graphs

▶ **Example 11.** The graphs G and H shown in Figure 2 are co-spectral and thus $\mathsf{HOM}_{\mathcal{C}}(G) = \mathsf{HOM}_{\mathcal{C}}(H)$, but it is easy to see that $\mathsf{HOM}_{\mathcal{P}}(G) \neq \mathsf{HOM}_{\mathcal{P}}(H)$ for the class $\mathcal{P}$ of all paths.

Let G' be a cycle of length 6 and H' the disjoint union of two triangles. Then obviously, $\mathsf{HOM}_{\mathcal{C}}(G') \neq \mathsf{HOM}_{\mathcal{C}}(H')$. However, color refinement does not distinguish G' and H' and thus $\mathsf{HOM}_{\mathcal{T}}(G') = \mathsf{HOM}_{\mathcal{T}}(H')$.

Let us now turn to the proof of Theorem 2.

Proof of Theorem 2. Let A and B be the adjacency matrices of G and H, respectively. Since A is a symmetric and real matrix, its eigenvalues are real and the corresponding eigenspaces are orthogonal and span $\mathbf{R}^n$. Let $\mathbf{1}$ be the n-dimensional all-1 vector, and let $X = \{\lambda_1, \dots, \lambda_k\}$ be the set of all eigenvalues of A whose corresponding eigenspaces are not orthogonal to $\mathbf{1}$. We call these eigenvalues the *useful* eigenvalues of A and without loss of generality assume $\lambda_1 > \cdots > \lambda_k$. The n-dimensional all-1 vector $\mathbf{1}$ can be expressed as a direct sum of eigenvectors of A corresponding to useful eigenvalues. In particular, there is a unique decomposition $\mathbf{1} = \sum_{i=1}^{k} u_i$ such that each u_i is a non-zero eigenvector in the eigenspace of λ_i. Moreover, the vectors $u_1, \dots, u_k$ are orthogonal. For the matrix B, we analogously define its set of useful eigenvalues $Y = \{\mu_1, \dots, \mu_{k'}\}$ and the direct sum $\mathbf{1} = \sum_{i=1}^{k'} v_i$.

We prove the equivalence of the following three assertions (of which the first and third appear in the statement of Theorem 2).

1. $\mathsf{HOM}_{\mathcal{P}}(G) = \mathsf{HOM}_{\mathcal{P}}(H)$.
2. A and B have the same set of useful eigenvalues $\lambda_1, \dots, \lambda_k$ and $\|u_i\| = \|v_i\|$ holds for all $i \in \{1, \dots, k\}$. Here, $\|.\|$ denotes the Euclidean norm with $\|x\|^2 = \sum_j x_j^2$.
3. The system $\mathsf{F}_{\text{iso}}(G, H)$ of linear equations has a real solution.

Note that in 2, we do not require that the useful eigenvalues occur with the same multiplicities in A and B. We show the implications $(1 \Rightarrow 2)$, $(2 \Rightarrow 3)$, and $(3 \Rightarrow 1)$.

$(1 \Rightarrow 2)$: Suppose that $\mathsf{Hom}(P_\ell, G) = \mathsf{Hom}(P_\ell, H)$ holds for all paths P_ℓ. Equivalently, this can be stated in terms of the adjacency matrices A and B: for all $\ell \in \mathbf{N}$, we have $\mathbf{1}^T A^\ell \mathbf{1} = \mathbf{1}^T B^\ell \mathbf{1}$. We claim that A and B have the same useful eigenvalues, and that the projections of $\mathbf{1}$ onto the corresponding eigenspaces have the same lengths.

Note that $A^\ell \mathbf{1} = \sum_{i=1}^{k} \lambda_i^\ell u_i$ holds. Thus we have

$$\mathbf{1}^T A^\ell \mathbf{1} = \left(\sum_{i=1}^{k} u_i^T \right) \left(\sum_{i=1}^{k} \lambda_i^\ell u_i \right) = \sum_{i=1}^{k} \|u_i\|^2 \cdot \lambda_i^\ell \,. \tag{10}$$

The term $\mathbf{1}^T B^\ell \mathbf{1}$ can be expanded analogously, which together yields

$$\sum_{i=1}^{k} \|u_i\|^2 \cdot \lambda_i^\ell = \sum_{i=1}^{k'} \|v_i\|^2 \cdot \mu_i^\ell \quad \text{for all } \ell \in \mathbf{N}. \tag{11}$$

Since all coefficients $c_{\lambda_i} = \|u_i\|^2$ and $d_{\mu_i} = \|v_i\|^2$ are non-zero, we are in the situation of Lemma 10. We obtain $k = k'$ and, for all $i \in \{1, \dots, k\}$, we obtain $\lambda_i = \mu_i$ and $\|u_i\| = \|v_i\|$. This is exactly the claim that we want to show.

($2 \Rightarrow 3$): We claim that the $(n \times n)$-matrix X defined via

$$X = \sum_{i=1}^{k} \frac{1}{\|u_i\|^2} \cdot u_i v_i^T \tag{12}$$

satisfies the F_{iso} equations $AX = XB$ and $X\mathbf{1} = \mathbf{1} = X^T\mathbf{1}$. Indeed, we have

$$AX = \sum_{i=1}^{k} \frac{1}{\|u_i\|^2} \cdot Au_i v_i^T = \sum_{i=1}^{k} \frac{\lambda_i}{\|u_i\|^2} \cdot u_i v_i^T = \sum_{i=1}^{k} \frac{1}{\|u_i\|^2} \cdot u_i v_i^T B^T = XB^T = XB\,, \tag{13}$$

This follows, since $Au_i = \lambda_i u_i$, $Bv_i = \lambda_i v_i$, and B is symmetric. Moreover, we have

$$X\mathbf{1} = \sum_{i=1}^{k} \frac{1}{\|u_i\|^2} \cdot Au_i v_i^T \mathbf{1} = \sum_{i=1}^{k} \frac{1}{\|u_i\|^2} \cdot u_i v_i^T \sum_{j=1}^{k} v_j = \sum_{i=1}^{k} \frac{1}{\|u_i\|^2} \cdot u_i \cdot v_i^T v_i = \mathbf{1}\,. \tag{14}$$

This holds by definition of u_i and v_i and from $v_i^T v_i = \|v_i\|^2 = \|u_i\|^2$. The claim $X^T\mathbf{1} = \mathbf{1}$ follows analogously.

($3 \Rightarrow 1$): Suppose there is a matrix X with $X^T\mathbf{1} = X\mathbf{1} = \mathbf{1}$ and $AX = XB$. We obtain $A^\ell X = XB^\ell$ by induction for all $\ell \in \mathbf{N}_{>0}$. For $\ell = 0$, this also holds since $A^0 = I_n$ by convention. As a result, we have $\mathbf{1}^T A^\ell \mathbf{1} = \mathbf{1}^T A^\ell X\mathbf{1} = \mathbf{1}^T XB^\ell \mathbf{1} = \mathbf{1}^T B^\ell \mathbf{1}$ for all $\ell \in \mathbf{N}$. Since these scalars count the length-ℓ walks in G and H, respectively, we obtain $\mathsf{Hom}(P_\ell, G) = \mathsf{Hom}(P_\ell, H)$ for all paths P_ℓ as claimed. ◀

5 Homomorphisms from bounded tree width and path width

We briefly outline the main ideas of the proofs of Theorems 3 and 4; the technical details are deferred to the full version of this paper. In Theorem 3, the equivalence between ii and iii is essentially known, so we focus on the equivalence between i and ii. The proof is similar to the proof of Theorem 1 in Section 3.

Let us fix $k \geq 2$. The idea of the k-WL algorithm is to iteratively color k-tuples of vertices. Initially, each k-tuple $(v_1, \dots, v_k)$ is colored by its *atomic type*, that is, the isomorphism type of the labeled graph $G[\{v_1, \dots, v_k\}]$. Then in the refinement step, to define the new color of a k-tuple $\bar{v}$ we look at the current color of all k-tuples that can be reached from k by adding one vertex and then removing one vertex.

Similar to the tree unfolding of a graph G at a vertex v, we define the *Weisfeiler-Leman tree unfolding* at a k-tuple $\bar{v}$ of vertices. These objects have some resemblance to the pebbling comonad, which was defined by Abramsky, Dawar, and Wang [1] in the language of category theory. The WL-tree unfolding describes the color of $\bar{v}$ computed by k-WL; formally it may be a viewed as a pair (T, F) consisting of a graph F together with a "rooted" tree decomposition (potentially infinite, but again we cut it off at some finite depth). Similar to the numbers $\mathsf{Cr}(T, G)$ and the vector $\mathsf{CR}(G)$, we now have numbers $\mathsf{WL}((T, F), G)$ and a vector $\mathsf{WL}(G)$ such that $\mathsf{WL}(G) = \mathsf{WL}(H)$ holds if and only if k-WL does not distinguish G and H. Then we define a linear transformation Φ with $\mathsf{HOM}_{\mathcal{T}_k}(G) = \Phi\mathsf{WL}(G)$. The existence of this linear transformation directly yields the implication ii $\Longrightarrow$ i of Theorem 3. To prove the converse, we show that the transformation Φ is invertible by giving a suitable LU-decomposition of full rank. This completes our sketch of the proof of Theorem 3.

The proof of Theorem 4 requires a different argument, because now we have to use a solution (X_π) of the system $\mathsf{L}_{\text{iso}}^{k+1}(G,H)$ to prove that the path width k homomorphism vectors $\mathsf{HOM}_{\mathcal{P}_k}(G)$ and $\mathsf{HOM}_{\mathcal{P}_k}(H)$ are equal. The key idea is to express entries of a suitable variant of $\mathsf{HOM}_{\mathcal{P}_k}(G)$ as a linear combinations of entries of the corresponding vector for H using the values X_π as coefficients.

6 Conclusions

We have studied the homomorphism vectors $\mathsf{HOM}_{\mathcal{F}}(G)$ for various graph classes $\mathcal{F}$, focusing on classes $\mathcal{F}$ where it is tractable to compute the entries $\mathsf{Hom}(F,G)$ of the vector. Our main interest was in the "expressiveness" of these vectors, that is, in the question what $\mathsf{HOM}_{\mathcal{F}}(G)$ tells us about the graph G. For the classes $\mathcal{C}$ of cycles, $\mathcal{T}$ of trees, $\mathcal{T}_k$ of graphs of tree width at most k, and $\mathcal{P}$ of paths, we have obtained surprisingly clean answers to this question, relating the homomorphism vectors to various other well studied formalisms that on the surface have nothing to do with homomorphism counts.

Some interesting questions remain open. The most obvious is whether the converse of Theorem 4 holds, that is, whether for two graphs G, H with $\mathsf{HOM}_{\mathcal{P}_k}(G) = \mathsf{HOM}_{\mathcal{P}_k}(H)$, the system $\mathsf{L}_{\text{iso}}^{k+1}(G,H)$ has a real solution (and hence the Nullstellensatz propositional proof system has no degree-$(k+1)$ refutation of G and H being isomorphic).

Another related open problem in spectral graph theory is to characterize graphs which are identified by their spectrum, up to isomorphism. In our framework, Proposition 9 ensures that we can equivalently ask for the following characterization: for which graphs G does the vector $\mathsf{HOM}_{\mathcal{C}}(G)$ determine the entire homomorphism vector $\mathsf{HOM}(G)$?

Despite the computational intractability, it is also interesting to study the vectors $\mathsf{HOM}_{\mathcal{F}}(G)$ for classes $\mathcal{F}$ of unbounded tree width. Are there natural classes $\mathcal{F}$ (except of course the class of all graphs) for which the vectors $\mathsf{HOM}_{\mathcal{F}}(G)$ characterize G up to isomorphism? For example, what about classes of bounded degree or the class of planar graphs? And what is the complexity of deciding whether $\mathsf{HOM}_{\mathcal{F}}(G) = \mathsf{HOM}_{\mathcal{F}}(H)$ holds when G and H are given as input? Our results imply that this problem is in polynomial time for the classes $\mathcal{T}$, $\mathcal{T}_k$, and $\mathcal{P}$. For the class of all graphs, it is in quasi-polynomial time by Babai's quasi-polynomial isomorphism test [5]. Yet it seems plausible that there are classes $\mathcal{F}$ (even natural classes decidable in polynomial time) for which the problem is co-NP-hard.

Maybe the most interesting direction for further research is to study the graph similarity measures induced by homomorphism vectors. A simple way of defining an inner product on the homomorphism vectors is by letting

$$\Big\langle \mathsf{HOM}_{\mathcal{F}}(G), \mathsf{HOM}_{\mathcal{F}}(H) \Big\rangle := \sum_{\substack{k \geq 1 \\ \mathcal{F}_k \neq \emptyset}} \frac{1}{k^k |\mathcal{F}_k|} \sum_{F \in \mathcal{F}_k} \mathsf{Hom}(F,G)\mathsf{Hom}(F,H),$$

where $\mathcal{F}_k$ denotes the class of all graph $F \in \mathcal{F}$ with k vertices. The mapping $(G,H) \mapsto \langle \mathsf{HOM}_{\mathcal{F}}(G), \mathsf{HOM}_{\mathcal{F}}(H) \rangle$ is what is known as a *graph kernel* in machine learning. It induces a (pseudo)metric $d_{\mathcal{T}}$ on the class of graphs. It is an interesting question how it relates to other graph similarity measures, for example, the metric induced by the Weisfeiler-Leman graph kernel. Our Theorem 1 implies that the metric $d_{\mathcal{T}}$ for the class $\mathcal{T}$ of trees and the metric induced by the Weisfeiler-Leman graph kernel have the same graphs of distance zero.

References

1. Samson Abramsky, Anuj Dawar, and Pengming Wang. The pebbling comonad in finite model theory. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005129.
2. Dana Angluin. Local and global properties in networks of processors (extended abstract). In Raymond E. Miller, Seymour Ginsburg, Walter A. Burkhard, and Richard J. Lipton, editors, *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 82–93. ACM, 1980. doi:10.1145/800141.804655.
3. Vikraman Arvind, Johannes Köbler, Sebastian Kuhnert, and Yadu Vasudev. Approximate graph isomorphism. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science*, volume 7464 of *Lecture Notes in Computer Science*, pages 100–111. Springer, 2012. doi:10.1007/978-3-642-32589-2_12.
4. Albert Atserias and Elitza N. Maneva. Sherali–Adams relaxations and indistinguishability in counting logics. *SIAM Journal on Computing*, 42(1):112–137, 2013. doi:10.1137/120867834.
5. László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.
6. Paul Beame, Russell Impagliazzo, Jan Krajíček, Toniann Pitassi, and Pavel Pudlák. Lower bounds on Hilbert's Nullstellensatz and propositional proofs. *Proceedings of the London Mathematical Society*, s3-73(1):1–26, 1996. doi:10.1112/plms/s3-73.1.1.
7. Christoph Berkholz and Martin Grohe. Limitations of algebraic approaches to graph isomorphism testing. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 155–166. Springer Verlag, 2015. doi:10.1007/978-3-662-47672-7_13.
8. Samuel R. Buss. Lower bounds on Nullstellensatz proofs via designs. In *Proof Complexity and Feasible Arithmetics*, pages 59–71. American Mathematical Society, 1998.
9. Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3):265–298, 2004. doi:10.1142/S0218001404003228.
10. Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004. doi:10.1016/j.tcs.2004.08.008.
11. Martin Grohe, Kristian Kersting, Martin Mladenov, and Pascal Schweitzer. Color refinement and its applications. To appear. URL: https://lii.rwth-aachen.de/images/Mitarbeiter/pub/grohe/cr.pdf.
12. Martin Grohe and Martin Otto. Pebble games and linear equations. *Journal of Symbolic Logic*, 80(3):797–844, 2015. doi:10.1017/jsl.2015.28.
13. Martin Grohe and Wied Pakusa. The descriptive complexity of solving linear equation systems and its applications. In *Proceedings of the 32nd ACM-IEEE Symposium on Logic in Computer Science*, 2017. doi:10.1109/LICS.2017.8005081.
14. Neil Immerman and Eric Lander. Describing graphs: A first-order approach to graph canonization. In *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*, pages 59–81. Springer New York, 1990. doi:10.1007/978-1-4612-4478-3_5.
15. Andreas Krebs and Oleg Verbitsky. Universal covers, color refinement, and two-variable counting logic: Lower bounds for the depth. In *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 689–700. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.69.
16. László Lovász. Operations with structures. *Acta Mathematica Hungarica*, 18:321–328, 1967.
17. László Lovász. *Large Networks and Graph Limits*. American Mathematical Society, 2012.

18 Peter N. Malkin. Sherali–Adams relaxations of graph isomorphism polytopes. *Discrete Optimization*, 12:73–97, 2014. doi:10.1016/j.disopt.2014.01.004.

19 Viswanath Nagarajan and Maxim Sviridenko. On the maximum quadratic assignment problem. *Math. Oper. Res.*, 34(4):859–868, 2009. doi:10.1287/moor.1090.0418.

20 Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011. URL: https://dl.acm.org/citation.cfm?id=2078187.

21 Gottfried Tinhofer. Graph isomorphism and theorems of Birkhoff type. *Computing*, 36(4):285–300, 1986. doi:10.1007/BF02240204.

22 Gottfried Tinhofer. A note on compact graphs. *Discrete Applied Mathematics*, 30(2-3):253–264, 1991. doi:10.1016/0166-218X(91)90049-3.

23 Edwin R. Van Dam and Willem H. Haemers. Which graphs are determined by their spectrum? *Linear Algebra and its applications*, 373:241–272, 2003. doi:10.1016/S0024-3795(03)00483-X.

24 S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010. URL: https://portal.acm.org/citation.cfm?id=1859891.

Sample-Optimal Identity Testing with High Probability

Ilias Diakonikolas[1]
USC, Los Angeles, CA, USA
diakonik@usc.edu

Themis Gouleakis[2]
CSAIL,MIT, Cambridge, MA, USA
tgoule@mit.edu

John Peebles[3]
CSAIL,MIT, Cambridge, MA, USA
jpeebles@mit.edu

Eric Price
UT Austin, Austin, TX, USA
ecprice@cs.utexas.edu

Abstract

We study the problem of testing identity against a given distribution with a focus on the high confidence regime. More precisely, given samples from an unknown distribution p over n elements, an explicitly given distribution q, and parameters $0 < \varepsilon, \delta < 1$, we wish to distinguish, *with probability at least* $1 - \delta$, whether the distributions are identical versus ε-far in total variation distance. Most prior work focused on the case that $\delta = \Omega(1)$, for which the sample complexity of identity testing is known to be $\Theta(\sqrt{n}/\varepsilon^2)$. Given such an algorithm, one can achieve arbitrarily small values of δ via black-box amplification, which multiplies the required number of samples by $\Theta(\log(1/\delta))$.

We show that black-box amplification is suboptimal for any $\delta = o(1)$, and give a new identity tester that achieves the optimal sample complexity. Our new upper and lower bounds show that the optimal sample complexity of identity testing is

$$\Theta\left(\frac{1}{\varepsilon^2}\left(\sqrt{n\log(1/\delta)} + \log(1/\delta)\right)\right)$$

for any n, ε, and δ. For the special case of uniformity testing, where the given distribution is the uniform distribution U_n over the domain, our new tester is surprisingly simple: to test whether $p = U_n$ versus $d_{\mathrm{TV}}(p, U_n) \geq \varepsilon$, we simply threshold $d_{\mathrm{TV}}(\widehat{p}, U_n)$, where $\widehat{p}$ is the empirical probability distribution. The fact that this simple "plug-in" estimator is sample-optimal is surprising, even in the constant δ case. Indeed, it was believed that such a tester would not attain sublinear sample complexity even for constant values of ε and δ.

An important contribution of this work lies in the analysis techniques that we introduce in this context. First, we exploit an underlying strong convexity property to bound from below the expectation gap in the completeness and soundness cases. Second, we give a new, fast method for obtaining provably correct empirical estimates of the true worst-case failure probability for a broad class of uniformity testing statistics over all possible input distributions – including all previously studied statistics for this problem. We believe that our novel analysis techniques will be useful for other distribution testing problems as well.

[1] Supported by NSF Award CCF-1652862 (CAREER) and a Sloan Research Fellowship.
[2] Supported by NSF Awards, including No. CCF-1650733, CCF-1733808, and IIS-1741137
[3] Supported by NSF Awards, including No. CCF-1650733, CCF-1733808, and IIS-1741137

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 41; pp. 41:1–41:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Mathematics of computing → Probability and statistics

Keywords and phrases distribution testing, property testing, sample complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.41

Related Version The full version of this paper can be found at `https://eccc.weizmann.ac.il/report/2017/133/`.

1 Introduction

Distribution property testing [15, 4, 5], originating in statistical hypothesis testing [18, 17], studies problems of the form: given sample access to one or more unknown distributions, determine whether they satisfy some global property or are "far" from satisfying the property. (See Section 1.1 for a formal definition.) During the past two decades problems of this form have received significant attention within the computer science community. See [20, 6] for two recent surveys.

Research in this field has primarily centered on determining tight bounds on the sample complexity of testing various properties *in the constant probability of success regime*. That is, the testing algorithm must succeed with a probability of (say) at least 2/3. This constant confidence regime is fairly well understood. For a range of fundamental properties [19, 7, 22, 12, 11, 1, 10, 9] we now have *sample-optimal* testers that use provably optimal number of samples (up to constant factors) in this regime.

In sharp contrast, the high confidence regime – i.e., the case where the desired failure probability is subconstant – is poorly understood even for the most basic properties. For essentially all distribution property testing problems studied in the literature, the standard amplification method is the only way known to achieve a high confidence success probability. Amplification is a black-box method that can boost the success probability to any desired accuracy. However, using it increases the number of required samples beyond what is necessary to obtain constant confidence. Specifically, to achieve a high confidence success probability of $1 - \delta$ via amplification, the number of samples required increases by a factor of $\Theta(\log(1/\delta))$ compared to the constant confidence regime.

This discussion raises the following natural questions: *For a given distribution property testing problem, does black-box amplification give sample-optimal testers for obtaining a high confidence success probability? Specifically, is the* $\Theta(\log(1/\delta))$ *multiplicative increase in the sample size the best possible? If not, can we design testers that have optimal sample complexity in terms of all relevant problem parameters, including the error probability* δ*?*

We believe that these are fundamental questions that merit theoretical investigation in their own right. As Goldreich notes [14], "eliminating the error probability as a parameter does not allow to ask whether or not one may improve over the straightforward error reduction". From a practical perspective, understanding this high confidence regime is important to applications of hypothesis testing (e.g., in biology), because the failure probability δ of the test can be reported as a p-value. (The family of distribution testing algorithms with success probability $1 - \delta$ for a given problem is equivalent to the family of statistical tests whose p-value (probability of Type I error) and probability of Type II error are both at most δ.) Standard techniques for addressing the problem of multiple comparisons, such as Bonferroni correction, require vanishingly small p-values.

Perhaps surprisingly, with one exception [16], this basic problem has not been previously investigated in the finite sample regime. A conceptual contribution of this work is to raise this problem as a fundamental goal in distribution property testing. We note here that

the analogous question in the context of *distribution learning* has been intensely studied in statistics and probability theory (see, e.g., [23, 8]) and tight bounds are known in a range of settings.

1.1 Formal Framework

The focus of this work is on the task of identity testing, which is arguably *the* most fundamental distribution testing problem.

▶ **Definition 1** (Distribution Identity Testing Problem). Given a target distribution q with domain D of size n, parameters $0 < \varepsilon, \delta < 1$, and sample access to an unknown distribution p over the same domain, we want to distinguish *with probability at least* $1 - \delta$ between the following cases:

- Completeness: $p = q$.
- Soundness: $d_{\mathrm{TV}}(p, q) \geq \varepsilon$.

We call this the problem of (ε, δ) *testing identity to* q. The special case of q being uniform is known as *uniformity testing*. An algorithm that solves one of these problems will be called an (ε, δ)-*tester* for identity/uniformity.

Note that $d_{\mathrm{TV}}(p, q)$ denotes the total variation distance or statistical distance between distributions p and q, i.e., $d_{\mathrm{TV}}(p, q) \stackrel{\text{def}}{=} \frac{1}{2} \cdot \|p - q\|_1$. The goal is to characterize the sample complexity of the problem: i.e., the number of samples that are necessary and sufficient to correctly distinguish between the completeness and soundness cases with probability $1 - \delta$.

1.2 Our Results

Our main result is a complete characterization of the worst-case sample complexity of identity testing in the high confidence regime. For this problem, we show that black-box amplification is suboptimal for *any* $\delta = o(1)$, and give a new identity tester that achieves the optimal sample complexity:

▶ **Theorem 2** (Main Result). *There exists a computationally efficient* (ε, δ)*-identity tester for discrete distributions of support size* n *with sample complexity*

$$\Theta\left(\frac{1}{\varepsilon^2}\left(\sqrt{n\log(1/\delta)} + \log(1/\delta)\right)\right). \tag{1}$$

Moreover, this sample size is information-theoretically optimal, up to a constant factor, for all n, ε, δ.

As we explain in Section 1.3, [16] gave a tester that achieves the optimal sample complexity when the sample size is $o(n)$. However this tester *completely* fails with $\Omega(n)$ samples, as may be required when either ε or δ are sufficiently small. Theorem 2 provides a complete characterization of the worst-case sample complexity of the problem with a *single* statistic for all settings of parameters n, ε, δ.

Brief Overview of Techniques. To analyze our tester, we introduce two new techniques for the analysis of distribution testing statistics, which we describe in more detail in Section 1.4. Our techniques leverage a simple common property of numerous distribution testing statistics which does not seem to have been previously exploited in their analysis: their convexity. Our first technique crucially exploits an underlying strong convexity property to bound from below the expectation gap between the completeness and soundness cases.

We remark that this is a contrast to most known distribution testers where bounding the expectation gap is easy, and the challenge is in bounding the variance of the statistic.

Our second technique implies a new, fast method for obtaining empirical estimates of the true worst-case failure probability of any member of a broad class of uniformity testing statistics. This class includes all uniformity testing statistics studied in the literature. Critically, these estimates come with provable guarantees about the worst-case failure probability of the statistic over all possible input distributions, and have tunable additive error. We elaborate in Section 1.4.

1.3 Discussion and Prior Work

Uniformity testing is the first and one of the most well-studied problems in distribution testing [15, 19, 22, 12, 9]. As already mentioned, the literature has almost exclusively focused on the case of constant error probability δ. The first uniformity tester, introduced by Goldreich and Ron [15], counts the number of collisions among the samples and was shown to work with $O(\sqrt{n}/\varepsilon^4)$ samples [15]. A related tester proposed by Paninski [19], which relies on the number of distinct elements in the set of samples, was shown to have the optimal $m = \Theta(\sqrt{n}/\varepsilon^2)$ sample complexity, as long as $m = o(n)$. Recently, a chi-squared based tester was shown in [22, 12] to achieve the optimal $\Theta(\sqrt{n}/\varepsilon^2)$ sample complexity without any restrictions. Finally, the original collision-based tester of [15] was very recently shown to also achieve the optimal $\Theta(\sqrt{n}/\varepsilon^2)$ sample complexity [9]. Thus, the situation for constant values of δ is well understood.

The problem of identity testing against an arbitrary (explicitly given) distribution was studied in [3], who gave an $(\varepsilon, 1/3)$-tester with sample complexity $\tilde{O}(n^{1/2})/\mathrm{poly}(\varepsilon)$. The tight bound of $\Theta(n^{1/2}/\varepsilon^2)$ was first given in [22] using a chi-squared type tester (inspired by [7]). In subsequent work, a similar chi-squared tester that also achieves the same sample complexity bound was given in [1]. (We note that the [22, 1] testers have sub-optimal sample complexity in the high confidence regime, even for the case of uniformity.) In a related work, [12] obtained a reduction of identity to uniformity that preserves the sample complexity, up to a constant factor, in the constant error probability regime. More recently, Goldreich [13], building on [10], gave a different reduction of identity to uniformity that preserves the error probability. We use the latter reduction in this paper to obtain an optimal identity tester starting from our new optimal uniformity tester.

Since the sample complexity of identity testing is $\Theta(\sqrt{n}/\varepsilon^2)$ for $\delta = 1/3$ [22, 12], standard amplification gives a sample upper bound of $\Theta(\sqrt{n}\log(1/\delta)/\varepsilon^2)$ for this problem. It is not hard to observe that this naive bound cannot be optimal for all values of δ. For example, in the extreme case that $\delta = 2^{-\Theta(n)}$, this gives a sample complexity of $\Theta(n^{3/2}/\varepsilon^2)$. On the other hand, one can *learn* the underlying distribution (and therefore test for identity) with $O(n/\varepsilon^2)$ samples for such values of δ[4].

The case where $1 \gg \delta \gg 2^{-\Theta(n)}$ is more subtle, and it is not a priori clear how to improve upon naive amplification. Theorem 2 provides a smooth transition between the extremes of $\Theta(\sqrt{n}/\varepsilon^2)$ for constant δ and $\Theta(n/\varepsilon^2)$ for $\delta = 2^{-\Theta(n)}$. It thus provides a quadratic improvement in the dependence on δ over the naive bound for all $\delta \geq 2^{-\Theta(n)}$, and shows that this is the best possible. For $\delta < 2^{-\Theta(n)}$, it turns out that the additive $\Theta(\log(1/\delta)/\varepsilon^2)$ term is necessary, as outlined in Section 1.4, so learning the distribution is optimal for such values of δ.

[4] This follows from the fact that, for any distribution p over n elements, the empirical probability distribution $\widehat{p}_m$ obtained after $m = \Omega((n + \log(1/\delta))/\varepsilon^2)$ samples drawn from p is ε-close to p in total variation distance with probability at least $1 - \delta$.

We obtain the first sample-optimal *uniformity tester* for the high confidence regime. Our sample-optimal identity tester follows from our uniformity tester by applying the recent result of Goldreich [13], which provides a black-box reduction of identity to uniformity. We also show a matching information-theoretic lower bound on the sample complexity.

The sample-optimal uniformity tester we introduce is remarkably simple: to distinguish between the cases that p is the uniform distribution U_n over n elements versus $d_{\mathrm{TV}}(p, U_n) \geq \varepsilon$, we simply compute $d_{\mathrm{TV}}(\widehat{p}, U_n)$ for the empirical distribution $\widehat{p}$. The tester accepts that $p = U_n$ if the value of this statistic is below some well-chosen threshold, and rejects otherwise.

It should be noted that such a tester was not previously known to work with sub-learning sample complexity – i.e., fewer than $\Theta(n/\varepsilon^2)$ samples – even in the constant confidence regime. Surprisingly, in a literature with several different uniformity testers [15, 19, 22, 12], no one has previously used the empirical total variation distance. On the contrary, it would be natural to assume – as was suggested in [4, 5] – that this tester cannot possibly work. A likely reason for this is the following observation: When the sample size m is smaller than the domain size n, the empirical total variation distance is very far from the true distance to uniformity. This suggests that the empirical distance statistic gives little, if any, information in this setting.

Despite the above intuition, we prove that the natural "plug-in" estimator relying on the empirical distance from uniformity actually works for the following reason: the empirical distance from uniformity is *noticeably smaller* for the uniform distribution than for "far from uniform" distributions, *even with a sub-linear sample size*. Moreover, we obtain the stronger statement that the "plug-in" estimator is a sample-optimal uniformity tester for all parameters n, ε and δ.

In [16], it was shown that the distinct-elements tester of [19] achieves the optimal sample complexity of $m = \Theta(\sqrt{n \log(1/\delta)}/\varepsilon^2)$, as long as $m = o(n)$. When $m = \Omega(n)$, as is the case in many practically relevant settings (see, e.g., the Polish lottery example in [21] with $n < \sqrt{n}/\varepsilon^2 \ll n/\varepsilon^2$), this tester is known to fail completely even in the constant confidence regime. On the other hand, in such settings the sample size is *not* sufficiently large so that we can actually *learn* the underlying distribution.

It is important to note that *all* previously considered uniformity testers [15, 19, 22, 12] do not achieve the optimal sample complexity (as a function of all parameters, including δ), and this is *inherent*, i.e., not just a failure of previous analyses. Roughly speaking, since the collision statistic [15] and the chi-squared based statistic [22, 12] are not Lipschitz, it can be shown that their high-probability performance is poor. Specifically, in the completeness case ($p = U_n$), if many samples happen to land in the same bucket (domain element), these test statistics become quite large, leading to their suboptimal behavior *for all* $\delta = o(1)$. (For a formal justification, the reader is referred to Section V of [16]). On the other hand, the distinct-elements tester [19] does not work for $m = \omega(n)$. For example, if ε or δ are sufficiently small to necessitate $m \gg n \log n$, then typically all n domain elements will appear in both the completeness and soundness cases, hence the test statistic provides no information.

1.4 Our Techniques

1.4.1 Upper Bound for Uniformity Testing

We would like to show that the test statistic $d_{\mathrm{TV}}(\widehat{p}, U_n)$ is with high probability larger when $d_{\mathrm{TV}}(p, U_n) \geq \varepsilon$ than when $p = U_n$. We start by showing that among all possible alternative distributions p with $d_{\mathrm{TV}}(p, U_n) \geq \varepsilon$, it suffices to consider those in a very simple family. We then show that the test statistic is highly concentrated around its expectation, and that the expectations are significantly different in the two cases. The main technical components of our paper are our techniques for accomplishing these tasks.

To simplify the structure of p, it can be shown that if p majorizes another distribution q, then the test statistic $d_{\mathrm{TV}}(\widehat{p}, U_n)$ stochastically dominates $d_{\mathrm{TV}}(\widehat{q}, U_n)$. (In fact, this statement holds for *any* test statistic that is a convex symmetric function of the empirical histogram.) We defer this proof to the full version. Therefore, for any p, if we average out the large and small entries of p, the test statistic becomes harder to distinguish from uniform.

We remark as a matter of independent interest that this stochastic domination lemma immediately implies a fast algorithm for performing rigorous empirical comparisons of test statistics. A major difficulty in empirical studies of distribution testing is that it is not possible to directly check the failure probability of a tester over every possible distribution as input, because the space of such distributions is quite large. Our structural lemma reduces the search space dramatically for uniformity testing: for any convex symmetric test statistic (which includes all existing ones), the worst case distribution will have αn coordinates of value $(1+\varepsilon/\alpha)/n$, and the rest of value $(1-\varepsilon/(1-\alpha))/n$, for some α. Hence, there are only n possible worst-case distributions for any ε. Notably, this reduction does not lose anything, so it could be used to identify the non-asymptotic optimal constants that a distribution testing statistic achieves for a given set of parameters.

Returning to our uniformity tester, at the cost of a constant factor in ε we can assume $\alpha = 1/2$. As a result, we only need to consider p to be either U_n or of the form $\frac{1\pm\varepsilon}{n}$ in each coordinate. We now need to separate the expectation of the test statistic in these two situations. The challenge is that both expectations are large, and we do not have a good analytic handle on them. We therefore introduce a new technique for showing a separation between the completeness and soundness cases that utilizes the strong convexity of the test statistic. Specifically, we obtain an explicit expression for the Hessian of the expectation, as a function of p. The Hessian is diagonal, and for our two situations of $p_i \approx 1/n$ each entry is within constant factors of the same value, giving a lower bound on its eigenvalues. Since the expectation is minimized at $p = U_n$, strong convexity implies an expectation gap. Specifically, we prove that this gap is $\varepsilon^2 \cdot \min(m^2/n^2, \sqrt{m/n}, 1/\varepsilon)$.

Finally, we need to show that the test statistic concentrates about its expectation. For $m \geq n$, this follows from McDiarmid's inequality: since the test statistic is $1/m$-Lipschitz in the m samples, with probability $1-\delta$ it lies within $\sqrt{\log(1/\delta)/m}$ of its expectation. When m is larger than the desired sample complexity given in (1), this is less than the expectation gap above. The concentration is trickier when $m < n$, since the expectation gap is smaller, so we need to establish tighter concentration. We get this by using a Bernstein variant of McDiarmid's inequality, which is stronger than the standard version of McDiarmid in this context. We note that the use of the stochastic domination is also crucial here. Since our statistic is a symmetric convex function of the histogram values, we show (details deferred to the full version) that for each soundness case distribution, there is a different distribution that has possible probability mass values exclusively in the set $\{\frac{1+\varepsilon'}{n}, \frac{1}{n}, \frac{1-\varepsilon'}{n}\}$, for some $\varepsilon' = O(\varepsilon)$, and is harder to distinguish from the uniform distribution. However, we could show that this distribution has a stronger Lipschitz-type property than the other soundness case distributions. Therefore, we are able to use a stronger concentration bound via McDiarmid's inequality and argue that even though other soundness case distributions may have weaker concentration, they still have smaller error due to our stochastic domination argument.

1.4.2 Upper Bound for Identity Testing

In [13], it was shown how to reduce ε-testing of an arbitrary distribution q over $[n]$ to $\varepsilon/3$-testing of U_{6n}. This reduction preserves the error probability δ, so applying it gives an identity tester with the same sample complexity as our uniformity tester, up to constant

factors.

1.4.3 Sample Complexity Lower Bound

To match our upper bound (1), we need two lower bounds. The lower bound of $\Omega(\frac{1}{\varepsilon^2}\log(1/\delta))$ is straightforward from the same lower bound as for distinguishing a fair coin from an ε-biased coin, while the $\sqrt{n\log(1/\delta)}/\varepsilon^2$ bound is more challenging.

For intuition, we start with a $\sqrt{n\log(1/\delta)}$ lower bound for constant ε. When $p = U_n$, the chance that all m samples are distinct is at least $(1-m/n)^m \approx e^{-m^2/n}$. Hence, if $m \ll \sqrt{n\log(1/\delta)}$, this would happen with probability significantly larger than 2δ. On the other hand, if p is uniform over a random subset of $n/2$ coordinates, the m samples will also all be distinct with probability $(1-2m/n)^m > 2\delta$. The two situations thus look the same with 2δ probability, so no tester could have accuracy $1-\delta$.

This intuition can easily be extended to include a $1/\varepsilon$ dependence, but getting the desired $1/\varepsilon^2$ dependence requires more work. First, we Poissonize the number of samples, so we independently see $\mathrm{Poi}(mp_i)$ samples of each coordinate i; with exponentially high probability, this Poissonization only affects the sample complexity by constant factors. Then, in the alternative hypothesis, we set each p_i independently at random to be $\frac{1\pm\varepsilon}{n}$. This has the unfortunate property that p no longer sums to 1, so it is a "pseudo-distribution" rather than an actual distribution. Still, it is exponentially likely to sum to $\Theta(1)$, and using techniques from [24, 10] this is sufficient for our purposes.

At this point, we are considering a situation where the number of times we see each coordinate is either $\mathrm{Poi}(m/n)$ or $\frac{1}{2}(\mathrm{Poi}((1-\varepsilon)\frac{m}{n}) + \mathrm{Poi}((1+\varepsilon)\frac{m}{n}))$, and every coordinate is independent of the others. These two distributions have Hellinger distance at least $\varepsilon^2 m/n$ in each coordinate. Then the composition property for Hellinger distance over n independent coordinates implies $m \geq \sqrt{n\log(1/\delta)}/\varepsilon^2$ is necessary for success probability $1-\delta$.

1.5 Notation

We write $[n]$ to denote the set $\{1,\ldots,n\}$. We consider discrete distributions over $[n]$, which are functions $p : [n] \to [0,1]$ such that $\sum_{i=1}^n p_i = 1$. We use the notation p_i to denote the probability of element i in distribution p. For $S \subseteq [n]$, we will denote $p(S) = \sum_{i\in S} p_i$. We will also sometimes think of p as an n-dimensional vector. We will denote by U_n the uniform distribution over $[n]$.

For $r \geq 1$, the ℓ_r-norm of a distribution is identified with the ℓ_r-norm of the corresponding vector, i.e., $\|p\|_r = (\sum_{i=1}^n |p_i|^r)^{1/r}$. The ℓ_r-distance between distributions p and q is defined as the the ℓ_r-norm of the vector of their difference. The total variation distance between distributions p and q is defined as $d_{\mathrm{TV}}(p,q) \stackrel{\mathrm{def}}{=} \max_{S\subseteq[n]} |p(S)-q(S)| = (1/2)\cdot\|p-q\|_1$. The Hellinger distance between p and q is $H(p,q) \stackrel{\mathrm{def}}{=} (1/\sqrt{2})\cdot\|\sqrt{p}-\sqrt{q}\|_2 = (1/\sqrt{2})\cdot\sqrt{\sum_{i=1}^n(\sqrt{p_i}-\sqrt{q_i})^2}$. We denote by $\mathrm{Poi}(\lambda)$ the Poisson distribution with parameter λ.

2 Sample-Optimal Uniformity Testing

In this section, we describe and analyze our optimal uniformity tester. Given samples from an unknown distribution p over $[n]$, our tester returns "YES" with probability $1-\delta$ if $p = U_n$, and "NO" with probability $1-\delta$ if $d_{\mathrm{TV}}(p,U_n) \geq \varepsilon$.

2.1 Our Test Statistic

We define a very natural statistic that yields a uniformity tester with optimal dependence on the domain size n, the proximity parameter ε, and the error probability δ. Our statistic is a thresholded version of the empirical total variation distance between the unknown distribution p and the uniform distribution. Our tester Test-Uniformity is described in the following pseudocode:

Algorithm Test-Uniformity$(p, n, \varepsilon, \delta)$
Input: sample access to a distribution p over $[n]$, $\varepsilon > 0$, and $\delta > 0$.
Output: "YES" if $p = U_n$; "NO" if $d_{\mathrm{TV}}(p, U_n) \geq \varepsilon$.

1. Draw $m = \Theta\left((1/\varepsilon^2) \cdot \left(\sqrt{n \log(1/\delta)} + \log(1/\delta)\right)\right)$ i.i.d. samples from p.
2. Let $X = (X_1, X_2, \ldots, X_n) \in \mathbb{Z}^n_{>0}$ be the histogram of the samples. That is, X_i is the number of times domain element i appears in the (multi-)set of samples.
3. Define the random variable $S = \frac{1}{2} \sum_{i=1}^{n} \left| \frac{X_i}{m} - \frac{1}{n} \right|$ and set the threshold

$$t = \mu(U_n) + C \cdot \begin{cases} \varepsilon^2 \cdot \frac{m^2}{n^2} & \text{for } m \leq n \\ \varepsilon^2 \cdot \sqrt{\frac{m}{n}} & \text{for } n < m \leq \frac{n}{\varepsilon^2} \\ \varepsilon & \text{for } \frac{n}{\varepsilon^2} \leq m \end{cases},$$

   where C is a universal constant (derived from the analysis of the algorithm), and $\mu(U_n)$ is the expected value of the statistic in the completeness case. (We can compute $\mu(U_n)$ in $O(m)$ time using the procedure in Appendix A of the full version.)
4. If $S \geq t$ return "NO"; otherwise, return "YES".

The main part of this section is devoted to the analysis of Test-Uniformity, establishing the following theorem:

▶ **Theorem 3.** *There exists a universal constant $C > 0$ such that the following holds: Given*

$$m \geq C \cdot (1/\varepsilon^2) \left(\sqrt{n \log(1/\delta)} + \log(1/\delta)\right)$$

samples from an unknown distribution p, Algorithm Test-Uniformity *is an (ε, δ)-tester for uniformity of distribution p.*

As we point out in Appendix A of the full version, the value $\mu(U_n)$ can be computed efficiently, hence our overall tester is computationally efficient. To prove correctness of the above tester, we need to show that the expected value of the statistic in the completeness case is sufficiently separated from the expected value in the soundness case, and also that the value of the statistic is highly concentrated around its expectation in both cases. In Section 2.2, we bound from below the difference in the expectation of our statistic in the completeness and soundness cases. The proof the desired concentration, which completes the proof of Theorem 3, is deferred to the full version.

2.2 Bounding the Expectation Gap

The expectation of the statistic in algorithm Test-Uniformity can be viewed as a function of the n variables $p_1, \ldots, p_n$. We denote this expectation by $\mu(p) \stackrel{\text{def}}{=} \mathbb{E}[S(X_1, \ldots, X_n)]$ when the samples are drawn from distribution p.

Our analysis has a number of complications for the following reason: the function $\mu(p) - \mu(U_n)$ is a linear combination of sums that have no indefinite closed form, even if the distribution p assigns only two possible probabilities to the elements of the domain. This statement is made precise in Appendix B of the full version. As such, we should only hope to obtain an approximation of this quantity.

A natural approach to try and obtain such an approximation would be to produce separate closed form approximations for $\mu(p)$ and $\mu(U_n)$, and combine these quantities to obtain an approximation for their difference. However, one should not expect such an approach to work in our context. The reason is that the difference $\mu(p) - \mu(U_n)$ can be much smaller than $\mu(p)$ and $\mu(U_n)$; it can even be arbitrarily small. As such, obtaining separate approximations of $\mu(p)$ and $\mu(U_n)$ to any fixed accuracy would contribute too much error to their difference.

To overcome these difficulties, we introduce the following technique, which is novel in this context. We directly bound from below the difference $\mu(p) - \mu(U_n)$ using *strong convexity*. Specifically, we show that the function μ is strongly convex with appropriate parameters and use this fact to bound the desired expectation gap. The main result of this section is the following lemma:

▶ **Lemma 4.** *Let p be a distribution over $[n]$ and $\varepsilon = d_{\mathrm{TV}}(p, U_n)$. For all $m \geq 6$ and $n \geq 2$, we have that:*

$$\mu(p) - \mu(U_n) \geq \Theta(1) \cdot \begin{cases} \varepsilon^2 \cdot \frac{m^2}{n^2} & \text{for } m \leq n \\ \varepsilon^2 \cdot \sqrt{\frac{m}{n}} & \text{for } n < m \leq \frac{n}{\varepsilon^2} \\ \varepsilon & \text{for } \frac{n}{\varepsilon^2} \leq m \end{cases} .$$

We note that the bounds in the right hand side above are tight, up to constant factors. Any asymptotic improvement would yield a uniformity tester with sample complexity that violates our tight information-theoretic lower bounds.

The proof of Lemma 4 (which will be deferred to the full version) requires a couple of important intermediate lemmas. Our starting point is as follows: By the intermediate value theorem, we have the quadratic expansion

$$\mu(p) = \mu(U_n) + \nabla\mu(U_n)^\intercal (p - U_n) + \frac{1}{2}(p - U_n)^\intercal H_{p'} (p - U_n) \,,$$

where $H_{p'}$ is the Hessian matrix of the function μ at some point p' which lies on the line segment between U_n and p. This expression can be simplified as follows: First, we show that our μ is minimized over all probability distributions on input U_n (see the full version of the paper for details). Thus, the gradient $\nabla\mu(U_n)$ must be orthogonal to being a direction in the space of probability distributions. In other words, $\nabla\mu(U_n)$ must be proportional to the all-ones vector. More formally, since μ is symmetric its gradient is a symmetric function, which implies it will be symmetric when given symmetric input. Moreover, $(p - U_n)$ is a direction within the space of probability distributions, and therefore sums to 0, making it orthogonal to the all-ones vector. Thus, we have that $\nabla\mu(U_n)^\intercal (p - U_n) = 0$, and we obtain

$$\mu(p) - \mu(U_n) = \frac{1}{2}(p - U_n)^\intercal H_{p'} (p - U_n) \geq \frac{1}{2}\|p - U_n\|_2^2 \cdot \sigma \geq \frac{1}{2}\|p - U_n\|_1^2 / n \cdot \sigma \,, \tag{2}$$

where σ is the minimum eigenvalue of the Hessian of μ on the line segment between U_n and p.

The majority of this section is devoted to proving a lower bound for σ. Before doing so, however, we must first address a technical consideration. Because we are considering a function over the space of probability distributions – which is not full-dimensional – the

Hessian and gradient of μ with respect to $\mathbb{R}^n$ depend not only on the definition of our statistic S, but also its parameterization. For the purposes of this subsection, we parameterize S as $S(x) = \sum_{i=1}^{n} \max\left\{\frac{x_i}{m} - \frac{1}{n}, 0\right\} = \frac{1}{m}\sum_{i=1}^{n} \max\left\{x_i - \frac{m}{n}, 0\right\}$.

In the analysis we are about to perform, it will be helpful to replace $\frac{m}{n}$ with a free parameter t which we will eventually set back to roughly m/n. Thus, we define

$$S_t(x) \triangleq \frac{1}{m}\sum_{i=1}^{n} \max\{x_i - t, 0\}$$

and

$$\mu_t(p) \triangleq \mathbb{E}_{x\sim\text{Multinomial}(m,p)}[S_t(x)] = \frac{1}{m}\sum_{i=1}^{n}\sum_{k=\lceil t\rceil}^{m}\binom{m}{k}p_i^k(1-p_i)^{m-k}(k-t)\ . \tag{3}$$

Note that when $t = m/n$ we have $S_t = S$ and $\mu_t = \mu$. Also note that when we compute the Hessian of $\mu_t(p)$, we are treating $\mu_t(p)$ as a function of p and not of t. In the following lemma, we derive an exact expression for the entries of the Hessian. This result is perhaps surprising in light of the likely nonexistence of a closed form expression for $\mu(p)$. That is, while the expectation $\mu(p)$ may have no closed form, we prove that the Hessian of $\mu(p)$ does in fact have a closed form.

▶ **Lemma 5.** *The Hessian of $\mu_t(p)$ viewed as a function of p is a diagonal matrix whose ith diagonal entry is given by*

$$h_{ii} = s_{t,i}\ ,$$

where we define $s_{t,i}$ as follows: Let Δt be the distance of t from the next largest integer, i.e., $\Delta t \triangleq \lceil t\rceil - t$. Then, we have that

$$s_{t,i} = \begin{cases} 0 & \text{for } t = 0 \\ (m-1)\binom{m-2}{t-1}p_i^{t-1}(1-p_i)^{m-t-1} & \text{for } t \in \mathbb{Z}_{>0} \\ \Delta t\cdot s_{\lfloor t\rfloor,i} + (1-\Delta t)\cdot s_{\lceil t\rceil,i} & \text{for } t \geq 0 \text{ and } t \notin \mathbb{Z} \end{cases}.$$

In other words, we will derive the formula for integral $t \geq 1$ and then prove that the value for nonintegral $t \geq 0$ can be found by linearly interpolating between the closest integral values of t.

Proof. Note that because $S_t(x)$ is a separable function of x, $\mu_t(p)$ is a separable function of p, and hence the Hessian of $\mu_t(p)$ is a diagonal matrix. By Equation 3, the i-th diagonal entry of this Hessian can be written explicitly as the following expression:

$$s_{t,i} = \frac{\partial^2}{\partial p_i^2}\mu_t(p) = \frac{d^2}{dp_i^2}\frac{1}{m}\sum_{k=\lceil t\rceil}^{m}\binom{m}{k}p_i^k(1-p_i)^{m-k}(k-t)\ .$$

Notice that if we sum starting from $k = 0$ instead of $k = \lceil t\rceil$, then the sum equals the expectation of $\text{Bin}(m, p_i)$ minus t. That is, notice that:

$$\frac{\mathrm{d}^2}{\mathrm{d}p_i^2}\frac{1}{m}\sum_{k=0}^{m}\binom{m}{k}p_i^k(1-p_i)^{m-k}(k-t) = \frac{\mathrm{d}^2}{\mathrm{d}p_i^2}\frac{1}{m}(p_i m - t) = 0\ .$$

By this observation and the fact that the summand is 0 for integer t when $k = t$, we can switch which values of k we are summing over to k from 0 through $\lfloor t \rfloor$ if we negate the expression:

$$s_{t,i} = \frac{\partial^2}{\partial^2 p_i}\mu_t(p) = \frac{1}{m}\frac{\mathrm{d}^2}{\mathrm{d}p_i^2}\sum_{k=0}^{\lfloor t \rfloor}\binom{m}{k}p_i^k(1-p_i)^{m-k}(t-k) \; .$$

We first prove the case when $t \in \mathbb{Z}_+$. In this case, we view $s_{t,i}$ as a sequence with respect to t (where i is fixed), which we denote s_t. We now derive a generating function for this sequence.[5] Observe that derivatives that are not with respect to the formal variable commute with taking generating functions. Then, the generating function for the sequence $\{s_t\}$ is

$$\frac{\mathrm{d}^2}{\mathrm{d}p_i^2}\frac{1}{m}\left(x\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{(p_i x + 1 - p_i)^m}{1-x}\right) - \frac{x\frac{\mathrm{d}}{\mathrm{d}x}(p_i x + 1 - p_i)^m}{1-x}\right) = (m-1)(p_i x + 1 - p_i)^{m-2}x \; .$$

Note that the coefficient on x^0 is 0, so $s_{0,i} = 0$ as claimed. For $t \in \mathbb{Z}_{>0}$, the right hand side is the generating function of

$$(m-1)\binom{m-2}{t-1}p^{t-1}(1-p)^{m-t-1} \; .$$

Thus, this expression gives the i-th entry Hessian in the $t \in \mathbb{Z}_{\geq 0}$, as claimed.

Now consider the case when t is not an integer. In this case, we have:

$$\begin{aligned}
s_{t,i} &\triangleq \frac{\mathrm{d}^2}{\mathrm{d}p_i^2}\frac{1}{m}\sum_{k=\lceil t \rceil}^{m}\binom{m}{k}p_i^k(1-p_i)^{m-k}(k-t) \\
&= \frac{\mathrm{d}^2}{\mathrm{d}p_i^2}\frac{1}{m}\sum_{k=\lceil t \rceil}^{m}\binom{m}{k}p_i^k(1-p_i)^{m-k}(k-\lceil t \rceil + \Delta t) \\
&= s_{\lceil t \rceil,i} + \Delta t\frac{\mathrm{d}^2}{\mathrm{d}p_i^2}\frac{1}{m}\sum_{k=\lceil t \rceil}^{m}\binom{m}{k}p_i^k(1-p_i)^{m-k}. \\
&= s_{\lceil t \rceil,i} - \Delta t\frac{\mathrm{d}^2}{\mathrm{d}p_i^2}\frac{1}{m}\sum_{k=0}^{\lceil t \rceil - 1}\binom{m}{k}p_i^k(1-p_i)^{m-k} \; .
\end{aligned}$$

The last equality is because if we change bounds on the sum so they are from 0 through m, we get 1 which has partial derivative 0. Thus, we can flip which terms we are summing over if we negate the expression.

Note that this expression we are subtracting above can be alternatively written as:

$$\Delta t\frac{\mathrm{d}^2}{\mathrm{d}p_i^2}\frac{1}{m}\sum_{k=0}^{\lceil t \rceil - 1}\binom{m}{k}p_i^k(1-p_i)^{m-k} = \Delta t \cdot (s_{\lceil t \rceil,i} - s_{\lfloor t \rfloor,i}) \; .$$

Thus, we have

$$s_{t,i} = s_{\lceil t \rceil,i} - \Delta t \cdot (s_{\lceil t \rceil,i} - s_{\lfloor t \rfloor,i}) = \Delta t \cdot s_{\lfloor t \rfloor,i} + (1 - \Delta t) \cdot s_{\lceil t \rceil,i} \; ,$$

as desired. This completes the proof of Lemma 5. ◀

[5] To avoid potential convergence issues, we view generating functions as formal polynomials from the ring of infinite formal polynomials. Under this formalism, there is no need to deal with convergence at all.

It will be convenient to simplify the exact expressions of Lemma 5 into something more manageable. This is done in the following lemma:

▶ **Lemma 6.** *Fix any constant $c > 0$. The Hessian of $\mu(p)$, viewed as a function of p, is a diagonal matrix whose i-th diagonal entry is given by*

$$h_{ii} = s_{t:=m/n,i} \geq \Theta(1) \cdot \begin{cases} \frac{m^2}{n} & \text{for } m \leq n \\ \sqrt{mn} & \text{for } n < m \leq c \cdot \frac{n}{\varepsilon^2} \end{cases},$$

assuming $p_i = \frac{1 \pm \varepsilon}{n}$, $m \geq 6$, $n \geq 2$, and $\varepsilon \leq 1/2$.

Similarly, these bounds are tight up to constant factors, as further improvements would violate our sample complexity lower bounds.

Proof. By Lemma 5, we have an exact expression $s_{t,i}$ for the ith entry of the Hessian of $\mu_t(p)$.

First, consider the case where $m \leq n$. Then we have

$$s_{t,i} = (1 - \Delta t) \cdot s_{\lceil t \rceil, i} \, .$$

Substituting $t = m/n$, $\lceil t \rceil = 1$, and $\Delta t = \lceil t \rceil - t = 1 - m/n$ gives

$$s_{t,i} = \frac{m}{n} \cdot (m-1)(1-p_i)^{m-2} = \Theta(1) \cdot \frac{m^2}{n} \, .$$

Now consider the case where $n < m \leq \Theta(1) \cdot \frac{n}{\varepsilon^2}$. Note that the case where $n < m < 2n$ follows from (i) the fact that $s_{t,i}$ for fractional t linearly interpolates between the value of $s_{t',i}$ the nearest two integral values of t' and (ii) the analyses of the cases where $m \leq n$ and $2n \leq m \leq \Theta(1)\frac{n}{\varepsilon^2}$. Thus, all we have left to do is prove the case where $2n \leq m \leq \Theta(1) \cdot \frac{n}{\varepsilon^2}$.

Since $s_{t,i}$ is a convex combination of $s_{\lceil t \rceil, i}$ and $s_{\lfloor t \rfloor, i}$, it suffices to bound from below these quantities for $t = m/n$. Both of these tasks can be accomplished simultaneously by bounding from below the quantity $s_{t=m/n+\gamma,i}$ for arbitrary $\gamma \in [-1, 1]$.

We do this as follows: Let $t = m/n + \gamma$. Using Stirling's approximation, we can show that for any $\gamma \in [-1, 1]$, we get:

$$s_{t,i} \geq \Theta(1) \cdot \sqrt{mn}$$

The calculations are deferred to the full version. ◀

3 Conclusions and Future Work

In this paper, we gave the first uniformity tester that is sample-optimal, up to constant factors, as a function of the confidence parameter. Our tester is remarkably simple and our novel analysis may be useful in other related settings. By using a known reduction of identity to uniformity, we also obtain the first sample-optimal identity tester in the same setting.

Our result is a step towards understanding the behavior of distribution testing problems in the high-confidence setting. We view this direction as one of fundamental theoretical and important practical interest. A number of interesting open problems remain. Perhaps the most appealing one is to design a *general technique* (see, e.g., [10]) that yields sample-optimal testers in the high confidence regime for a wide range of properties. From the practical standpoint, it would be interesting to perform a detailed experimental evaluation of the various algorithms (see, e.g., [16, 2]).

References

1 J. Acharya, C. Daskalakis, and G. Kamath. Optimal testing for properties of distributions. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3591–3599, 2015.

2 S. Balakrishnan and L. A. Wasserman. Hypothesis testing for densities and high-dimensional multinomials: Sharp local minimax rates. *CoRR*, abs/1706.10003, 2017. URL: `http://arxiv.org/abs/1706.10003`.

3 T. Batu, E. Fischer, L. Fortnow, R. Kumar, R. Rubinfeld, and P. White. Testing random variables for independence and identity. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science*, pages 442–451, 2001.

4 T. Batu, L. Fortnow, R. Rubinfeld, W. D. Smith, and P. White. Testing that distributions are close. In *IEEE Symposium on Foundations of Computer Science*, pages 259–269, 2000. URL: `citeseer.ist.psu.edu/batu00testing.html`.

5 T. Batu, L. Fortnow, R. Rubinfeld, W. D. Smith, and P. White. Testing closeness of discrete distributions. *J. ACM*, 60(1):4, 2013.

6 C. L. Canonne. A survey on distribution testing: Your data is big. but is it blue? *Electronic Colloquium on Computational Complexity (ECCC)*, 22:63, 2015.

7 S. Chan, I. Diakonikolas, P. Valiant, and G. Valiant. Optimal algorithms for testing closeness of discrete distributions. In *SODA*, pages 1193–1203, 2014.

8 L. Devroye and G. Lugosi. *Combinatorial methods in density estimation*. Springer Series in Statistics, Springer, 2001.

9 I. Diakonikolas, T. Gouleakis, J. Peebles, and E. Price. Collision-based testers are optimal for uniformity and closeness. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:178, 2016.

10 I. Diakonikolas and D. M. Kane. A new approach for testing properties of discrete distributions. In *FOCS*, pages 685–694, 2016. Full version available at abs/1601.05557.

11 I. Diakonikolas, D. M. Kane, and V. Nikishkin. Optimal algorithms and lower bounds for testing closeness of structured distributions. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 1183–1202, 2015.

12 I. Diakonikolas, D. M. Kane, and V. Nikishkin. Testing identity of structured distributions. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 1841–1854, 2015.

13 O. Goldreich. The uniform distribution is complete with respect to testing identity to a fixed distribution. *ECCC*, 23, 2016.

14 O. Goldreich. Commentary on two works related to testing uniformity of distributions, 2017. URL: `http://www.wisdom.weizmann.ac.il/~oded/MC/229.html`.

15 O. Goldreich and D. Ron. On testing expansion in bounded-degree graphs. Technical Report TR00-020, Electronic Colloquium on Computational Complexity, 2000.

16 D. Huang and S. Meyn. Generalized error exponents for small sample universal hypothesis testing. *IEEE Trans. Inf. Theor.*, 59(12):8157–8181, dec 2013.

17 E. L. Lehmann and J. P. Romano. *Testing statistical hypotheses*. Springer Texts in Statistics. Springer, 2005.

18 J. Neyman and E. S. Pearson. On the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 231(694-706):289–337, 1933. `doi:10.1098/rsta.1933.0009`.

19 L. Paninski. A coincidence-based test for uniformity given very sparsely-sampled discrete data. *IEEE Transactions on Information Theory*, 54:4750–4755, 2008.

20 R. Rubinfeld. Taming big probability distributions. *XRDS*, 19(1):24–28, 2012.

21 R. Rubinfeld. Taming probability distributions over big domains. *Talk given at STOC'14 Workshop on Efficient Distribution Estimation*, 2014. Available at http://www.iliasdiakonikolas.org/stoc14-workshop/rubinfeld.pdf.

22 G. Valiant and P. Valiant. An automatic inequality prover and instance optimal identity testing. In *FOCS*, 2014.

23 A. W. van der Vaart and J. A. Wellner. *Weak convergence and empirical processes*. Springer Series in Statistics. Springer-Verlag, New York, 1996. With applications to statistics.

24 Y. Wu and P. Yang. Minimax rates of entropy estimation on large alphabets via best polynomial approximation. *IEEE Transactions on Information Theory*, 62(6):3702–3720, June 2016.

Approximating All-Pair Bounded-Leg Shortest Path and APSP-AF in Truly-Subcubic Time

Ran Duan[1]
Institute for Interdisciplinary Information Sciences, Tsinghua University, China
duanran@mail.tsinghua.edu.cn

Hanlin Ren
Institute for Interdisciplinary Information Sciences, Tsinghua University, China
rhl16@mails.tsinghua.edu.cn

Abstract

In the bounded-leg shortest path (BLSP) problem, we are given a weighted graph G with nonnegative edge lengths, and we want to answer queries of the form "what's the shortest path from u to v, where only edges of length $\leq L$ are considered?". A more general problem is the APSP-AF (all-pair shortest path for all flows) problem, in which each edge has two weights – a length d and a capacity f, and a query asks about the shortest path from u to v where only edges of capacity $\geq f$ are considered.

In this article we give an $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-3/2}\log W)$ time algorithm to compute a data structure that answers APSP-AF queries in $O(\log(\epsilon^{-1}\log(nW)))$ time and achieves $(1+\epsilon)$-approximation, where $\omega < 2.373$ is the exponent of time complexity of matrix multiplication, W is the upper bound of integer edge lengths, and n is the number of vertices. This is the first truly-subcubic time algorithm for these problems on dense graphs. Our algorithm utilizes the $O(n^{(\omega+3)/2})$ time max-min product algorithm [Duan and Pettie 2009]. Since the all-pair bottleneck path (APBP) problem, which is equivalent to max-min product, can be seen as all-pair reachability for all flow, our approach indeed shows that these problems are almost equivalent in the approximation sense.

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

Keywords and phrases Graph Theory, Approximation Algorithms, Combinatorial Optimization

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.42

1 Introduction

The shortest path problem is one of the most fundamental problems in algorithmic graph theory. In this paper we study one of its variants, the apBLSP(all-pair *bounded-leg* shortest path), and a more general problem, the APSP-AF(all-pair shortest path for all flows) problem.

In apBLSP, we are given a weighted graph G, and we want to find the shortest path from u to v when only using edges with length no more than L. Answering one such query is easy: just discard edges with length $> L$ and run a shortest-path algorithm. So we consider that there are many queries (u, v, L), and we need to preprocess the graph G to answer these queries efficiently.

The apBLSP problem is a special case of the APSP-AF problem. In the APSP-AF problem, we're given a directed graph G, and each edge has a *length* d and a *capacity* f. Let W and K be the maximum length and maximum capacity, respectively. We assume that all lengths are nonnegative integers. The length of a path is the sum of lengths of all its edges, while the capacity of a path is the minimum capacity over its edges. The goal

[1] R. Duan is supported by a China Youth 1000-Talent grant.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 42; pp. 42:1–42:12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is to compute from G a data structure that handles such queries: given $u, v \in V(G)$ and $1 \leq f \leq K$, determine the shortest path (in terms of length) from u to v with capacity at least f. We call the length of this shortest path "distance from i to j under flow constraint f". Given an instance of the apBLSP problem, we can give the capacity of edges in the order reversely as the order of their lengths, then the apBLSP is easily reducible to APSP-AF.

We consider data structures whose query algorithm uses *no* arithmetic operations. In other words, such a data structure would precompute the answers for all possible queries and store them up. As observed in [7], any such data structure requires $\Omega(n^4)$ space to report the exact answer. It's easy to come up with an $O(n^4)$ exact algorithm: let $\delta(i, j, f)$ be the distance from i to j under flow constraint f, we add edges in descending order of their capacities, and when we add an edge e from i to j with length d and capacity f, we set

$$\delta(i', j', f) \leftarrow \min\{\delta(i', j', f_0), \delta(i', i, f_0) + d + \delta(j, j', f_0)\}, \forall i', j' \in V,$$

where f_0 is the capacity of the last edge we added before e. In this sense, the exact version of APSP-AF is not very interesting. In the approximation setting, what we would like to compute is actually a *df*-matrix[2] A, which is a matrix whose entries are sets of pairs (d, f). The answer to the query (i, j, f) is simply $\min\{d : (d, f') \in A_{ij}, f' \geq f\}$, and can be computed in $O(\log |A_{ij}|)$ time by binary search. The data structure achieves a stretch of $(1 + \epsilon)$ if for any possible query (i, j, f), the returned value is in $[D, (1 + \epsilon)D]$ where D is the actual APSP-AF distance.

Related work. The bounded-leg shortest path problem was firstly considered by Bose *et al.* [4], where they showed that a data structure of size $O(\epsilon^{-1}n^2 \log n)$ can be constructed in $O(n^5)$ time to $(1 + \epsilon)$-approximate the bounded-leg length in Euclidean graphs; if explicit paths are required, their data structure needs $O(\epsilon^{-1}n^3 \log n)$ size. They also gave a data structure of size $O(n^{5/2})$ computable in $O(n^{5/2})$ time to support exact apBLSP queries in planar graphs. Roditty and Segal [11] showed a $(1 + \epsilon)$-approximate data structure for general graphs, which has size $\tilde{O}(n^{2.5})$ and can be computed in $\tilde{O}(n^4)$ time.[3] They also obtained a $(1 + \epsilon)$-approximate data structure for apBLSP in any l_p-metric, which has size $O(n^2 \log n)$ and is computable in $O(n^3(\log^3 n + \log^2 n \cdot \epsilon^{-d}))$ time. In [7], Duan and Pettie improved the time complexity for general graphs to $O(n^3 \log^3 n)$ and the space complexity to $O(\epsilon^{-1}n^2 \log n)$. For planar directed graphs, they also gave a data structure with size $O(kn^{1+1/k})$ that answers bounded-leg reachability queries in $\tilde{O}(n^{\frac{k-1}{2k}})$ time, for any integer $k \geq 2$.

The APSP-AF problem was introduced by Shinn and Takaoka [12, 13, 14, 15]. In graphs with unit edge lengths, they showed an $O(\sqrt{K}n^{(\omega+9)/4})$ exact algorithm. In general graphs, they showed exact algorithms of running time $O(Kn^3)$, $\tilde{O}(\sqrt{K}W^{(\omega+5)/4}n^{(\omega+9)/4})$ and $O(m^2n + \min(nW, mn^2 \log \frac{W}{mn}))$ respectively.

We can also view APSP-AF as an *offline* version of partially dynamic shortest path problem: edge insertions and distance queries appear by descending order of f, and we know the whole sequence of insertions and queries at the beginning of algorithm. Most works on dynamic shortest path problem are *online*: the sequence of operations and queries must be processed in order. In *incremental* setting, Ausiello *et al.* [1] showed an algorithm to handle all pair distance queries and edge insertions in a digraph in $O(Wn^3 \log(nW))$ time. In *decremental* setting, Bernstein [3] showed a $(1+\epsilon)$-approximate algorithm with a total running

[2] See section 2.2.
[3] $\tilde{O}$ hides poly$(\log n)$ factor.

time of $\tilde{O}(mn\epsilon^{-1}\log R)$ in weighted digraphs, where R is the ratio of the largest weight that was ever seen in the graph to the smallest such weight. For unweighted digraphs, Baswana, Hariharan and Sen [2] showed a randomized algorithm with $O(\min(n^{3/2}m\sqrt{\log n}, n^3\log^2 n))$ total update time which returns exact answers w.h.p, and a $(1+\epsilon)$-approximate algorithm with $O(nm\epsilon^{-2}\log n + n^2\epsilon^{-1}\sqrt{m\log n})$ total update time. More results can be found in surveys [6, 10]. However, the online setting has its disadvantages: in both incremental and decremental settings, even a reachability oracle in a truly-subcubic total time would refute the OMV conjecture [9].

Our algorithm relies on fast algorithms for max-min product. The first truly-subcubic time algorithm for max-min product was discovered by Vassilevska, Williams and Yuster [16], which has running time $O(n^{2+\omega/3})$. Subsequently, Duan and Pettie [8] improved this algorithm to $O(n^{(\omega+3)/2})$ time.

Our contribution. We show that a data structure for APSP-AF within stretch $(1+\epsilon)$ can be computed in $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-3/2}\log W)$ time for any n-vertex graph and any $\epsilon > 0$, where $\omega < 2.373$ is the exponent of time complexity of matrix multiplication [5]. This is the first truly-subcubic time (*i.e.* $O(n^{3-\delta}\mathrm{poly}(\epsilon^{-1}, \log W))$ for some $\delta > 0$) algorithm to approximate apBLSP or APSP-AF problem in general graphs when $m = \Theta(n^2)$ and $W = \Omega(n)$. Our data structure uses $O(n^2\epsilon^{-1}\log n \log(nW))$ space.

We also establish the equivalence between approximating APSP-AF and computing max-min product. In particular, it's shown in Section 3 that if the max-min product of two matrices can be computed in $T(n)$ time, then a $(1+\epsilon)$-approximate data structure for APSP-AF can be computed in $\tilde{O}(T(n)\epsilon^{-2}\log W)$ time. This is optimal up to $\mathrm{poly}(\epsilon^{-1}, \log n, \log W)$ factors in the sense that APSP-AF approximation is at least as hard as max-min product. In fact, max-min product is reducible to the special case of *bounded-leg reachability problem* where the goal is only to query if u can reach v via edges of capacity $\geq f$. [4] Consider computing the max-min product of two $n \times n$ matrices A, B. We construct a directed graph $G = (V, E)$ with $3n$ vertices $V = \{u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n, w_1, w_2, \ldots, w_n\}$. For every i, j, link an edge from u_i to v_j with capacity A_{ij} and link an edge from v_i to w_j with capacity B_{ij}. Let C be the max-min product of A and B, then $C_{ij} \geq f$ iff w_j is reachable from u_i by edges of capacity $\geq f$.

2 Preliminaries

We denote $[n] = \{1, \ldots, n\}$, $[n]_0 = [n] \cup \{0\}$, $\min(\varnothing) = +\infty, \max(\varnothing) = -\infty$. Rows and columns of matrices are numbered from 1 to n. Let $\omega < 2.373$ be the exponent of time complexity of matrix multiplication. For a set of $m \times n$ matrices $\{A_i\}$, $\min\{A_i\}$ is the entry-wise minimum of them. Let W, K be the upper bound of lengths and capacities of edges, respectively; *i.e.* for any edge, its length is in $[W]_0$ and its capacity is in $[K]$.

2.1 Matrix products

We introduce the *distance product* and *max-min product* for two matrices, denoted as $\star$ and $\circledvee$ respectively.

[4] It should be "edges of length $\leq L$" for bounded-leg reachability problem; However this constraint is equivalent.

▶ **Definition 2.1.** For two matrices A, B of size $n \times n$, we define:

$$(A \star B)_{ij} = \min_{k\in[n]}\{A_{ik} + B_{kj}\};$$
$$(A \circledcirc B)_{ij} = \max_{k\in[n]}\{\min\{A_{ik}, B_{kj}\}\}.$$

A simpler version of our algorithm in Section 3, which runs in $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-2}\log W)$, makes use of max-min product algorithm [8] for two matrices.

▶ **Lemma 2.2** ([8]). *There is an algorithm that, given two $n \times n$ matrices A and B, compute $A \circledcirc B$ in $O(n^{(\omega+3)/2})$ time.*

We also need the following lemma, stating that the distance product of matrices with large entries can be approximated by several distance products of matrices with small entries:

▶ **Lemma 2.3** (Lemma 5.1 of [17]). *Suppose A, B are two matrices whose entries are in $[M]_0 \cup \{+\infty\}$, $C = A \star B$. Let R be a power of 2, and* Scale(A, M, R) *be a matrix A' such that*

$$A'_{ij} = \begin{cases} \lceil RA_{ij}/M \rceil & \text{if } 0 \le A_{ij} \le M \\ +\infty & \text{otherwise} \end{cases}.$$

Define C' as:

$$C' = \min_{\lfloor \log_2 R\rfloor \le r \le \lceil \log_2 M\rceil} \{(2^r/R)\cdot(\textsc{Scale}(A, 2^r, R) \star \textsc{Scale}(B, 2^r, R))\}$$

Then for any i, j, $C_{ij} \le C'_{ij} \le (1 + \frac{4}{R})C_{ij}$.

2.2 The distance/flow pairs and their operations

We introduce the notion of *df-pairs* in [12]. Our algorithm is based on manipulations of *df*-pairs and *df*-matrices.

▶ **Definition 2.4.** A *df-pair* is a pair (d, f) where d represents distance and f represents capacity or flow. We may assume $f \in [K]$.

A *df-set* is a set of *df*-pairs.

A *df-matrix* is a matrix whose entries are *df*-sets. If for every *df*-pair $(d, f) \in A_{ij}$, $d \in [R]_0$, then we say A is *within distance R*.

▶ **Definition 2.5** ([12]). The *merit* order $<_m$ is defined as $(d_1, f_1) >_m (d_2, f_2)$ iff $d_1 < d_2 \wedge f_1 \ge f_2$. For a *df*-set S, define $\mathcal{C}(S) = \{(d, f) : \nexists (d', f') \in S, (d', f') >_m (d, f)\}$.

▶ Remark. Given a *df*-set S sorted by distances, $\mathcal{C}(S)$ can be computed in $O(|S|)$ time.

We can see if $(d_1, f_1) >_m (d_2, f_2)$, then (d_2, f_2) is not useful since its distance is larger but flow is no larger. The operator $\mathcal{C}$ can be used to delete redundant elements of a *df*-set.

▶ **Definition 2.6** ([12]). For two *df*-sets, define their addition and multiplication as

$$S_1 + S_2 = \mathcal{C}(S_1 \cup S_2)$$
$$S_1 \cdot S_2 = \mathcal{C}(\{(d_1 + d_2, \min(f_1, f_2)) : (d_1, f_1) \in S_1, (d_2, f_2) \in S_2\}).$$

The multiplication of two *df*-sets S_1, S_2 can be understood as we try to link the paths pair-wisely in them, so the distance is the sum of their distances, and the flow is the minimum of their flows.

▶ **Definition 2.7.** For a *df*-set S, define $S(f) = \min\{d : (d, f') \in S, f' \geq f\}$. Similarly, for a *df*-matrix A, define $A(f)$ be the matrix whose entries are in $\mathbb{R} \cup \{+\infty\}$ such that $(A(f))_{ij} = A_{ij}(f)$.

For two *df*-sets S, S', we say $S = S'$ if $\forall f \in [K]$, $S(f) = S'(f)$. The relation $=$ is an equivalence relation. We say S' is a $(1+\epsilon)$-approximation of S, denoted as $S' \approx_\epsilon S$, if $S(f) \leq S'(f) \leq (1+\epsilon)S(f)$ for any $f \in [K]$. Similarly, for two *df*-matrices A, A' of size $n \times n$, say $A = A'$ if $A_{ij} = A'_{ij}$ for all $i, j \in [n]$, and say A' is a $(1+\epsilon)$-approximation of A, denoted as $A' \approx_\epsilon A$, if A'_{ij} is a $(1+\epsilon)$-approximation of A_{ij} for all $i, j \in [n]$.

▶ **Definition 2.8.** The *product* of two *df*-matrices A, B, denoted as $A \star B$, is defined as $(A \star B)_{ij} = \sum_{k\in[n]} A_{ik} \cdot B_{kj}$. We define $A^1 = A$ and $A^p = A^{p-1} \star A$ for integer $p > 1$.

Intuitively, this product can be understood in the following way: suppose we have a 3-layer graph $G = (V, E)$ where $V = \{u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n, w_1, w_2, \ldots, w_n\}$. Let A be the *df*-matrix representing edges among $\{u_i\}$ and $\{v_k\}$: for all $(d, f) \in A_{ik}$, there is an edge with length d and capacity f from u_i and v_k. Similarly let B be the *df*-matrix representing edges among $\{v_k\}$ to $\{w_j\}$. Let $C = A \star B$, then each element $(d, f) \in C_{ij}$ corresponds to a path from u_i to w_j with total length d and capacity f.

We have the following facts:

▶ **Fact 2.9.** *Let S_1, S_2 be two df-sets and $f \in [K]$. Then $(\mathcal{C}(S))(f) = S(f)$, $(S_1 + S_2)(f) = \min\{S_1(f), S_2(f)\}$ and $(S_1 \cdot S_2)(f) = S_1(f) + S_2(f)$.*

Proof. These are immediate from definition. ◀

▶ **Fact 2.10.** *For three df-matrices A, B, C and $\epsilon \geq 0$, $C \approx_\epsilon A \star B$ if and only if $C(f) \approx_\epsilon A(f) \star B(f)$ for any $f \in [K]$. In particular when $\epsilon = 0$, $C = A \star B$ iff $C(f) = A(f) \star B(f)$ for all $f \in [K]$.*

Proof. For all i, j, f, from Fact 2.9,

$$(A \star B)_{ij}(f) = (\sum_{k\in[n]} A_{ik} \cdot B_{kj})(f) = \min_{k\in[n]}\{A_{ik}(f) + B_{kj}(f)\} = (A(f) \star B(f))_{ij},$$

and

$$\begin{aligned}
&C \approx_\epsilon (A \star B)\\
\iff &\forall i, j, f,\ C_{ij}(f)/(A \star B)_{ij}(f) \in [1, 1+\epsilon]\\
\iff &\forall i, j, f,\ C_{ij}(f)/(A(f) \star B(f))_{ij} \in [1, 1+\epsilon]\\
\iff &\forall f,\ C(f) \approx_\epsilon A(f) \star B(f).
\end{aligned}$$

◀

▶ **Fact 2.11.** *Let A, B, C be df-matrices. If B is a $(1+\epsilon_1)$-approximation of A and C is a $(1+\epsilon_2)$-approximation of B, then C is a $(1+\epsilon_1)(1+\epsilon_2)$-approximation of A.*

Proof. For any i, j, f, $A_{ij}(f) \leq B_{ij}(f) \leq C_{ij}(f) \leq (1+\epsilon_2)B_{ij}(f) \leq (1+\epsilon_1)(1+\epsilon_2)A_{ij}(f)$. ◀

▶ **Fact 2.12.** *Let A, B be df-matrices and $\epsilon > 0$. If $B \approx_\epsilon A$, then $B \star B \approx_\epsilon A \star A$.*

Proof. For any i, j, f, from Fact 2.10, $(A \star A)(f) = A(f) \star A(f)$, and:

$$\begin{aligned}
(A \star A)(f)_{ij} =& \min_{k\in[n]}\{A(f)_{ik} + A(f)_{kj}\}\\
\leq& \min_{k\in[n]}\{B(f)_{ik} + B(f)_{kj}\} = (B \star B)(f)_{ij}\\
\leq& (1+\epsilon) \min_{k\in[n]}\{A(f)_{ik} + A(f)_{kj}\}\\
=& (1+\epsilon)(A \star A)(f)_{ij}.
\end{aligned}$$

◀

3 Main Algorithm

In this section we introduce an $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-2}\log W)$ algorithm for approximating APSP-AF problem within a stretch of $(1+\epsilon)$. The idea is not hard. Suppose M is the *df*-matrix corresponding to G, and $D = M^n$, then D is the exact *df*-matrix representing all-pair distances for all flows. To approximate D, we only need an algorithm that approximates the product of *df*-matrices by a stretch of $(1+\Theta(\epsilon/\log n))$. By Lemma 2.3, this can be done by exact *df*-matrix product algorithms that only handles distances no more than $R = O(\epsilon^{-1}\log n)$, which turns out to be expressible as $O(R^2)$ max-min products.

3.1 Exact product for small distances

The following algorithm shows that the product of two *df*-matrices A and B can be reduced to $O(R^2)$ max-min products if the distances are integers between 0 and R.

Algorithm 1 Exact product for two *df*-matrices.

function EXACT-PROD(A, B, R)
  $C_{ij} \leftarrow \varnothing$
  for d_0 in $[R]_0$ **do**
    $A_{ij}^{(d_0)} \leftarrow \max\{f : (d,f) \in A_{ij}, d \le d_0\}$
    $B_{ij}^{(d_0)} \leftarrow \max\{f : (d,f) \in B_{ij}, d \le d_0\}$
  for d_1, d_2 in $[R]_0^2$ **do**
    $C' \leftarrow A^{(d_1)} \mathbin{\textcircled{\vee}} B^{(d_2)}$
    $C_{ij} \leftarrow C_{ij} \cup \{(d_1+d_2, C'_{ij})\}$
  $C_{ij} \leftarrow \{\mathcal{C}(C_{ij})\}$; **return** C

▶ **Lemma 3.1.** EXACT-PROD *correctly returns the product $A \star B$ of two df-matrices A, B which are both within distance R.*

Proof. Let $C = \text{EXACT-PROD}(A, B, R)$. For any $i, j \in [n]$ and $f \in [K]$,

$$\begin{aligned}
&(A(f) \star B(f))_{ij} \\
=& \min_{k\in[n]}\{A(f)_{ik} + B(f)_{kj}\} \\
=& \min_{k\in[n]}\{\min\{d_1 : (d_1,f_1) \in A_{ik}, f_1 \ge f\} + \min\{d_2 : (d_2,f_2) \in B_{kj}, f_2 \ge f\}\} \\
=& \min_{d_1,d_2}\{d_1+d_2 : \exists k \in [n], A_{ik}^{(d_1)} \ge f, B_{kj}^{(d_2)} \ge f\} \\
=& \min_{d_1,d_2}\{d_1+d_2 : \left(A^{(d_1)} \mathbin{\textcircled{\vee}} B^{(d_2)}\right)_{ij} \ge f\} \\
=& \min\{d' : (d',f') \in C_{ij}, f' \ge f\} = (C(f))_{ij},
\end{aligned}$$

thus $C = A \star B$ by Fact 2.10. ◀

3.2 Approximate product for arbitrary distances

For two *df*-matrices A, B within distance M, we can compute an approximation of $A \star B$ by applying Lemma 2.3. Given a parameter R which is a power of 2, the following algorithm computes a $(1+\frac{4}{R})$-approximation of $A \star B$.

Algorithm 2 Approximate product of two *df*-matrices.

function SCALE(a, M, R)
  $a' = \begin{cases} \lceil R \cdot a/M \rceil & 0 \le a \le M \\ +\infty & \text{otherwise} \end{cases}$
  return a'
function APPROX-PROD(A, B, M, R)
  $C_{ij} \leftarrow \varnothing$
  for $r \leftarrow \lfloor \log_2 R \rfloor$ to $\lceil \log_2 M \rceil$ **do**
    $A^{(r)}_{ij} \leftarrow \mathcal{C}(\{(\text{SCALE}(d, 2^r, R), f) : (d, f) \in A_{ij}\})$
    $B^{(r)}_{ij} \leftarrow \mathcal{C}(\{(\text{SCALE}(d, 2^r, R), f) : (d, f) \in B_{ij}\})$
    $C^{(r)} = \text{EXACT-PROD}(A^{(r)}, B^{(r)}, R)$
    $C_{ij} \leftarrow C_{ij} \cup \{((2^r/R) \cdot d, f) : (d, f) \in C^{(r)}_{ij}\}$
  $C_{ij} \leftarrow \{\mathcal{C}(C_{ij})\}$; **return** C

▶ **Lemma 3.2.** *Let A, B be two df-matrices within distance M, and R a power of 2, $C =$ APPROX-PROD(A, B, M, R), then $C \approx_{\frac{4}{R}} A \star B$. Moreover, for each i, j, $|C_{ij}| = O(R \log M)$.*

Proof. Fix $f \in [K]$. Then $A^{(r)}(f) = \text{SCALE}(A(f), 2^r, R)$, $B^{(r)}(f) = \text{SCALE}(B(f), 2^r, R)$. By Lemma 3.1,

$$C(f) = \min_{\lfloor \log_2 R \rfloor \le r \le \lceil \log_2 M \rceil} \{(2^r/R) \cdot (A^{(r)}(f) \star B^{(r)}(f))\}.$$

From Lemma 2.3, $C(f) \approx_{\frac{4}{R}} A(f) \star B(f)$ for all f. The lemma is immediate from Fact 2.10. The "moreover" part holds since $|C_{ij}| \le \sum_{r=\lfloor \log_2 R \rfloor}^{\lceil \log_2 M \rceil} |C^{(r)}_{ij}| \le 2R\lceil \log_2 M \rceil$ for any i, j. ◀

3.3 Main procedure

Consider an instance G of APSP-AF problem, we represent G as a *df*-matrix A: A_{ij} is the set of all (d, f) such that there is an edge from i to j with length d and capacity f. Given $\epsilon > 0$, the following algorithm computes a *df*-matrix that approximates answers of APSP-AF problem within a stretch of $1 + \epsilon$.

Algorithm 3 the main procedure

function APSP-AF-APPROX(A, W, ϵ)
  $M \leftarrow nW$
  $R \leftarrow 4\lceil \log_2 n \rceil / \ln(1 + \epsilon)$
  $R \leftarrow 2^{\lceil \log_2 R \rceil}$
  $\tilde{D}^{(0)}_{ij} \leftarrow A_{ij}$
  $\tilde{D}^{(0)}_{ii} \leftarrow \{(0, K)\}$
  for $t = 1$ to $\lceil \log_2 n \rceil$ **do**
    $\tilde{D}^{(t)} \leftarrow \text{APPROX-PROD}(\tilde{D}^{(t-1)}, \tilde{D}^{(t-1)}, M, R)$
  return $\tilde{D}^{(\lceil \log_2 n \rceil)}$

▶ **Theorem 3.3.** *For a graph G, let A be the df-matrix representing G as above, D be the df-matrix such that $D(f)_{ij}$ is the shortest distance from i to j under flow constraint f in G, and $\tilde{D} =$ APSP-AF-APPROX(A, W, ϵ). Then $\tilde{D} \approx_\epsilon D$. Moreover,* APSP-AF-APPROX *runs in $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-2} \log W)$ time, and $\tilde{D}$ occupies $O(n^2\epsilon^{-1} \log n \log(nW))$ space.*

Proof. The time bottleneck of APSP-AF-Approx is $O(\log(nW))$ invocations of Exact-Prod, which runs in $\tilde{O}(n^{(\omega+3)/2}R^2)$ time. From Lemma 3.2, for each i, j, we have $|\tilde{D}_{ij}| = O(R\log(nW))$. Since $R = O(\epsilon^{-1}\log n)$, the "moreover" part is proved.

Let $D^{(t)}_{ij}(f)$ be the shortest distance from i to j under flow constraint f, where only paths of $\le 2^t$ edges are considered. Then $D^{(t)} = D^{(t-1)} \star D^{(t-1)}$, $\tilde{D}^{(t)} \approx_{\frac{4}{R}} \tilde{D}^{(t-1)} \star \tilde{D}^{(t-1)}$ (by Lemma 3.2), and $D^{(0)} = \tilde{D}^{(0)}$. We can prove by induction that $\tilde{D}^{(t)}$ is a $(1+\frac{4}{R})^t$-approximation of $D^{(t)}$. Base case ($t = 0$) is obvious; suppose this is true for t_0, let $\hat{D}^{(t_0+1)} = \tilde{D}^{(t_0)} \star \tilde{D}^{(t_0)}$, then $\hat{D}^{(t_0+1)}$ is a $(1+\frac{4}{R})^{t_0}$-approximation of $D^{(t_0+1)}$ by induction hypothesis and Fact 2.12. Since $\tilde{D}^{(t_0+1)} \approx_{\frac{4}{R}} \hat{D}^{(t_0+1)}$, $\tilde{D}^{(t_0+1)}$ is a $(1+\frac{4}{R})^{(t_0+1)}$-approximation of $D^{(t_0+1)}$ by Fact 2.11.

Since

$$\left(1+\frac{4}{R}\right)^{\lceil\log_2 n\rceil} \le \left(1+\frac{\ln(1+\epsilon)}{\lceil\log_2 n\rceil}\right)^{\lceil\log_2 n\rceil} \le 1+\epsilon,$$

$\tilde{D} = \tilde{D}^{(\lceil\log_2 n\rceil)}$ is a $(1+\epsilon)$-approximation of D. ◀

3.4 Computing witnesses

Our algorithm can be adapted to answer path queries in addition to distance queries: given $i, j \in [n]$ and $f \in [K]$, we not only find the approximated distance from i to j under flow constraint f, but also find an actual path with that distance. Suppose the path contains ℓ vertices, then a query takes $O(\ell \log\log_{1+\epsilon}(nW))$ time to report the whole path.

First we notice that the max-min product algorithm in [8] can be modified to return *witnesses* of max-min product:

▶ **Lemma 3.4** ([8]). *For two $n \times n$ matrices A, B, a pair of matrices (C, W) can be computed in $O(n^{(\omega+3)/2})$ time, where $C = A \otimes B$ and for any $i, j \in [n]$, suppose $k = W_{ij}$, then $C_{ij} = \min\{A_{ik}, B_{kj}\}$. We call W a* witness *of $A \otimes B$.*

We attach a node in the graph as additional information to every *df*-pair we encounter. For a *df*-pair representing a path p from i to j, suppose node x is attached to it, then x is on p. Moreover, if $x = i$ or $x = j$, then p contains at most one edge. More precisely:

- In Algorithm 1, we compute a witness matrix W besides computing $C' = A^{(d_1)} \otimes B^{(d_2)}$. Each *df*-pair $(d_1 + d_2, C'_{ij})$ in line 8 is attached with W_{ij};
- In Algorithm 2, line 10, the node attached with $((2^r/R)\cdot d, f)$ is the same as the node attached with (d, f);
- In Algorithm 3, line 6, the node attached with $(0, K) \in \tilde{D}^{(0)}_{ii}$ is i.

It is easy to see from above modifications that, suppose node x is attached with $(d, f) \in \tilde{D}_{ij}$, then there is a path from i to j with distance $\le d$, flow $\ge f$ which passes through node x.

We give a simple recursive algorithm that, based on nodes attached to *df*-pairs in the data structure, find the whole path for a query (u, v, f).

Algorithm 4 Path querying

```
1: function QUERY(u, v, f)
2:     (d', f') ← arg min{d' : (d', f') ∈ D̃_uv, f' ≥ f}
3:     x ← the node attached with (d', f')
4:     if x = u or x = v then
5:         return edge u → v
6:     else
7:         return QUERY(u, x, f)+QUERY(x, v, f)
```

4 Faster Implementation of Exact Product

In this section we show how to multiply two *df*-matrices within distance R in $\tilde{O}(n^{(\omega+3)/2}R^{3/2})$ time. This implies an $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-3/2}\log W)$ time algorithm for the original APSP-AF approximation problem.

The idea is to look into details of the max-min product algorithm in [8]. The task of computing max-min product of two matrices can be composed into $O(t)$ Boolean matrix multiplications and $O(n^3/t)$ extra work, so in our case we have $O(tR^2)$ matrix multiplications and $O(R^2n^3/t)$ extra work. However, these $O(tR^2)$ matrix multiplications can be expressed as $O(t)$ distance products of matrices whose elements are in $[R]$, thus can be accelerated by Zwick's algorithm for distance product [17] to run in $\tilde{O}(tRn^\omega)$ time.

▶ **Lemma 4.1** ([17]). *The distance product of two $n \times n$ matrices whose entries are in $[M]$ can be computed in $\tilde{O}(Mn^\omega)$ time.*

4.1 Row-balancing and column-balancing

The max-min product algorithm in [8] uses the concept of row-balancing and column-balancing. The function of row (column)-balancing is to rearrange the entries of a sparse matrix such that every row (column) has a moderate number of entries. We adapt this technique to *df*-matrices here.

▶ **Definition 4.2.** Let A be an $n \times n$ *df*-matrix where $m = \sum_{i,j\in[n]} |A_{ij}|$. We define the *row-balancing* of A, denoted by $\mathbf{rb}(A)$, as a pair of $n \times n$ *df*-matrices (A', A''), where each row of A' or A'' contains at most $k = \lceil m/n \rceil$ *df*-pairs in total. More precisely, we first sort all *df*-pairs in the i-th row of A by ascending order of f values, and partition them into blocks of size $\le k$: $T_i^1, T_i^2, \ldots, T_i^{a_i}$. The last block $T_i^{a_i}$ has *at most* k *df*-pairs, and other blocks have *exactly* k *df*-pairs. We define A' to contain all entries in the last block: $A'_{ij} = A_{ij} \cap T_i^{a_i}$. Since $\sum_{i\in[n]}(a_i - 1) \le m/k \le n$, we can rearrange all $T_i^j (1 \le j < a_i)$ into n distinct rows, by selecting an injection $\rho : \{(i, j) : i \in [n], j \in [a_i - 1]\} \to [n]$. Then we define $A''_{ij} = A_{i'j} \cap T_{i'}^q$ where $(i', q) = \rho^{-1}(i)$.

The *column-balancing* of A, denoted by $\mathbf{cb}(A)$, is a pair of $n \times n$ *df*-matrices (A', A'') such that $(A'^\top, A''^\top) = \mathbf{rb}(A^\top)$.

4.2 Dominance product

▶ **Definition 4.3.** The *dominance product* of two *df*-matrices A, B, denoted as $A \circledcirc B$, is defined as $(A \circledcirc B)_{ij} = \min\{d_1 + d_2 : (d_1, f_1) \in A_{ik}, (d_2, f_2) \in B_{kj}, f_1 \le f_2\}$.

There is also a "3-layer graph interpretation" of the dominance product of two *df*-matrices: consider a graph $G = (V, E)$ where $V = \{u_1, u_2, \ldots, u_n, v_1, v_2, \ldots, v_n, w_1, w_2, \ldots, w_n\}$. Let

A be the *df*-matrix representing edges among $\{u_i\}$ and $\{v_k\}$: for all $(d, f) \in A_{ik}$, there is an edge with length d and capacity f from u_i and v_k. Similarly let B be the *df*-matrix representing edges among $\{v_k\}$ to $\{w_j\}$. Let $C = A \circledcirc B$, then C_{ij} is the minimum distance of a path from u_i to w_j, where it's required that the capacity of the first edge is no more than the capacity of the second edge.

▶ **Lemma 4.4.** *Given two df-matrices A, B within distance R, let $m_1 = \sum_{i,j\in[n]} |A_{ij}|$, $m_2 = \sum_{i,j\in[n]} |B_{ij}|$. The dominance product $A \circledcirc B$ can be computed in $\tilde{O}(m_1 m_2/n + Rn^\omega)$ time.*

Proof. Let $(A', A'') = \mathbf{cb}(A)$, the column-balancing of A. Define two (scalar) matrices $\tilde{A}$ and $\tilde{B}$:

$$\tilde{A}_{ik} = \min\{d : (d, f) \in A''_{ik}\}, \tilde{B}_{kj} = \min\{d : (d, f) \in B_{k'j}, f \geq \text{max flow of } T^{q'}_{k'}\},$$

where $(k', q') = \rho^{-1}(k)$. Note that $(\tilde{A} \star \tilde{B})_{ij}$ is the smallest $d_1 + d_2$ such that:

- $(d_1, f_1) \in A_{ik'}$, $(d_2, f_2) \in B_{k'j}$;
- $(d_1, f_1) \in T^{q'}_{k'}$, $q' < a_{k'}$, and f_2 dominates all flows in $T^{q'}_{k'}$.

What we haven't considered are pairs $(d_1, f_1) \in A_{ik'}$ and $(d_2, f_2) \in B_{k'j}$ such that (d_1, f_1) is contained in the largest part of column k', or f_2 is smaller than the largest flow in $T^{q'}_{k'}$, that is, f_1 and f_2 locate in the same "block". In either cases, for any (d_2, f_2) there are only $O(m_1/n)$ entries in A' or A'' to compare. The time complexity for distance product is $\tilde{O}(Rn^\omega)$, and the time complexity for the rest pairs are $O(m_1 m_2/n)$. ◀

4.3 Faster exact product

▶ **Theorem 4.5.** *Given two df-matrices A, B within distance R, their product $A \star B$ can be computed in $\tilde{O}(n^{(\omega+3)/2} R^{3/2})$ time.*

Proof. Let $D^{(d)}_{ij} = \max\{f_1 : d_1 + d_2 = d, f_1 \leq f_2, (d_1, f_1) \in A_{ik}, (d_2, f_2) \in B_{kj}\}$ and $D'^{(d)}_{ij} = \max\{f_2 : d_1 + d_2 = d, f_1 \geq f_2, (d_1, f_1) \in A_{ik}, (d_2, f_2) \in B_{kj}\}$, then $(A \star B)_{ij} = \mathcal{C}\left(\bigcup_d \left\{(d, \max\{D^{(d)}_{ij}, D'^{(d)}_{ij}\})\right\}\right)$. We show $D^{(d)}$ for all $0 \leq d \leq 2R$ can be computed in $\tilde{O}(n^{(\omega+3)/2} R^{3/2})$ total time; $D'^{(d)}$ is similar.

Sort all *df*-pairs of A and B by their f values in increasing order, and then partition this sorted list into t parts $L_1, L_2, \ldots, L_t$, where each part has $O(n^2 R/t)$ elements. Let $A^{(r)}_{ij} = A_{ij} \cap L_r$, $B^{(r)}_{ij} = B_{ij} \cap L_r$ for $1 \leq r \leq t$. For each r we let $(A'^{(r)}, A''^{(r)}) = \mathbf{rb}(A^{(r)})$, and compute $A^{(r)} \circledcirc B$, $A'^{(r)} \circledcirc B$, $A''^{(r)} \circledcirc B$. For each d, i, j, we can compute $D^{(d)}_{ij}$ as follows:

1. We first determine which part $D^{(d)}_{ij}$ lies in, *i.e.* find the largest r such that $(A^{(r)} \circledcirc B)_{ij} \leq d$;
2. If $(A'^{(r)} \circledcirc B)_{ij} \leq d$, then we only need to consider k where $A'^{(r)}_{ik} \neq \varnothing$ to determine $D^{(d)}_{ij}$;
3. Otherwise, we find the largest q such that $(A''^{(r)} \circledcirc B)_{\rho(i,q),j} \leq d$, and determine $D^{(d)}_{ij}$ by looking through the q-th part of i-th row of $A^{(r)}$.

The above procedure takes $O(nR/t)$ time for each d, i, j. We compute $A^{(r)} \circledcirc B$ in $\tilde{O}(n^3 R^2/t^2 + Rn^\omega)$ time as follows. Let $f' = \min\{f : (d, f) \in L_{r+1}\}$, then $A^{(r)} \circledcirc B = \min\{A^{(r)} \circledcirc B^{(r)}, A^{(r)}(0) \star B(f')\}$. The $A^{(r)} \circledcirc B^{(r)}$ part considers comparisons inside L_r, which we compute in $\tilde{O}((n^2 R/t)^2/n + Rn^\omega)$ time by Lemma 4.4; the $A^{(r)}(0) \star B(f')$ part represents comparisons between L_r and $\bigcup_{r<r'\leq t} L_{r'}$, and is simply a distance product computable in $\tilde{O}(Rn^\omega)$ time. We can compute $A'^{(r)} \circledcirc B$ and $A''^{(r)} \circledcirc B$ similarly as we did in $A^{(r)} \circledcirc B$. Therefore, we spend $\tilde{O}(n^3 R^2/t + tRn^\omega)$ time in total for all $1 \leq r \leq t$. The theorem follows by setting $t = n^{(3-\omega)/2} R^{1/2}$. ◀

4.4 Computing witnesses

We modify this algorithm to support path queries as in Section 3.4. We call $k \in [n]$ is a witness of $D^{(d)}_{ij}$ if there is $(d_1, f_1) \in A_{ik}$, $(d_2, f_2) \in B_{kj}$ such that $d_1 + d_2 = d$, $f_1 \le f_2$ and $D^{(d)}_{ij} = f_1$. For any d, i, j, a witness of $D^{(d)}_{ij}$ can be found by step 2,3 when computing $D^{(d)}_{ij}$. Similarly we can find witnesses of $D'^{(d)}_{ij}$. For each d, i, j, if $D^{(d)}_{ij} \ge D'^{(d)}_{ij}$, attach the witness of $D^{(d)}_{ij}$ to the df-pair $(d, D^{(d)}_{ij})$; otherwise attach the witness of $D'^{(d)}_{ij}$ to the df-pair $(d, D'^{(d)}_{ij})$. The time complexity remains the same.

5 Conclusions

Our work shows that the apBLSP and APSP-AF problem can be approximated within a stretch of $(1+\epsilon)$ in $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-3/2}\log W)$ time. Also, a faster algorithm for max-min product would imply a faster algorithm for approximating APSP-AF problem. In this sense, since approximating APSP-AF is at least as hard as max-min product, our algorithm is optimal up to $\tilde{O}(\mathrm{poly}(\epsilon^{-1})\log W)$ factors.

We think the main open problem left by our work is to further improve the dependence on ϵ. Can the APSP-AF problem be approximated in, say, $\tilde{O}(n^{(\omega+3)/2}\epsilon^{-1}\log W)$ time? There is an $\epsilon^{-1/2}$ gap here, and we might need a more refined approach to fill this gap.

References

1. Giorgio Ausiello, Giuseppe F Italiano, Alberto Marchetti Spaccamela, and Umberto Nanni. Incremental algorithms for minimal length paths. *Journal of Algorithms*, 12(4):615–638, 1991.
2. Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. *Journal of Algorithms*, 62(2):74–92, 2007.
3. Aaron Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. *SIAM Journal on Computing*, 45(2):548–574, 2016.
4. Prosenjit Bose, Anil Maheshwari, Giri Narasimhan, Michiel Smid, and Norbert Zeh. Approximating geometric bottleneck shortest paths. *Computational Geometry*, 29(3):233–249, 2004.
5. Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.
6. Camil Demetrescu and Giuseppe F Italiano. Dynamic shortest paths and transitive closure: Algorithmic techniques and data structures. *Journal of Discrete Algorithms*, 4(3):353–383, 2006.
7. Ran Duan and Seth Pettie. Bounded-leg distance and reachability oracles. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 436–445. Society for Industrial and Applied Mathematics, 2008.
8. Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Twentieth Acm-Siam Symposium on Discrete Algorithms, SODA 2009, New York, Ny, Usa, January*, pages 384–391, 2009.
9. Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30. ACM, 2015.
10. Daniel P Martin. Dynamic shortest path and transitive closure algorithms: A survey. *arXiv preprint arXiv:1709.00553*, 2017.

11 Liam Roditty and Michael Segal. On bounded leg shortest paths problems. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 775–784. Society for Industrial and Applied Mathematics, 2007.

12 Tong-Wook Shinn and Tadao Takaoka. Efficient graph algorithms for network analysis. In *First International Conference on Resource Efficiency in Interorganizational Networks-ResEff 2013*, page 236, 2013.

13 Tong-Wook Shinn and Tadao Takaoka. Combining all pairs shortest paths and all pairs bottleneck paths problems. In *Latin American Symposium on Theoretical Informatics*, pages 226–237. Springer, 2014.

14 Tong-Wook Shinn and Tadao Takaoka. Combining the shortest paths and the bottleneck paths problems. In *Proceedings of the Thirty-Seventh Australasian Computer Science Conference-Volume 147*, pages 13–18. Australian Computer Society, Inc., 2014.

15 Tong-Wook Shinn and Tadao Takaoka. Variations on the bottleneck paths problem. *Theoretical Computer Science*, 575:10–16, 2015. Special Issue on Algorithms and Computation.

16 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All-pairs bottleneck paths for general graphs in truly sub-cubic time. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 585–589. ACM, 2007.

17 Uri Zwick. All pairs shortest paths in weighted directed graphs – exact and almost exact algorithms. In *Foundations of Computer Science, 1998. Proceedings. Symposium on*, pages 310–319, 1998.

Single-Source Bottleneck Path Algorithm Faster than Sorting for Sparse Graphs

Ran Duan[1]
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
duanran@mail.tsinghua.edu.cn

Kaifeng Lyu
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
lkf15@mails.tsinghua.edu.cn

Yuanhang Xie
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
xieyh15@mails.tsinghua.edu.cn

Abstract

In a directed graph $G = (V, E)$ with a capacity on every edge, a *bottleneck path* (or *widest path*) between two vertices is a path maximizing the minimum capacity of edges in the path. For the single-source all-destination version of this problem in directed graphs, the previous best algorithm runs in $O(m + n \log n)$ ($m = |E|$ and $n = |V|$) time, by Dijkstra search with Fibonacci heap [Fredman and Tarjan 1987]. We improve this time bound to $O(m\sqrt{\log n} + \sqrt{mn \log n \log\log n})$, which is $O(n\sqrt{\log n \log\log n})$ when $m = O(n)$, thus it is the first algorithm which breaks the time bound of classic Fibonacci heap when $m = o(n\sqrt{\log n})$. It is a Las-Vegas randomized approach. By contrast, the s-t bottleneck path has algorithm with running time $O(m\beta(m, n))$ [Chechik et al. 2016], where $\beta(m, n) = \min\{k \geq 1 : \log^{(k)} n \leq \frac{m}{n}\}$.

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

Keywords and phrases Graph Algorithm, Bottleneck Path, Combinatorial Optimization

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.43

1 Introduction

The *bottleneck path* problem is a graph optimization problem finding a path between two vertices with the maximum flow, in which the flow of a path is defined as the minimum capacity of edges on that path. The bottleneck problem can be seen as a mathematical formulation of many network routing problems, e.g. finding the route with the maximum transmission speed between two nodes in a network, and it has many other applications such as digital compositing [11]. It is also the important building block of other algorithms, such as the improved Ford-Fulkerson algorithm [10, 12], and k-splittable flow algorithm [1]. The minimax path problem which finds the path that minimizes the maximum weight on it is symmetric to the bottleneck path problem, thus has the same time complexity.

[1] R. Duan is supported by a China Youth 1000-Talent grant.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 43; pp. 43:1–43:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 Our Results

In a directed graph $G = (V, E)$ ($n = |V|, m = |E|$), we consider the single-source bottleneck path (SSBP) problem, which finds the bottleneck paths from a given source node s to all other vertices. In the comparison-based model, the previous best time bound for SSBP is the traditional Dijkstra's algorithm [8] with Fibonacci heap [14], which runs in $O(m + n \log n)$ time. Some progress has been made for slight variants of the SSBP problem: When the graph is undirected, SSBP is reducible to minimum spanning tree [16], thus can be solved in randomized linear time [18]; for the single-source single-destination bottleneck path (s-t BP) problem in directed graphs, Gabow and Tarjan [15] showed that it can be solved in $O(m \log^* n)$ time, and this bound was subsequently improved by Chechik et al [5] to $O(m\beta(m, n))$. However, until now no algorithm is known to be better than Dijkstra's algorithm for SSBP in directed graphs. And as noted in [14], Dijkstra's algorithm can be used to sort n numbers, so a "sorting barrier", $O(m + n \log n)$, prevents us from finding a more efficient implementation of Dijkstra's algorithm.

In this paper, we present a breakthrough algorithm for SSBP that overcomes the sorting barrier. Our main result is shown in the following theorem:

▶ **Theorem 1.** *Let $G = (V, E)$ be directed graph with edge weights $w : E \to \mathbb{R}$. In comparison-based model, SSBP can be solved in expected $O\left(\sqrt{nm \log n \log\log n} + m\sqrt{\log n}\right)$ time.*

An alternative way to state our result is: for $m \leq n \log\log n$, SSBP can be solved in $O(\sqrt{nm \log n \log\log n})$ time; for $m > n \log\log n$, SSBP can be solved in $O(m\sqrt{\log n})$ time. In particular, when $m = O(n)$ the time bound is $O(n\sqrt{\log n \log\log n})$. Our algorithm is inspired by previous works on the s-t BP problem: the $O(m \log^* n)$-time algorithm by Gabow and Tarjan [15] and the $O(m\beta(m, n))$-time algorithm by Chechik et al [5]. See Section 3 for our intuitions.

1.2 Related Works

A "sorting barrier" seemed to exist for the the Minimum Spanning Tree problem (MST) for many years [3, 17, 19], but it was eventually broken by [25, 6]. Fredman and Tarjan [14] gave an $O(m\beta(m, n))$-time algorithm by introducing Fibonacci heap. The current best time bounds for MST include randomized linear time algorithm by Karger et al [18], Chazelle's $O(m\alpha(m, n))$-time deterministic algorithm [4] and Pettie and Ramachandran's optimal approach [20].

The single-source single-destination version of the bottleneck path (s-t BP) problem is proved to be equivalent to the Bottleneck Spanning Tree (BST) problem (see [5]). In the bottleneck spanning tree problem, we want to find a spanning tree rooted at source node s minimizing the maximum edge weight in it. For undirected graph, the s-t BP can be reduced to the MST problem. For directed graph, Dijkstra's algorithm [8] gave an $O(n \log n + m)$-time solution using Fibonacci heap [14]. Then Gabow and Tarjan [15] gave an $O(m \log^* n)$-time algorithm based on recursively splitting edges into levels. Recently, Chechik et al. [5] improved the time complexity of BST and BP to randomized $O(m\beta(m, n))$ time, where $\beta(m, n) = \min\{k \geq 1 : \log^{(k)} n \leq \frac{m}{n}\}$. All these algorithms are under comparison-based model. For word RAM model, an $O(m)$-time algorithm has been found by Chechik et al. [5].

For the all-pairs version of the bottleneck path (APBP) problem, we can sort all the edges and use Dijkstra search to obtain an $O(mn)$ time bound. For dense graphs, it has been shown that APBP can be solved in truly subcubic time. Shapira et al. [21] gave an $O(n^{2.575})$-time APBP algorithm on vertex-weighted graphs. Then Vassilevska et al. [23]

showed that APBP for edge-weighted graphs can be solved in $O(n^{2+\omega/3}) = O(n^{2.791})$ time based on computing $(\max, \min)$-product of real matrices, which was then improved by Duan and Pettie [9] to $O(n^{(3+\omega)/2}) = O(n^{2.686})$. Here $\omega < 2.373$ is the exponent of time bound for fast matrix multiplication [7, 24].

2 Preliminaries

For a directed graph G, we denote $w(u, v)$ to be the edge weight of $(u, v) \in E$. Without additional explanation, we use the symbol n to denote the number of nodes and m to denote the number of edges in G.

2.1 Bottleneck Path Problems

The *capacity* of a path is defined to be the minimum weight among traversed edges, i.e., if a path traverses $e_1, \ldots, e_l \in E$, then the capacity of the path is $\min_{i=1}^{l} w(e_i)$. For any $u, v \in V$, a path from u to v with maximum capacity is called a *bottleneck path from u to v*, and we denote this maximum capacity by $b(u, v)$.

▶ **Definition 2.** The *Single-Source Bottleneck Path* (SSBP) problem is: Given a directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$ and a source $s \in V$, output $b(s, t)$ for every $t \in V$, which is the maximum path capacity among all the paths from s to t.

We use the triple (G, w, s) to denote a SSBP instance with graph G, weight function $w(\cdot)$ and source node s.

It is more convenient to present our algorithm on a slight variant of the SSBP problem. We shall call it *Arbitrary-Source Bottleneck Path with Initial Capacity* (ASBPIC) problem. We assume that the edge weight $w(e)$ of an edge $e \in E$ is either a real number or infinitely large $(+\infty)$. We say an edge e is *unrestricted* if $w(e) = +\infty$; otherwise we say the edge e is *restricted*. In the ASBPIC problem, an *initial capacity* $h(v)$ is given for every node $v \in V$, and the *capacity* of a path is redefined to be the minimum between the initial capacity of the starting node and the minimum edge weights in the path, i.e., if the path starts with the node $v \in V$ and traverses $e_1, \ldots, e_l$, then its capacity is $\min\left(\{h(v)\} \cup \{w(e_i)\}_{i=1}^{l}\right)$. For any $v \in V$, a path ended with v with maximum capacity is called a *bottleneck path ended with v*, and we denote this maximum capacity as $d(v)$.

▶ **Definition 3.** The *Arbitrary-Source Bottleneck Path with Initial Capacity* (ASBPIC) problem is: Given a directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R} \cup \{+\infty\}$ and initial capacity function $h : V \to \mathbb{R} \cup \{\pm\infty\}$, output $d(v)$ for every $v \in V$, which is the maximum path capacity among all the paths ended with v.

We use the triple (G, w, h) to denote an ASBPIC instance with graph G, weight function $w(\cdot)$, and inital capacity function $h(\cdot)$.

Note that ASBPIC and SSBP are equivalent under linear-time reductions. Given an ASBPIC instance, we construct a new graph G' from G by adding a new node v_0 which has outgoing edges with weight $h(v)$ to all the nodes $v \in V$ having $h(v) > -\infty$, then it suffices to find bottleneck paths from v_0 to all other nodes in G'. On the other hand, a SSBP instance (G, w, s) can be easily reduced to the ASBPIC instance (G, w, h), where $h(s) = \max_{e \in E}\{w(e)\}$ and $h(v) = -\infty$ for all $v \neq s$.

2.2 Dijkstra's Algorithm for SSBP and ASBPIC

SSBP can be easily solved using a variant of Dijkstra's algorithm [8]. In this algorithm, each node is in one of the three status: *unsearched*, *active*, or *scanned*. We associate each node $v \in V$ with a *label* $d'(v)$, which is the maximum path capacity among all the paths from s to v that only traverse scanned nodes or v.

Initially, all the nodes are unsearched except s is active, and we set $d'(s) = +\infty$ and $d'(v) = -\infty$ for all $v \neq s$. We repeat the following step, which we call the *Dijkstra step*, until none of nodes is active:

- Select an active node u with maximum label and mark u as scanned. For every outgoing edge (u, v) of u, update the label of v by

$$d'(v) \leftarrow \max\{d'(v), \min\{d'(u), w(u, v)\}\}, \tag{1}$$

  and mark v as active if v is unsearched.

We use priority queue to maintain the order of labels for active nodes. This algorithm runs in $O(m + n \log n)$ time when Fibonacci heap [14] is used.

The algorithm we introduced above can also be adapted for solving ASBPIC. The only thing we need to change is that in the initial stage all nodes are active and $d'(v) = h(v)$ for every $v \in V$. The resulting algorithm again runs in $O(m + n \log n)$ time. We shall call these two algorithms as *Dijkstra's algorithm for SSBP* and *Dijkstra's algorithm for ASBPIC*, or simply call any of them *Dijkstra's algorithm* when no confusion can arise.

2.3 Weak and Strong Connectivity in Directed Graph

We also need some definitions about connectivity in graph theory in this paper. A directed graph is said to be *weakly-connected* if it turns to be a connected undirected graph when changing all of its directed edges to undirected edges. A directed graph is said to be *strongly-connected* if every pair of nodes can be reached from each other. A *weakly-* (or *strongly-*) *connected components* is defined to be a maximal weakly- (or strongly-) connected subgraph.

3 Intuitions for SSBP

If all the edge weights are integers and are restricted in $\{1, \ldots, c\}$, then SSBP can be solved in $O(m + c)$ time using Dijkstra's algorithm with bucket queue. If the edge weights are not necessarily small integers but all the edges given to us are already sorted by weights, then we can replace the edge weights by their ranks and use Dijkstra's algorithm with bucket queue to solve the problem in $O(m)$ time. However, edges are not sorted in general. If we sort the edges directly, then a cost of $\Omega(m \log m)$ time is unavoidable in a comparison-based model, which is more expensive than the $O(m + n \log n)$ running time of Dijkstra's algorithm.

Our algorithm is inspired by previous works on the single-source single-destination bottleneck path problem (s-t BP): the $O(m \log^* n)$-time algorithm by Gabow and Tarjan [15] and the $O(m\beta(m, n))$-time algorithm by Chechik et al [5]. Gabow and Tarjan's algorithm for s-t BP consists of several stages. Let $b(s, t)$ be the capacity of a bottleneck path from s to t. Initially, we know that $b(s, t)$ is in the interval $(-\infty, +\infty)$. In each stage, we narrow the interval of possible values of $b(s, t)$. Assume that $b(s, t)$ is known to be in the range (l, r). Let $m_{(l,r)}$ be the the number of edges with weights in the range (l, r) and k be a parameter. By applying the median-finding algorithm [2] repeatedly, we choose k thresholds $\lambda_1, \cdots, \lambda_k$ to split (l, r) into $k + 1$ subintervals $(l, \lambda_1), [\lambda_1, \lambda_2), [\lambda_2, \lambda_3), \ldots, [\lambda_{k-1}, \lambda_k), [\lambda_k, r)$ such that for each subinterval, there are $O(m_{(l,r)}/k)$ edges of weight in it. Gabow and Tarjan then show

that *locating* which subinterval contains $b(s,t)$ can be done in $O(m_{(l,r)})$ time by incremental search. Finally, the $O(m \log^* n)$ running time bound is achieved by setting k appropriately at each stage.

The algorithm by Chechik et al. is based on the framework of Gabow and Tarjan's algorithm, but instead of selecting the thresholds $\lambda_1, \ldots, \lambda_k$ by median-finding repeatedly in $O(m_{(l,r)} \log k)$ time, in this algorithm we select the k thresholds by randomly sampling in edge weights, and sort them in $O(k \log k)$ time. These thresholds partition the edges evenly with high probability, but it requires $\Omega(m \log k)$ time to compute the partition explicitly. Then they show that actually we can locate which subinterval contains $b(s,t)$ in $O(m_{(l,r)}+nk)$ (or $O(m_{(l,r)} + n \log k)$) time, without computing the explicit partition. The time bound for the overall algorithm is again obtained by setting k appropriately at each stage.

We adapt Chechik et al.'s framework for the s-t BP problem to the SSBP problem. Our SSBP algorithm actually works on an equvialent problem called ASBPIC. In ASBPIC, there is no fixed source but every node has an initial capacity, and for all destination $t \in V$ we need to compute the capacity $d(t)$ of a bottleneck path ended with t (See Section 2.1 for details). Instead of locating the subinterval for a single destination t, our algorithm locates the subintervals for all destinations $t \in V$. Thus we adopt a divide-and-conquer methodology. At each recursion, we follow the idea from Chechik et al. [5] to randomly sample k thresholds. Then we *split* the nodes into $k+1$ levels $V_0, \ldots, V_k$, where the i-th level contains the nodes t that have $d(t)$ in the i-th subinterval ($0 \le i \le k$). For each level V_i of nodes, we compute $d(t)$ for every $t \in V_i$ by reducing to solve the SSBP on a subgraph consisting of all the nodes in V_i and some of the edges connecting them. We set k to be fixed in all recursive calls, and the maximum recursion depth is $O(\log n / \log k)$ with high probability.

The split algorithm becomes the key part of our algorithm. Note that at each recursion, we should reduce or avoid the use of operations that cost time $O(\log k)$ per node or per edge (e.g., binary searching for the subinterval containing a particular edge weight). This is because that, for example, if we execute an $O(\log k)$-time operation for each edge at each recursion, then the overall time cost is $O(m \log k) \cdot O(\log n / \log k) = O(m \log n)$, which means no improvement comparing with the previous $O(m+n \log n)$-time Dijkstra's algorithm. Surprisingly, we can design an algorithm such that whenever we execute an $O(\log k)$-time operation, we can always find one edge that does not appear in any subsequent recursive calls. Thus total time complexity for such operations is $O(m \log k)$, which gives us some room to obtain a better time complexity by adjusting the parameter k.

4 Our Algorithm

Our algorithm for SSBP is as follows: Given a SSBP instance (G, w, s), we first reduce the SSBP problem to an ASBPIC instance (G, w, h), and then use a recursive algorithm (Figure 1) to solve the ASBPIC problem. The reduction is done by setting $h(s) = \max_{e \in E}\{w(e)\}$ and $h(v) = -\infty$ for all $v \neq s$ as described in the preliminaries.

For convenience, we assume that in the original graph, all edge weights are distinct. This assumption can be easily removed.

A high-level description of our recursive algorithm for ASBPIC is shown in Figure 1. For two set A and B, $A \uplus B$ stands for the union of A and B with the assumption that $A \cap B = \varnothing$. We use $E^{(r)}$ to denote the set of restricted edges in G, and similarly we use $E_i^{(r)}$ to denote the set of restricted edges in G_i for each ASBPIC instance (G_i, w_i, h_i). When the thresholds $\lambda_0, \ldots, \lambda_{k+1}$ are presence, we define the *index* of x for every $x \in \mathbb{R}$ to be the unique index i such that $\lambda_i \le x < \lambda_{i+1}$, and we denote it as $I(x)$. For $x = \pm\infty$, we define

Algorithm 1 Main Algorithm.

Input: Directed graph $G = (V, E)$, weight function $w(\cdot)$, initial capacity function $h(\cdot)$; Parameter $k \geq 1$.

Output: For each $v \in V$, output $d(v)$, the capacity of a bottleneck path ended with v.

1: **if** G is not weakly-connected **then**
2: Compute $\{d(v)\}_{v \in V}$ in each weakly-connected component recursively.
3: Let $E^{(r)} = \{e \in E \mid w(e) < +\infty\}$ be the set of restricted edges.
4: **if** $|E^{(r)}| \leq 1$ **then**
5: Compute $\{d(v)\}_{v \in V}$ in linear time and exit. ▷ Section 4.2
6: Sample $l = \min\{k, |E^{(r)}|\}$ distinct edges from $E^{(r)}$ uniformly randomly.
7: Sort the sampled edges by weights, and let $\lambda_1 < \cdots < \lambda_l$ be their weights.
8: Let $\lambda_0 = -\infty, \lambda_{l+1} = +\infty$.
9: Split V into $l + 1$ levels: $V_0 \uplus V_1 \uplus \cdots \uplus V_l$, where $V_i = \{v \in V \mid \lambda_i \leq d(v) < \lambda_{i+1}\}$. ▷ Section 4.3
10: For every level V_i, reduce the computation of $\{d(v)\}_{v \in V_i}$ to a new ASBPIC instance (G_i, w_i, h_i), where $G_i = (V_i, E_i)$ is a subgraph of G consisting of all the nodes in V_i and some of edges that connect them. Solve each (G_i, w_i, h_i) instance recursively.

$I(-\infty) = 0, I(+\infty) = l$. Note that all the subgraphs G_i at Line 10 are disjoint. We denote $r = |E| - \sum_{i=0}^{l} |E_i|$ to be the total number of edges in E that do not appear in any recursive calls of (G_i, w_i, h_i). For an edge $(u, v) \in E$, if u and v belong to different levels, then we say that (u, v) is *cross-level*. If u and v belong to the same level V_i and $w(u, v) < \lambda_i$, then we say that (u, v) is *below-level*; conversely, if $w(u, v) \geq \lambda_{i+1}$ then we say that (u, v) is *above-level*.

Besides the problem instance (G, w, h) of ASBPIC, our algorithm requires an additional integral parameter k. At the top level of recursion, we set the parameter $k = 2^{\Theta\left(\sqrt{n \log n \log\log n / m}\right)}$ if $m \leq n \log\log n$, or $k = 2^{\Theta\left(\sqrt{\log n}\right)}$ if $m > n \log\log n$. We fix this value k all through our algorithm. The value of the parameter k does not affect the correctness of our algorithm, but it controls the number of recursive calls at each recursion.

At each recursion, our algorithm first checks if G contains only one weakly-connected component. If not, then our algorithm calls itself to compute d in each weakly-connected component recursively. Now we can assume that G is weakly-connected (so $n \leq m$).

If the number of restricted edges is no more than 1, we claim that we can compute $d(v)$ for all v in linear time. The specific algorithm will be introduced in Section 4.2.

▶ **Lemma 4.** *ASBPIC can be solved in $O(m)$ time if there is at most one restricted edge.*

If the number of restricted edges is more than 1, then our algorithm first sample $l = \min\{k, |E^{(r)}|\}$ distinct edges from $E^{(r)}$ uniformly randomly and sort them by weights, that is, if the number of restricted edges is more than k, then we sample k distinct restricted edges and sort them; otherwise we just sort all the restricted edges. Let λ_i be the weight of the edge with rank i $(1 \leq i \leq l)$ and $\lambda_0 = -\infty$, $\lambda_{l+1} = +\infty$.

Next, we split V into $l + 1$ levels of nodes $V = V_0 \uplus V_1 \uplus \cdots V_l$, where the i-th level of nodes is $V_i = \{v \in V \mid I(d(v)) = i\} = \{v \in V \mid \lambda_i \leq d(v) < \lambda_{i+1}\}$. The basic idea of the split algorithm is: we run Dijkstra's algorithm for ASBPIC on the graph produced by mapping every edge weight $w(e)$ and initial capacity $h(v)$ in G to their indices $I(w(e))$ and $I(h(v))$, and we obtain the final label value $d'(v)$ for each node $v \in V$ (Remember that $d'(v)$ is the label of v in Dijkstra's algorithm). It is easy to show that the final label value $d'(v)$ equals $I(d(v))$, so the nodes can be easily split into levels according to their final labels. The specific

split algorithm will be introduced in Section 4.3. The time complexity for a single splitting is given below. In Theorem 9 we show that this implies that the total time cost for splitting is $O(m \log n / \log k + n \log n \log\log k / \log k + m \log k)$.

▶ **Lemma 5.** *Splitting V into levels at Line 9 can be done in $O(m + n \log\log k + r \log k)$.*

Finally, for every level V_i, we compute $d(\cdot)$ for nodes in this level by reducing to a new ASBPIC instance (G_i, w_i, h_i), where G_i is a subgraph of G consisting of all the nodes in V_i and some of edges that connect them. We solve each new instance by a recursive call. The construction of (G_i, w_i, h_i) is as follows:

- $G_i = (V_i, E_i)$, where V_i is the nodes at level i in G, and E_i is the set of edges which connect two nodes at level i and are not below-level, i.e., $E_i = \{(u, v) \in E \mid u, v \in V_i, w(u, v) \geq \lambda_i\}$;
- For any $e \in E_i$, $w_i(e) = +\infty$ if e is above-level; otherwise $w_i(e) = w(e)$;
- For any $v \in V_i$, $h_i(v) = \max(\{h(v)\} \cup \{w(u, v) \in E \mid u \in V_{i+1} \uplus \cdots \uplus V_l\})$.

▶ **Lemma 6.** *We can construct all the new ASBPIC instances $(G_0, w_0, h_0), \ldots, (G_l, w_l, h_l)$ in $O(m)$ time. For $v \in V_i$, the value of $d(v)$ in the instance (G_i, w_i, h_i) exactly equals to the value of $d(v)$ in the instance (G, w, h).*

Proof. We can construct all these instances (G_i, w_i, h_i) for all $0 \leq i \leq l$ by linearly scanning all the nodes and edges, which runs in $O(m)$ time. We prove the correctness by transforming (G, w, h) to (G_i, w_i, h_i) step by step, while preserving the values of $d(v)$ for all $v \in V_i$.

By definition, $\lambda_i \leq d(v) < \lambda_{i+1}$ for all $v \in V_i$. We can delete all the nodes at level less than i and delete all the edges with weight less than λ_i, since no bottleneck path ended with a node in V_i can traverse them. Also, for every edge e with weight $w(e) \geq \lambda_{i+1}$, we can replace the edge weight with $+\infty$ since $w(e)$ is certainly not the minimum in any path ended with a node in V_i.

For every edge $e = (u, v)$ where $u \in V_{i+1} \uplus \cdots \uplus V_l$ and $v \in V_i$, the edge weight $w(e)$ must be less than λ_{i+1}, otherwise $d(v) \geq \min\{d(u), w(u, v)\} \geq \lambda_{i+1}$ leads to a contradiction. Thus contracting all the nodes in $V_{i+1} \uplus \cdots \uplus V_l$ to a single node v_0 with infinite initial capacity is a transformation preserving the values of $d(v)$ for all $v \in V_i$. Finally, our construction follows by taking $h_i(v)$ to be the maximum between the weight of incoming edges from v_0 and the initial capacity $h(v)$ for every $v \in V_i$. ◀

▶ Remark. In any subsequent recursive calls of (G_i, w_i, h_i), neither cross-level nor below-level edges will appear, and all the above-level edges will become unrestricted. Also, it is easy to see that r is just the total number of cross-level and below-level edges (Recall that r is the number of edges that do not appear in any recursive calls).

4.1 Running Time

First we analyze the maximum recursion depth. The following lemma shows that randomly sampled thresholds evenly partition the restricted edges with high probability.

▶ **Lemma 7.** *Let $E^{(r)} = \{e_1, \ldots, e_q\}$ be q restricted edges sorted by their weights in E. Let $f_1, \ldots, f_k$ be $k \geq 2$ random edges sampled from $E^{(r)}$ such that $w(f_1) < w(f_2) < \cdots < w(f_k)$. Let $\lambda_i = w(f_i)$ for $i = 1, \ldots, k$, and $\lambda_0 = -\infty$, $\lambda_{k+1} = +\infty$. Let $F_i = \{e \in E \mid \lambda_i \leq w(e) < \lambda_{i+1}\}$. Then for every $t > 0$, $\max_{0 \leq i \leq k}\{|F_i|\} < tq \log k / k$ holds with probability $1 - k^{-\Omega(t)}$.*

Proof. Let $M = tq \log k / k$. If $\max_{0 \leq i \leq k}\{|F_i|\} \geq M$, then there exists an edge e_p such that e_p is chosen but for any $p + 1 \leq j < p + M$, e_j is not chosen. Note that when p is given, this

event happens with probability $\leq k \cdot (1/q) \cdot \prod_{i=1}^{k-1}((q-M-i)/(q-i)) \leq (k/q) \cdot (1-M/q)^{k-1}$. By the union bound for all possible p, we have

$$\Pr\left[\max_{0\leq i\leq k}\{|F_i|\} \geq tq\log k/k\right] \leq k(1-t\log k/k)^{k-1} \leq k^{-\Omega(t)},$$

which completes the proof. ◀

For our purposes it is enough to analyze the case that $k = 2^{\Omega\left(\sqrt{\log n}\right)}$. The following lemma gives a bound for the maximum recursion depth using Lemma 7.

▶ **Lemma 8.** *For $k = 2^{\Omega\left(\sqrt{\log n}\right)}$ where n is the number of nodes at the top level of recursion, the maximum recursion depth is $O(\log n/\log k)$ with probability $1 - n^{-\omega(1)}$.*

Proof. It is not hard to see that the total number of recursive calls of our main algorithm is $O(m)$. Applying Lemma 7 with $t = \Theta(\log n)$ and the union bound for all recursive calls, we know that with probability at least $1 - k^{-\Omega(t)} \cdot O(m) = 1 - n^{-\Omega(\log n)^{3/2}}$, after every split with $|E^{(r)}| > k$, the number of restricted edges in G_i is less than $(t\log k/k) \cdot |E^{(r)}|$ for every (G_i, w_i, h_i). Thus after $O(\log m/\log(k/(t\log k))) = O(\log n/\log k)$ levels of recursion, every ASBPIC instance (G, w, h) has $|E^{(r)}| \leq k$, and this means that in any recursive call of (G_i, w_i, h_i), the graph G_i has at most one restricted edge, which will be directly solved at Line 5. ◀

The overall time complexity of our algorithm is given by the following theorem:

▶ **Theorem 9.** *For $k = 2^{\Omega\left(\sqrt{\log n}\right)}$, with probability $1 - n^{-\omega(1)}$, our main algorithm shown in Figure 1 runs in $O(n\log n\log\log k/\log k + m\log n/\log k + m\log k)$ time.*

Proof. Let $r = |E| - \sum_{i=0}^{l}|E_i|$, $r' = |E^{(r)}| - \sum_{i=0}^{l}|E_i^{(r)}|$. First we show that the running time in each recursive call is $O(m + n\log\log k + (r+r')\log k)$.

In each recursive call of our algorithm, the time cost for sorting at Line 7 is $O(l\log l)$. For the sample edge e_i with rank i, either V_i is not empty, or this edge is cross-level, below-level, or above-level. Let l_1 be the number of edges in the former case, and l_2 be the number of edges in the latter case. For the former case, note that we only run the split algorithm for weakly-connected graphs, so there are at least $l_1 - 1$ cross-level edges, which implies $l_1 \leq r + 1$. For the latter case, e_i becomes unrestricted or does not appear for every G_i, so $l_2 \leq r'$. Thus $l = l_1 + l_2 \leq r + r' + 1$ and the time cost for sorting is $O((r+r')\log k)$.

By Lemma 5, the split algorithm runs in $O(m + n\log\log k + r\log k)$ time in each recursive call. All other parts of our algorithm run in linear time. Thus the running time for each recursion is $O(m + n\log\log k + (r+r')\log k)$. Note that the recursion depth is $O(\log n/\log k)$ with probability $1 - n^{\omega(1)}$. We can complete the proof by adding the time cost for all the $O(m)$ recursive calls together. ◀

Finally, we can get our main result by setting $k = 2^{\Theta\left(\sqrt{n\log n\log\log n/m}\right)}$ for $m \leq n\log\log n$ and setting $k = 2^{\Theta\left(\sqrt{\log n}\right)}$ for $m > n\log\log n$.

▶ **Theorem 10.** *For $m \leq n\log\log n$, SSBP can be solved in $O(\sqrt{nm\log n\log\log n})$ time with high probability; For $m > n\log\log n$, SSBP can be solved in $O(m\sqrt{\log n})$ time with high probability.*

▶ Remark. The above time bounds are also true for expected running time. It can be easily derived from the fact that the worst-case running time is at most $n^{O(1)}$.

In the rest of this section, we introduce the algorithm for ASBPIC with at most one restricted edge and the split algorithm.

4.2 Algorithm for the Graph with at Most One Restricted Edge

We introduce our algorithm for the graph with at most one restricted edge in the following two lemmas.

▶ **Lemma 11.** *For a given ASBPIC instance (G, w, h), if there is no restricted edge in G, then the values of $d(v)$ for all $v \in V$ can be computed in linear time.*

Proof. G contains only unrestricted edges, so for every $v \in V$, $d(v)$ is just equal to the maximum value of $h(u)$ among all the nodes u that can reach v. If v_1 and v_2 are in the same strongly-connected component, then $d(v_1) = d(v_2)$. Thus we can use Tarjan's algorithm [22] to contract every strongly-connected component to a node. The initial capacity of a node is the maximum $h(u)$ for all u in the component, and the capacity of an edge between nodes is the maximum among edges connecting components. Then Dijkstra approach on DAG takes linear time. ◀

▶ **Lemma 12.** *For a given ASBPIC instance (G, w, h), if there is exactly one restricted edge in G, then the values of $d(v)$ for all $v \in V$ can be computed in linear time.*

Proof. Let $e_0 = (u_0, v_0)$ be the only restricted edge in G. There are two kinds of paths in G:

1. Paths that do not traverse e. We remove e from G and use the algorithm in Lemma 11 to get $d(v)$ for every node $v \in V$.
2. Paths that traverse e. Note that $d(u_0)$ got in the previous step is the maximum capacity to u_0 through only restricted edges. Then we update $d(v)$ by $\max\{d(v), \min\{d(u_0), w(u_0, v_0)\}\}$ for every node v that can be reached from v_0.

We output the values of $d(\cdot)$ after these two kinds of updates, then all the paths should have been taken into account. ◀

4.3 Split

Now we introduce the split algorithm at Line 9 in our main algorithm. As before, we use the notation $I(x)$ for the index of a value x and $d'(v)$ is the label of v in Dijkstra's algorithm. The goal of this procedure is to split V into $l + 1$ levels, $V = V_0 \uplus V_1 \uplus \cdots \uplus V_l$, where $V_i = \{v \in V \mid I(d(v)) = i\}$. We need to show that this can be done in $O(m + n \log\log k + r \log k)$ time, where $r = |E| - \sum_{i=0}^{l} |E_i|$ is the total number of edges in E that do not appear in any (G_i, w_i, h_i).

A straightforward approach to achieve this goal is to use Dijkstra's algorithm as described in Section 2.2. We map all the edge weights and initial capacities to their indices using binary searches, and run Dijkstra's algorithm for ASBPIC. The output should be exactly $d'(v) = I(d(v))$ for every v. However, this approach is rather inefficient. Evaluating the index $I(x)$ for a given x requires $\Omega(\log l)$ time in a comparison-based model, thus in total, this algorithm needs $\Omega(n \log l)$ time to compute indices, and this does not meet our requirement for the running time.

The major bottleneck of the above algorithm is the inefficient index evaluations. Our algorithm overcomes this bottleneck by reducing the number of index evaluations for both edge weights and initial capacities to be at most $O(r + n/\log l)$.

4.3.1 Index Evaluation for Edge Weights

First we introduce our idea to reduce the number of index evaluations for edge weights. Recall that in Dijkstra's algorithm for ASBPIC we maintain a label $d'(v)$ for every $v \in V$.

In every Dijkstra step, we extract an active node u with the maximum label, and for all edges $(u, v) \in E$ we compute $\min\{d'(u), I(w(u, v))\}$ to update $d'(v)$. In the straightforward approach we evaluate every $I(w(u, v))$ using binary search, but actually this is a big waste:

1. If $w(u, v) \geq \lambda_{d'(u)}$, then $\min\{d'(u), I(w(u, v))\} = d'(u)$, so there is no need to evaluate $I(w(u, v))$.
2. If $w(u, v) < \lambda_{d'(u)}$, then $\min\{d'(u), I(w(u, v))\} = I(w(u, v))$, so we do need to evaluate $I(w(u, v))$. However, it can be shown that (u, v) is either a cross-level edge or a below-level edge, so (u, v) will not appear in any subsequent recursive calls of (G_i, w_i, h_i).

Using the method discussed above, we can reduce the number of index evaluations for edge weights to be at most $r = |E| - \sum_{i=0}^{l} |E_i|$ in Dijkstra's algorithm. Lemma 14 gives a formal proof for this.

4.3.2 Index Evaluation for Initial Capacities

Now we introduce our idea to reduce the number of index evaluations for initial capacities. Recall that in Dijkstra's algorithm, we need to initialize the label $d'(v)$ to be $I(h(v))$ for each $v \in V$, and maintain a priority queue for the labels of all active nodes. If we evaluate every $I(h(v))$ directly, we have to pay a time cost $\Omega(n \log l)$.

In our split algorithm, we first find a spanning tree T of G after replacing all the edges with undirected edges. Then we partition the tree T into $b = O(n/s)$ edge-disjoint (but not necessarily node-disjoint) subtrees, $T_1, \ldots, T_b$, each of size $O(s)$. This partition can be found in $O(n)$ time using a slight variant of the topological partition algorithm in Frederickson's paper [13].

▶ **Theorem 13.** *(see [13]) Given a tree $T = (V, E)$ with n nodes and given an integer $1 \leq s \leq n$, there exists a linear time algorithm that can partition T into edge-disjoint subtrees, $T_1, \ldots, T_b$, such that the number of nodes in each subtree is in the range $[s, 3s)$.*

We form b groups of nodes, $U_1, \ldots, U_b$, where U_i is the group of nodes that are in the i-th subtree T_i. In the running of Dijkstra's algorithm, we divide the active nodes in V into two kinds:

1. **Updated node**. This is the kind of node v that has $d'(v)$ already been updated by Dijkstra's update rule (1), which means $d'(v) = \min\{d'(u), I(w(u, v))\} \geq I(h(v))$ for some u after a previous update. The value of $\min\{d'(u), I(w(u, v))\}$ is evaluated according to Section 4.3.1, so the value of $d'(v)$ can be easily known. We can store such nodes in buckets to maintain the order of their labels.
2. **Initializing node**. This is the kind of node v whose $d'(v)$ has not been updated by Dijkstra's update rule (1), so $d'(v) = I(h(v))$. However, we do not store the value of $I(h(v))$ explicitly. For each group U_i, we put the initializing nodes in U_i into a priority queue in descending order by their initial capacities $h(\cdot)$. We only compute the value of $I(h(v))$ when v is the maximum in its group, and use buckets to maintain the maximum values from all groups.

At each iteration in Dijkstra's algorithm, we extract the active node with the maximum label among the updated nodes and initializing nodes and mark it as scanned. For the case that the maximum node $v \in U_j$ is an initializing node, we remove v from the priority queue of U_j, and compute $d'(u) = I(h(u))$ for the new maximum node u. However, if we compute this value directly using an index evaluation for $h(u)$, then we will suffer a total cost $\Omega(n \log l)$, which is rather inefficient.

Algorithm 2 Split Algorithm.

Input: Directed graph $G = (V, E)$, weight function $w(\cdot)$, initial capacity function $h(\cdot)$, $l + 2$ thresholds $-\infty = \lambda_0 < \lambda_1 < \cdots < \lambda_l < \lambda_{l+1} = +\infty$

Output: Split V into $V_0 \uplus V_1 \uplus \cdots \uplus V_l$ where $V_i = \{v \in V \mid \lambda_i \le d(v) < \lambda_{i+1}\}$.

1: Let $s = \min\{\lceil \log l \rceil, n\}$. Find a spanning tree T of G after replacing all the edges with undirected edges and partition T into $b = O(n/s)$ edge-disjoint subtrees $T_1, \ldots, T_b$, each of size $O(s)$. Form b groups of nodes, $U_1, \ldots, U_b$, where U_i is the group of nodes that are in the i-th subtree T_i.
2: For nodes in each group, we maintain a priority queue implemented by binary heap by the order of $h(\cdot)$.
3: Evaluate $I(h(v))$ only for the maximum node in each group, and initialize $l + 1$ buckets $B_0, \ldots, B_l$ which store the groups according to the index of $h(\cdot)$ of their maximum nodes.
4: Initialize $l + 1$ buckets $C_0, \ldots, C_l \leftarrow \varnothing$ which store the active nodes according to $d'(\cdot)$.
5: **for** $i \leftarrow l, l-1, \ldots, 1, 0$ **do**
6: **for all** $U \in B_i$ **do**
7: **while** $U \neq \varnothing$ **and** the maximum node u in U has $I(h(u)) = i$ **do**
8: Extract u from U and put u into C_i
9: $d'(u) \leftarrow i$
10: **while** $C_i \neq \varnothing$ **do**
11: Extract a node u from C_i
12: **for all** $(u, v) \in E$ **do**
13: $\bar{w} \leftarrow \min\{d'(u), I(w(u, v))\}$. (Evaluate $I(w(u, v))$ only if $w(u, v) < \lambda_{d'(u)}$)
14: **if** $\lambda_{\bar{w}} > h(v)$ **then**
15: **if** $d'(v)$ does not exist **or** $\bar{w} > d'(v)$ **then**
16: Delete v from $C_{d'(v)}$
17: Delete v from the group U containing it (if any) and put v into $C_{\bar{w}}$
18: $d'(v) \leftarrow \bar{w}$
19: **while** B_i contains at least one non-empty group **do**
20: Extract a non-empty group U from B_i
21: Evaluate $I(h(u))$ for the maximum node u in U and put U into $B_{I(h(u))}$
22: Split V according to d' and return

The idea to speed up is to check whether $d'(u) = d'(v)$ before performing an index evaluation. This can be done in $O(1)$ time since we can know the corresponding interval of $h(u)$ from the value of $d'(v)$ if $I(h(u)) = d'(v)$. We only actually evaluate $I(h(u))$ if $d'(u) \neq d'(v)$ after the Dijkstra step scans all nodes with level $d'(v)$. In this way, we can always ensure that the number of index evaluations in a group never exceeds the number of different final label values $d'(\cdot)$ in this group. Indeed, it can be shown that if there are c different final label values $d'(\cdot)$ in a group U_i, then there must be at least $c - 1$ cross-level edges in T_i, which implies that the number of index evaluations for initial capacities should be no greater than $r + b$. (Remember b is the number of groups in the partition.) Lemma 15 gives a formal proof for this.

4.3.3 The $O(m + n \log\log k + r \log k)$-time Split Algorithm

Now we are ready to introduce our split algorithm in full details. A pseudocode is shown in Figure 2. During the search at Line 5 - 21, B_i may contain groups with maximum nodes not at the i-th level, e.g., when the maximum node in a group U is deleted at Line 17 and

$I(h(u))$ of the new maximum node u in U has not been evaluated yet. We have the following observations:

- For all $v \in V$, $d'(v)$ is non-decreasing, and at the end we have $d'(v) = I(d(v))$.
- All the binary heaps of U_i are deletion only (so the priority queue can also be implemented by a sorted linked list).
- Only at Line 3, 13 and 21 we need to evaluate the index of $h(\cdot)$ of a node. Each index evaluation costs $O(\log l)$ time.
- The numbers of executions of the while loops at Line 7 - 9, Line 10 - 18, Line 19 - 21 are all bounded by $O(n)$.

Our algorithm is an implementation of Dijkstra's algorithm, so the correctness is obvious. The running time analysis is based on the following lemmas:

▶ **Lemma 14.** *If we evaluate $I(w(u,v))$ at Line 13, then the edge (u,v) will not be in any recursive calls of (G_i, w_i, h_i).*

Proof. We evaluate $I(w(u,v))$ only if $w(u,v) < \lambda_{d'(u)}$, so $\bar{w} = I(w(u,v))$ right after Line 13. Since u has already been scanned, $d'(u)$ here is just its final value $I(d(u))$. Note that $I(d(v)) \geq \min\{I(d(u)), I(w(u,v))\} = I(w(u,v))$. If $I(d(v)) > I(w(u,v))$, then $I(w(u,v))$ is smaller than both $I(d(u))$ and $I(d(v))$, thus (u,v) is a below-level edge or cross-level edge. If $I(d(v)) = I(w(u,v))$, then $I(d(v)) < I(d(u))$ and (u,v) is a cross-level edge. ◀

▶ **Lemma 15.** *At Line 3 and 21, if we evaluate $I(h(u))$ for c nodes u in some group U_i, then the number of different final label values $d'(\cdot)$ in U_i is at least c. Thus, the number of edges in the subtree T_i corresponding to U_i that do not appear in any recursive calls of (G_i, w_i, h_i) is at least $c-1$.*

Proof. At line 21, $I(d(u))$ must be less than i; otherwise, u should have been extracted at Line 11 before extracting u at Line 20, which is impossible. Also note that $d(u) \geq h(u)$, so $I(h(u)) \leq I(d(u))$ for all $u \in V$. Thus, if $u_1 \in U_i$ is evaluated at Line 3 and there are $c-1$ nodes $u_2, \ldots, u_c$ extracted from U_i at Line 21, then $I(d(u_1)), I(d(u_2)), \ldots, I(d(u_c))$ should be in c distinct ranges: $[I(h(u_1)), +\infty), [I(h(u_2)), I(h(u_1))), \ldots, [I(h(u_c)), I(h(u_{c-1})))$, which implies that the number of different final values of $d'(u)$ in U_i is at least c.

Suppose we remove all the cross-level edges in T_i, i.e., remove all the edges (u,v) in T_i whose final values of $d'(u)$ and $d'(v)$ differ. Then the tree should be decomposed into at least c components since there are at least c different final values of $d'(u)$ in U_i. Thus there are at least $c-1$ cross-level edges in T_i. ◀

Finally, we can derive the $O(m + n \log\log k + r \log k)$ time bound for our split algorithm.

Proof for Lemma 5. By Lemma 14, the number of index evaluations at Line 13 is at most r. Let c_i be the number of index evaluations at Line 3 and 21 for nodes in the group U_i. Then by Lemma 15, $\sum_{i=1}^{b} c_i \leq r + b$. Thus the total number of index evaluations in our split algorithm can be bounded by $2r + b \leq O(r + n/\log l)$, which costs $O(r \log l + n)$ time.

The extraction operations of the maximum element at Line 8 and 21 and the deletion operation at Line 17 runs in $O(\log s) \leq O(\log\log l)$ time per operation. Note that there are $O(n)$ such operations, thus all these operations cost $O(n \log\log l)$ time in total.

It can be easily seen that all other parts of our split algorithm runs in linear time, so the overall time complexity is $O(m + n\log\log l + r\log l) \leq O(m + n\log\log k + r\log k)$. ◀

5 Discussion

We give an improved algorithm for solving SSBP in $O(m\sqrt{\log n} + \sqrt{mn \log n \log\log n})$ faster than sorting for sparse graphs. This algorithm is a breakthrough compared to traditional Dijkstra's algorithm using Fibonacci heap. There are some open questions remained to be solved. Can our algorithm for SSBP be further improved to $O(m\beta(m, n))$, which is the time complexity for the currently best algorithm for s-t BP? Can we use our idea to obtain an algorithm for SSBP that runs faster than Dijkstra in word RAM model?

References

1 Georg Baier, Ekkehard Köhler, and Martin Skutella. On the k-splittable flow problem. In *European Symposium on Algorithms*, pages 101–113. Springer, 2002.

2 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973.

3 O Boruvka. O jistem problemu minimalnim, praca moravske prirodovedecke spolecnosti 3, 1926.

4 B. Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *J. ACM*, 47(6):1028–1047, 2000.

5 Shiri Chechik, Haim Kaplan, Mikkel Thorup, Or Zamir, and Uri Zwick. Bottleneck paths and trees and deterministic graphical games. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 47. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

6 D. Cheriton and R. E. Tarjan. Finding minimum spanning trees. *SIAM J. Comput.*, 5:724–742, 1976.

7 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990. Computational algebraic complexity editorial. doi:10.1016/S0747-7171(08)80013-2.

8 Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

9 Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 384–391. Society for Industrial and Applied Mathematics, 2009.

10 Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

11 Elena Fernandez, Robert Garfinkel, and Roman Arbiol. Mosaicking of aerial photographic maps via seams defined by bottleneck shortest paths. *Oper. Res.*, 46(3):293–304, 1998. doi:10.1287/opre.46.3.293.

12 L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(0):399–404, 1956. doi:10.4153/cjm-1956-045-5.

13 Greg N Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal on Computing*, 14(4):781–798, 1985.

14 Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

15 Harold N Gabow and Robert E Tarjan. Algorithms for two bottleneck optimization problems. *Journal of Algorithms*, 9(3):411–417, 1988.

16 T. C. Hu. The maximum capacity route problem. *Operations Research*, 9(6):898–900, 1961. URL: http://www.jstor.org/stable/167055.

17 Vojtech Jarnık. O jistém problému minimálnım. *Práca Moravské Prırodovedecké Spolecnosti*, 6:57–63, 1930.

18 David R Karger, Philip N Klein, and Robert E Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM (JACM)*, 42(2):321–328, 1995.

19 Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

20 S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49(1):16–34, 2002.

21 Asaf Shapira, Raphael Yuster, and Uri Zwick. All-pairs bottleneck paths in vertex weighted graphs. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 978–985. Society for Industrial and Applied Mathematics, 2007.

22 Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

23 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All-pairs bottleneck paths for general graphs in truly sub-cubic time. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 585–589. ACM, 2007.

24 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 887–898, New York, NY, USA, 2012. ACM. doi:10.1145/2213977.2214056.

25 A. C. Yao. An $O(|E| \log \log |V|)$ algorithm for finding minimum spanning trees. *Info. Proc. Lett.*, 4(1):21–23, 1975.

Improved Time Bounds for All Pairs Non-decreasing Paths in General Digraphs

Ran Duan[1]
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
duanran@mail.tsinghua.edu.cn

Yong Gu
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
guyong12@mails.tsinghua.edu.cn

Le Zhang
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
le-zhang12@mails.tsinghua.edu.cn

Abstract

We present improved algorithms for solving the All Pairs Non-decreasing Paths (APNP) problem on weighted digraphs. Currently, the best upper bound on APNP is $\tilde{O}(n^{(9+\omega)/4}) = O(n^{2.844})$, obtained by Vassilevska Williams [TALG 2010 and SODA'08], where $\omega < 2.373$ is the usual exponent of matrix multiplication. Our first algorithm improves the time bound to $\tilde{O}(n^{2+\omega/3}) = O(n^{2.791})$. The algorithm determines, for every pair of vertices s, t, the minimum last edge weight on a non-decreasing path from s to t, where a non-decreasing path is a path on which the edge weights form a non-decreasing sequence. The algorithm proposed uses the combinatorial properties of non-decreasing paths. Also a slightly improved algorithm with running time $O(n^{2.78})$ is presented.

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

Keywords and phrases Graph algorithms, Matrix multiplication, Non-decreasing paths

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.44

Funding This work was partially supported by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61361136003.

Acknowledgements The authors thank the anonymous reviewers for the constructive comments.

1 Introduction

Given a digraph with arbitrary real weights, a non-decreasing path is a path on which the edge weights form a non-decreasing sequence. Two of the problems studied on non-decreasing paths are the *Single Source Non-decreasing Paths* (SSNP) problem and the *All Pairs Non-decreasing Paths* (APNP) problem. The problem of SSNP was first studied by Minty [13]. The motivation is a train scheduling problem, as reviewed in [22]. Every train stop is mapped to a vertex. A train from stop v_1 with departure time t_1 to stop v_2 with arrival time t_2 is mapped to a vertex v with two edges (v_1, v), (v, v_2), of which the weights are t_1, t_2 resp. Now a trip from s to t is possible only when there exists a non-decreasing path from s to t in the

[1] Supported by a China Youth 1000-Talent grant.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 44; pp. 44:1–44:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

constructed digraph. As said in [22], for SSNP a folklore modification of Dijkstra's algorithm [5], implemented using Fibonacci heaps [8], gives the running time of $O(m + n \log n)$, where m, n are the number of edges and vertices resp. In the word RAM model the first linear-time algorithm was given by Vassilevska Williams [22]. With a slight modification, the algorithm also runs in $O(m \log \log n)$ time in the standard addition-comparison model.

A restriction of APNP in vertex-weighted digraphs is computationally equivalent to the problem of *Maximum Witness for Boolean Matrix Multiplication* (MWBMM) [22]. (Note that the complexity of computing MWBMM is at least $\Omega(n^\omega)$ [17].) An algorithm of $O(n^{2+\mu})$ time for the latter problem was given by Czumaj et al. [3], where μ satisfies the equation $\omega(1, \mu, 1) = 1 + 2\mu$ and $\omega(1, \mu, 1)$ is the exponent of the multiplication of an $n \times n^\mu$ matrix by an $n^\mu \times n$ matrix. Currently, the best available bounds on $\omega(1, \mu, 1)$ by Le Gall and Urrutia [11] imply that $\mu < 0.5286$. The first truly sub-cubic algorithm for edge-weighted APNP was also presented in [22]. The algorithm originally runs in $\tilde{O}(n^{(15+\omega)/6}) = O(n^{2.896})$ time[2] based on an $O(n^{2+\omega/3})$-time $(\min, \leq)$-product ($\min_k\{B[k, j] \mid A[i, k] \leq B[k, j]\}$ for $(\min, \leq)(A, B)$) algorithm from [21], where ω is the exponent of square matrix multiplication. The best upper bound on ω is currently $\omega < 2.373$ [12, 23]. By using a faster $O(n^{(3+\omega)/2})$-time algorithm from [6] for $(\min, \leq)$-product, the result can be improved to $\tilde{O}(n^{(9+\omega)/4})$ as indicated in the abstract. These two algorithms for $(\min, \leq)$-product are from [21] and [6] resp. The faster algorithm of [6] utilizes a simple technique called *row-balancing*, which we introduce in details in Section 2.

A closely related problem of APNP is the *All Pairs Bottleneck Paths* (APBP) problem, where the bottleneck weight of a path is the smallest weight of an edge on the path. Intuitively for a digraph with non-negative edge weights, APBP determines, for every pair of vertices s, t, the maximum amount of flow that can be routed from s to t along any single path. As indicated in [22], APNP is at least as hard as APBP. We briefly explain it here. Consider an $O(n^{3-\delta})$-time algorithm for APNP with $0 < \delta \leq 1$. Now to compute the $(\min, \leq)$-product of two $n \times n$ matrices A, B, a tripartite digraph $G' = (V_1 \cup V_2 \cup V_3, E')$ can be constructed as follows. The edge from $i \in V_1$ to $k \in V_2$ is represented by $A[i, k]$; the edge from $k \in V_2$ to $j \in V_3$ is similarly represented by $B[k, j]$. Hence the $(\min, \leq)$-product is solved in $O(n^{3-\delta})$ time. The $(\max, \min)$-product ($\max_k \min\{A[i, k], B[k, j]\}$ for $(\max, \min)(A, B)$) is a combination of two variants of $(\min, \leq)$-product. Therefore it can also be computed in $O(n^{3-\delta})$ time. If $(\max, \min)$-product is computable in $O(n^{3-\delta})$ time, then APBP is computable in $O(n^{3-\delta})$ time, as $(\mathbb{R}, \min, \max, \infty, -\infty)$ is a closed semiring [21]. There are several results on the APBP problem. On vertex-weighted digraphs, Shapira et al. [17] showed that APBP can be solved in $O(n^{2+\mu})$ time. They also demonstrated the computational equivalence (up to constant factors) among vertex-weighted APBP and several other problems. The forementioned problem of MWBMM is one of them. The first truly sub-cubic algorithm for edge-weighted APBP was given by Vassilevska et al. [21], with the running time of $O(n^{2+\omega/3})$. Later an improved algorithm was proposed in [6], which runs in $O(n^{(3+\omega)/2})$ time. There are several variants of APBP. One is the *All Pairs Bottleneck Shortest Paths* (APBSP) problem, which for every pair of vertices u, v, determines a path with the maximum bottleneck weight among all the shortest paths from u to v. The shortest paths here are measured w.r.t. the unweighted distances. On edge-capacitated digraphs, Vassilevska et al. [21] gave an algorithm for APBSP with the running time $\tilde{O}(n^{(15+\omega)/6})$, which was improved to $\tilde{O}(n^{(3+\omega)/2})$ in [6]. On vertex-capacitated digraphs, Shapira et al. [17] presented an $\tilde{O}(n^{(8+\mu)/3})$-time algorithm. Also the result was later improved by [6].

[2] Here $\tilde{O}(\cdot)$, as usual, hides poly-logarithmic factors.

Fast matrix multiplication algorithms have numerous applications in other graph problems as well. We list here only a subset of them, which includes finding a maximum triangle in vertex-weighted graphs [19, 20, 4], to obtain the All Pairs Shortest Paths [9, 10, 16, 18, 1, 25, 7], finding minimum weight cycles in directed or undirected graphs with integral edge weights [14, 24, 2, 15].

Our Results: We give faster algorithms for solving APNP on digraphs. The results are listed in Theorem 1, which follows directly from Theorem 18 and Theorem 21.

▶ **Theorem 1.** *Let $G = (V, E, w)$ be a real edge-weighted digraph. There exists a* deterministic *algorithm which solves the problem of APNP in $\tilde{O}(n^{2+\omega/3}) = O(n^{2.791})$ time. There also exists another slightly faster* deterministic *algorithm which runs in $O(n^{2.78})$ time using rectangular matrix multiplications.*

A High-level Overview: The problem of APNP can be solved by running (essentially) $(n-1)$ steps of the $(\min, \leq)$-product of the adjacency matrix of the input digraph. We utilize the fact that the adjacency matrix in the computation is always *fixed*, and therefore when the rows of it all have a bounded number of $(< \infty)$ entries, we find a simple (and faster) alternative to the repeated applications of the $(\min, \leq)$-product. To make use of this simple alternative, we partition the input digraph into many sparse subgraphs, and compute APNP by considering the edges one subgraph by one subgraph. However, even in a sparse subgraph the number of $(< \infty)$ entries in a row might be still as large as $\Omega(n)$. Thus we further classify a row as type *low* or *high*, where a high row corresponds to a high out-degree vertex in the subgraph. The simple alternative is used to replace the $(\min, \leq)$-product for the submatrix consisting of low rows. Also it computes the portion of a non-decreasing path until the *first* vertex of high out-degree. The remaining portion from this high out-degree vertex to the destination can be constructed by the relevant queries on a slightly modified data structure from [22]. However, to get an efficient algorithm we should first make sure of the queries worthy of asking, which is actually a weaker problem of the existence of non-decreasing paths.

The paper is organized as follows. We introduce the preliminaries required in the next section. Then in Section 3 we propose an algorithm for a simple case, which is also a sub-routine of the improved algorithms presented in Section 4.

2 Preliminaries

Given a real edge-weighted digraph $G = (V, E, w)$, where $w : E \to \mathbb{R}$ is a weight function defined on its edges, the All-Pairs Non-decreasing Paths (APNP) problem asks to determine, for every pair of vertices u, v, the minimum last edge weight on a non-decreasing path from u to v (∞ if such a path does not exist). Typically the output is in tabular form, i.e. a matrix R with $R[i, j]$ corresponding to the minimum last edge weight of a non-decreasing path from u_i to u_j. Also conventionally the entries $R[i, i]$'s are set as $-\infty$ [22]. The matrix R is called the "APNP matrix" of G. Given a matrix A, the i-th row, the j-th column of A are denoted as $A[i, \cdot]$, $A[\cdot, j]$ resp. Also for two matrices A, B of the same size, the entry-wise minimum of them is denoted as $\min(A, B)$. Given a path p in G, if vertex k_1 appears earlier on p than k_2, then the portion of p from k_1 to k_2 is denoted as $p[k_1, k_2]$. Notations $p(k_1, k_2]$, $p[k_1, k_2)$, $p(k_1, k_2)$ represent $p[k_1, k_2]$ excluding k_1, k_2, both k_1 and k_2 resp. The following special matrix products are used in later algorithms.

▶ **Definition 2** (Various Products). Given two $n \times n$ matrices A and B over a totally ordered set, the dominance product $A \circledS B$ is defined as

$$(A \circledS B)[i,j] = |\{k \mid A[i,k] \leq B[k,j]\}|.$$

The $(\min, \leq)$-product $A \circledcirc B$ is defined as

$$(A \circledcirc B)[i,j] = \begin{cases} \min_k\{B[k,j] \mid A[i,k] \leq B[k,j]\} & \text{if } \exists k, A[i,k] \leq B[k,j], \\ \infty & \text{otherwise.} \end{cases}$$

Next we review a simple technique called *row-balancing* [6]. Basically by row-balancing, a matrix is decomposed into a sparse matrix and a dense one, where the finite entries of the dense matrix are uniformly re-distributed across the rows.

▶ **Definition 3** ([6] Row-Balancing). Let A be an $n \times p$ matrix with m finite elements. Depending on context, the other elements will either all be ∞ or all be $-\infty$. We assume the former below. The row-balancing of A, or $\mathbf{rb}(A)$, is a pair (A', A'') of $n \times p$ matrices, each with at most $k = \lceil m/n \rceil$ elements in each row. The row-balancing is obtained by the following procedure: First, sort all the finite elements in the i-th row of A in non-increasing or non-decreasing order depending on context , and divide this list into several parts $T_i^1, T_i^2, \ldots, T_i^{a_i}$ such that all parts except the last one contain k elements and the last part $(T_i^{a_i})$ contains at most k elements. Let A' be the submatrix of A containing the last parts:

$$A'[i,j] = \begin{cases} A[i,j] & \text{if } A[i,j] \in T_i^{a_i}, \\ \infty & \text{otherwise.} \end{cases}$$

Since the remaining parts have exact k elements, there can be at most $m/k \leq n$ of them. We assign each part to a distinct row in A'', i.e., we choose an arbitrary mapping $\rho : [n] \times [p/k] \to [n]$ such that $\rho(i,q) = i'$ if T_i^q is assigned row i'; it is undefined if T_i^q doesn't exist. Let A'' be defined as:

$$A''[i',j] = \begin{cases} A[i,j] & \text{if } \rho^{-1}(i') = (i,q) \text{ and } (i,j) \in T_i^q, \\ \infty & \text{otherwise.} \end{cases}$$

Thus, every finite $A[i,j]$ in A has a corresponding element in either A' or A'', which is also in the j-th column. The column-balancing of A, or $\mathbf{cb}(A)$, is similarly defined as (A'^T, A''^T), where $(A', A'') = \mathbf{rb}(A^T)$.

This simple technique is very useful for computing $A \circledS B$, $A \circledcirc B$, $(\max, \min)(A, B)$, and several new hybrid products defined in [6]. Below is a theorem from [6], which shows how to compute $A \circledS B$ using $\mathbf{cb}(A)$.

▶ **Theorem 4** ([6] Sparse Dominance Product). *Let A and B be two $n \times n$ matrices where the number of non-(∞) values in A is m_1 and the number of non-$(-\infty)$ values in B is m_2. Then $A \circledS B$ can be computed in time $O(m_1 m_2/n + n^\omega)$.*

There is a symmetric problem of computing non-decreasing paths with maximum first edge weights. Note that the maximum first edge weight on a non-decreasing path from i to i is defined as ∞, and for $i \neq j$, if no non-decreasing path exists, the maximum first edge weight is defined as $-\infty$. The time complexity of the single source version is given below.

▶ **Theorem 5** ([22] Maximum First Edge Weight). *Given a digraph G with n vertices, m edges, and a vertex s, there exists an algorithm which in $O(m \log n)$ time outputs the maximum first edge weight on a non-decreasing path from s to v for every $v \in V$.*

We need a slightly modified auxiliary data structure from [22] to efficiently compute a non-decreasing path with the minimum last edge weight starting from a subset of the out-edges of the source vertex. A paraphrased proof is given below.

▶ **Theorem 6.** *Given a digraph $G = (V, E, w)$ with n vertices, m edges, and a vertex s, there exists an algorithm which in $O(m \log n)$ time constructs a balanced binary search tree $T(t)$ for every $t \in V$. With $T(t)$, given a weight value w', the algorithm can determine in $O(\log n)$ time the minimum last edge weight of a non-decreasing path from s to t, starting from any out-edge e of s with $w(e) \geq w'$.*

Proof. For every vertex v, add an attribute $d[v]$ and a list $L(v)$, where $d[s] = -\infty$, $d[v] = \infty$ if $v \neq s$, and $L(v) = \varnothing$ initially. Next start a search resembling DFS from s, where differently the edges of s are explored in *reverse* sorted order, and the search only follows non-decreasing paths. Also for every edge (u, v), once explored, we first run $d[v] \leftarrow \min\{d[v], w(u, v)\}$ and then remove (u, v) from G. Lastly, vertices can be repeatedly visited. Consider the time when this recursive search backtracks to the initial search at s. The algorithm will explore the next *unexplored* out-edge of s in the reverse sorted order. We can thus partition the search into different phases, each of which corresponds to the search starting from an unexplored out-edge of s until the algorithm backtracks to s. Consider the end of a specific phase corresponding to an out-edge e of s. For every v of which $d[v]$ becomes *strictly* smaller in this phase, append $(w(e), d[v])$ to $L(v)$. At the end of the whole search, for every v, scan $L(v)$, where for consecutive elements of equal $w(e)$, only the last one is retained. Next transform $L(v)$ into a balanced binary search tree $T(v)$ keyed by $w(e)$.

Given a weight value w', the value required is the $d[v]$ attribute of the predecessor found. ◀

We last review a standard technique called *bridging set*, which is repeatedly used in the literature [6, 22, 25, 17]. The following lemma is one from [22]. The set of size $\frac{n \log n}{L}$ constructed is an L-bridging set.

▶ **Lemma 7** ([22, 25])**.** *Given a collection of N subsets of $\{1, \ldots, n\}$, each of size L, one can find in deterministic $O(NL)$ time a set of $\frac{n \log n}{L}$ elements of $\{1, \ldots, n\}$ hitting every one of the subsets.*

Model of Computation: We use the standard addition-comparison computational model. The only operations performed on real numbers are comparisons in this paper.

3 Warm Up: A Simple Case

As a warm up for our main algorithms, we consider a simple algorithm, which is efficient if the out-edges are uniformly distributed across the vertices. The main purpose is to give an idea of how the combinatorial properties of non-decreasing paths are utilized, and the difficulty of extending the algorithm to the general case.

Given a digraph $G = (V, E, w)$, let $L(E)$ be a sorted list of the edges in E. We evenly divide $L(E)$ into t parts such that each part has at most $\lceil n^2/t \rceil$ edges. Each part corresponds to a subgraph of G. Therefore there are t subgraphs. Denote them as G_r for $1 \leq r \leq t$. The edge weights in G_{r-1} is no greater than those in G_r.

Consider the adjacency matrix A_j of G_j. By saying a *simple* case, we mean the number of $(< \infty)$ elements of any row in any A_j is bounded by d, which has $n/t \leq d \leq n$. This assumption is *only* used in Theorem 15 later. We use the notation $G_{\leq r}$ to represent the

subgraph induced by the ($< \infty$) entries of A_j for $1 \leq j \leq r$. The notation $G_{<r}$ can be inferred similarly. The proposed algorithm consists of t iterations. In the r-th iteration, the APNP matrix of $G_{<r}$ is extended to the APNP matrix of $G_{\leq r}$. Consider the APNP matrix R of $G_{<r}$. Except a technical issue of edges of equal weights straddling across different A_r's (which is handled later in this section), one can verify that by running $R \leftarrow \min(R, R \otimes A_r)$ for $(n-1)$ steps, the matrix R will be the APNP matrix of $G_{\leq r}$. Intuitively by n' steps of $R \leftarrow \min(R, R \otimes A_r)$, the algorithm considers all the non-decreasing paths containing at most n' edges of G_r. The combinatorial properties of non-decreasing paths bring the following observations in the extension from $G_{<r}$ to $G_{\leq r}$.

The APNP matrices of $G_{<r}$, $G_{\leq r}$ are denoted as R, R' resp. If $R[i,j] < \infty$, then the non-decreasing path in G from i to j with the minimum last edge weight is already computed, for existing non-decreasing paths with minimum last edge weights cannot be improved by introducing edges of no smaller weights.

▶ **Observation 8** (An Entry Only Computed Once)**.** If $R[i,j] < \infty$, then $R'[i,j] = R[i,j]$.

The above observation indicates that the exact value of an entry of the APNP matrix is computed in at most one extension among all the extensions.

By allowing non-decreasing paths with more edges of G_r, minimum last edge weights of non-decreasing paths never become larger. Therefore we have the following observation.

▶ **Observation 9** (Non-increasing of Entries)**.** In the process of the $(n-1)$ steps of $R \leftarrow \min(R, R \otimes A_r)$, the entry $R[i,j]$ is non-increasing.

Generally, given a non-decreasing path p from s to t with the minimum last edge weight, we can only claim subpaths starting from s (*prefixes*) can be replaced with non-decreasing paths with minimum last edge weights. However, as shown below, except the technicality on equal weights handled later, the claim also holds for certain other subpaths.

▶ **Observation 10** (Two Parts of A Path)**.** Consider a non-decreasing path p in $G_{\leq r}$ with the minimum last edge weight. W.l.o.g. the path p can be split into two portions p_1, p_2 lying within $G_{<r}$, G_r resp. Any prefix of p_2 can be replaced with a non-decreasing path in G_r with the minimum last edge weight.

We use the following definition to capture the entries of interest in R'.

▶ **Definition 11.** An entry $R'[i,j]$ is *new* w.r.t. R if $R'[i,j] < \infty$ but $R[i,j] = \infty$. Otherwise, it is *old*.

Use n_r to denote the number of new entries of the APNP matrix of $G_{\leq r}$ w.r.t. the APNP matrix of $G_{<r}$. The following observation is then obvious.

▶ **Observation 12** (Bounded Number of New Entries)**.** $\sum_{1 \leq r \leq t} n_r \leq n^2$.

We also need a different view of computing $C = \min(A, A \otimes B)$, where A, B are $n \times n$ matrices. The matrix C is the entry-wise minimum of A and the n matrices $A[\cdot, k] \otimes B[k, \cdot]$ for $1 \leq k \leq n$. An algorithm for computing C using this view is given in Table 1. Now we present the intuition of the algorithm for the simple case.

The Intuition: The matrices R, R' are defined as before. Among all the non-decreasing paths from i to j in $G_{\leq r}$ with the minimum last edge weight, consider one path p with the least number of edges in G_r. As in Observation 10 (Two Parts of A Path), w.l.o.g. p is a concatenation of subpaths $p[i,k_1]$, $p[k_1,j]$. Subpaths $p[i,k_1]$, $p[k_1,j]$ are within $G_{<r}$, G_r resp. The entry $R'[i,k_1]$ is old, whereas the entries $R'[i,k']$'s are new for $k' \in p(k_1,j)$

Algorithm 1 An algorithm for computing $C = \min(A, A \otimes B)$.

(S1) Initialize C as A. Construct a sorted list $L(B[k,\cdot])$ of the $(< \infty)$ elements of $B[k,\cdot]$ in non-increasing order, for every k.
(S2) For every $A[\cdot,k]$, and for every $A[i,k]$ within:
  (S21) Scan $L(B[k,\cdot])$ from head to tail until the first element of $L(B[k,\cdot])$ which is less than $A[i,k]$.
  (S22) $C[i,j] \leftarrow \min\{C[i,j], B[k,j]\}$ for every $B[k,j]$ scanned, excluding the first element of $L(B[k,\cdot])$ which is less than $A[i,k]$.

due to p having the least number of edges in G_r. Therefore during the $(n-1)$ steps of $R \leftarrow \min(R, R \otimes A_r)$, the old entries effectively change R *only* in the first step. The changes of R in the later $(n-2)$ steps are contributed only by the *new* entries. In the view of the algorithm in Table 1, the entry (i,k') of R is only compared with the k'-th row of A_r. As the number of $(<\infty)$ elements of the k'-th row of A_r is assumed to be bounded, the cost brought by all the new entries is bounded due to Observation 9 (Non-increasing of Entries) and Observation 12 (Bounded Number of New Entries).

Besides the algorithm in Table 1, we also need an "ordinary" way to compute $R \otimes A_r$ as shown below.

▶ **Lemma 13** (The First Edge). *Given the APNP matrix R of $G_{<r}$, the product $R \otimes A_r$ can be computed in $\tilde{O}(n^\omega + n_r \cdot \frac{n}{t})$ time.*

Proof. According to Observation 8 (An Entry Computed Only Once), only the new entries of $R \otimes A_r$ w.r.t. R are of concern. To get these, we first determine the set S of (i,j)'s with $(R \otimes A_r)[i,j] < \infty$ but $R[i,j] = \infty$. Again to determine whether $(R \otimes A_r)[i,j] < \infty$, we only need to compute $R \circledcirc A_r$, where a slight difference is that here only the $(<\infty)$ entries of A_r are considered. As the $(<\infty)$ entries of R are no greater than those of A_r, the product $R \circledcirc A_r$ is a matrix product of two Boolean matrices corresponding to the $(<\infty)$ entries of R, A_r resp., which is computable in $O(n^\omega)$ time.

To determine the exact value of $(R \otimes A_r)[i,j] < \infty$, we use an idea similar to one in [6]. Let $(A'_r, A''_r) = \mathbf{cb}(A_r)$. Compute $R \circledcirc A'_r$ and $R \circledcirc A''_r$ in $O(n^\omega)$ time. The value of $(R \otimes A_r)[i,j]$ is $R[i,\cdot] \otimes A_r[\cdot,j]$. In $\mathbf{cb}(A_r)$, the column $A_r[\cdot,j]$ is divided into a_j parts $T_j^1, \ldots, T_j^{a_j}$. To determine the right q' where $(R \otimes A_r)[i,j] \in T_j^{q'}$, check $(R \circledcirc A'_r)[i,j] > 0$, and if it does not hold, search for the largest q with $(R \circledcirc A''_r)[i, \rho(j,q)] > 0$. Note that T_j^q's with $q < a_j$ are assigned to columns $\rho(j,q)$'s in matrix A''_r. Once q' is known, the exact value of $(R \otimes A_r)[i,j]$ is returned by an exhaustive enumeration within $T_j^{q'}$. The total time for the new entries of $R \otimes A_r$ is $\tilde{O}(n^\omega + n_r \cdot \frac{n}{t})$, as each entry of $R \circledcirc A''_r$ is checked no more than once, and $T_j^{q'}$'s are of size $O(n/t)$. ◀

A Technicality on Equal Weights: To let the edges of a non-decreasing path first come from $G_{<r}$, then from G_r, we need to handle the case in which the edges of equal weight straddle across different A_r's. We split out these special edges and merge them into single matrices. Consider one such a matrix A'. The APNP matrix R' of the corresponding subgraph of A' can be computed by a transitive closure algorithm in $O(n^\omega)$ time. Suppose the APNP matrix before processing A' is R. Next we run $R \leftarrow \min(R, R \otimes R')$, which is similar to Lemma 13 (The First Edge), but now the finite entries of R' are all equal to the single value. Therefore the cost is $O(n^\omega)$. As there are only $O(t)$ such A''s, the total cost for the technicality is only $O(t \cdot n^\omega)$.

Algorithm 2 The algorithm for the simple case.

(S1) An APNP matrix R is initialized as $R[i,i] = -\infty$, and $R[i,j] = \infty$ for $i \neq j$.

(S2) $\forall r, 1 \leq r \leq t$, the APNP matrix R of $G_{<r}$ is extended to the APNP matrix of $G_{\leq r}$. Construct a sorted list $L(A_r[k,\cdot])$ of the $(< \infty)$ elements of $A_r[k,\cdot]$ in non-increasing order, for $1 \leq k \leq n$.

(S21) Run $R \leftarrow \min(R, R \circledcirc A_r)$ as in Lemma 13 (The First Edge). Let S be the set of the entries of R which get *strictly* smaller. Any entry *first* added to S, say $R[i,k]$, is associated with a pointer pointing to the head of $L(A_r[k,\cdot])$.

(S22) Run the following for $(n-2)$ steps or until $S = \varnothing$.

i. For every $R[i,k] \in S$:

A. Starting from the position of $L(A_r[k,\cdot])$ pointed to by the pointer associated with $R[i,k]$, move the pointer element by element until the first element of $L(A_r[k,\cdot])$ which is less than $R[i,k]$.

B. $R[i,j] \leftarrow \min\{R[i,j], A_r[k,j]\}$, for every element $A_r[k,j]$ of $L(A_r[k,\cdot])$ scanned, excluding the first element of $L(A_r[k,\cdot])$ which is less than $R[i,k]$.

ii. Re-initialize S as the set of the entries of R which get *strictly* smaller in this step.

iii. Any entry *first* added to S, say $R[i,k]$, is associated with a pointer pointing to the head of $L(A_r[k,\cdot])$.

The formal algorithm for the simple case is given in Table 2. The analysis for it is given in Theorem 15. Also by Observation 8 (An Entry Only Computed Once), the set S in Table 2 has the following property which is useful in the proof.

▶ **Observation 14.** The set S in Table 2 contains only the new entries of the APNP matrix of $G_{\leq r}$ w.r.t. the APNP matrix of $G_{<r}$.

▶ **Theorem 15.** *If the number of $(< \infty)$ elements in any row of A_r is bounded by d, then the steps S21 and S22 in Table 2 can find the APNP matrix of $G_{\leq r}$ in $\tilde{O}(n^\omega + n_r \cdot (d + \frac{n}{t}))$ time in every iteration. So the total time is $\tilde{O}(t \cdot n^\omega + n^2 \cdot (d + \frac{n}{t}))$.*

Proof. To show the correctness, we only need to prove that the step S22 in Table 2 is equivalent to $(n-2)$ steps of the algorithm in Table 1. This is almost obvious, as the entries not getting *strictly* smaller during the previous step of $R \leftarrow \min(R, R \circledcirc A_r)$ do not contribute anything in the current step of $R \leftarrow \min(R, R \circledcirc A_r)$. Also if S in the step S22 of Table 2 is empty, we can stop earlier than $(n-2)$ steps, as later steps of $R \leftarrow \min(R, R \circledcirc A_r)$ give the same R.

Note that the cost of the step S22 in Table 2 is charged to the scanning induced by the entries in S. By Observation 9 (Non-increasing of Entries) and Observation 14, the time of the step S22 of Table 2 in r-th iteration is $\tilde{O}(n_r \cdot d)$. Also by Lemma 13 (The First Edge), the step S21 of Table 2 takes $\tilde{O}(n^\omega + n_r \cdot \frac{n}{t})$ time, thus prove the theorem. ◀

So if the out-edges are uniformly distributed across all vertices, i.e., the number of $(< \infty)$ elements of any row in any A_r is always bounded by $O(n/t)$ for any t, then we can get a $\tilde{O}(n^{(3+\omega)/2})$ time APNP algorithm by setting $t = n^{(3-\omega)/2}$.

Generally a row of an A_r could have as many as $\Omega(n)$ $(< \infty)$ elements. Rows with a large number of $(< \infty)$ elements correspond to vertices of high out-degree. A new approach for them is given in the next section.

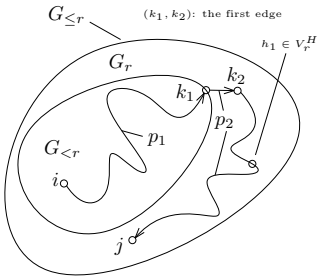


Figure 1 An illustration of the extension of the APNP matrix from $G_{<r}$ to $G_{\le r}$ in the general case.

4 Improved Algorithms for APNP

We move to the general case, in which the number of $(< \infty)$ elements in a row might not be bounded by $O(n/t)$. For ease of analysis, the edge set is divided into n^t subsets, each with at most $\lceil n^2/n^t \rceil = \lceil n^{2-t} \rceil$ edges. The vertices of G_r are classified as *high* out-degree with out-degree $> n^{1-t+s}$, or *low* out-degree otherwise, where $s > 0$ is a parameter to be chosen. Denote the sets of high out-degree, low out-degree vertices of G_r as V_r^H, V_r^L resp. Note that $|V_r^H| = O(n^{1-s})$.

The Intuition: As illustrated in Figure 1, among all the non-decreasing paths in $G_{\le r}$ from i to j with the minimum last edge weight, consider one path p with the least number of edges in G_r. By running the step S21 of Table 2, we construct the first edge (k_1, k_2). Thanks to p having the least number of edges in G_r, entries (i, k') of the APNP matrix of $G_{\le r}$ for $k' \in p[k_2, j)$ are new w.r.t. the APNP matrix of $G_{<r}$. The hard case is when there exists a vertex from V_r^H on $p[k_2, j)$. Consider the first such vertex h_1 from V_r^H. The important observation is that the edges on $p[k_2, h_1]$ are the out-edges of *low* out-degree vertices. Therefore, if we run the step S22 of Table 2, but differently only on the *low* out-degree vertices, then the portion $p[i, h_1]$ is successfully constructed. The last portion $p[h_1, j]$ can be constructed by Theorem 6, as $p[h_1, j]$ represents a non-decreasing path starting with an edge of weight no smaller than the last edge weight of $p[i, h_1]$. However, to get an efficient algorithm, we could not afford to construct $p[h_1, j]$ if it did not exist. To determine the existence of $p[h_1, j]$, we replace $p[h_1, j]$ with a non-decreasing path in G_r from h_1 to j with the *maximum* first edge weight. Then the existence of $p[h_1, j]$ is reduced to whether $p[i, h_1]$ can be concatenated with this replacement of $p[h_1, j]$.

The algorithm for the general case is described in Table 3. For the matrix H_1^N in Table 3, we have the following properties which are crucial for the correctness and an efficient algorithm.

▶ **Observation 16.** Consider one path p with the least number of edges in G_r among all the non-decreasing paths in $G_{\le r}$ from i to j with the minimum last edge weight. Let $p[k_1, j]$ be the portion of p in G_r. If $p[k_1, j] \neq \varnothing$, and there exists a vertex from V_r^H on $p(k_1, j)$, then the last edge weight of $p[i, h_1]$ is stored in $H_1^N[i, h_1]$, where h_1 is the first vertex in V_r^H on $p(k_1, j)$. For the matrix H_1^N, the number of $(< \infty)$ entries of H_1^N is no greater than n_r.

For the matrix X, the number of $X[i, j] = 1$ with $R[i, j] = \infty$ is no more than n_r. Such (i, j)'s of X are associated with non-decreasing paths never appearing in $G_{<r}$. Hence these (i, j)'s in the APNP matrix of $G_{\le r}$ are new w.r.t. R. By Theorem 6, the running time of the step S24 therefore is as follows.

Algorithm 3 The algorithm for the general case.

(S1) An APNP matrix R is initialized as $R[i,i] = -\infty$, and $R[i,j] = \infty$ for $i \neq j$.

(S2) $\forall r, 1 \leq r \leq n^t$, the APNP matrix R of $G_{<r}$ is extended to the APNP matrix of $G_{\leq r}$.

- **(S21)** Split out a sub-matrix A_r^L of A_r, which contains only the rows with the number of ($< \infty$) elements no greater than n^{1-t+s}. Initialize a matrix R' as R.
  - **i.** Run the steps S21, S22 in Table 2 with inputs R', A_r and A_r^L. The step S22 of Table 2 only works on A_r^L instead of A_r.
- **(S22)** With Theorem 5 (Maximum First Edge Weight), compute an $n^{1-s} \times n$ matrix H_2 with $H_2[i,j]$ representing the maximum first edge weight on a non-decreasing path in G_r from $i \in V_r^H$ to j.
- **(S23)** Group the entries of R' from V to V_r^H that are *new* w.r.t. R as an $n \times n^{1-s}$ matrix H_1^N. Construct a Boolean matrix X with $X[i,j] = 1$ if $(H_1^N \circledS H_2)[i,j] > 0$.
- **(S24)** Initialize a matrix R'' as R. For every $h_1 \in V_r^H$, use Theorem 6 to build the auxiliary data structure for h_1 in G_r, i.e., a set of $T(h_1, j)$'s for all $j \in V$.
  For every $X[i,j] = 1$ with $R[i,j] = \infty$:
  - **i.** For every $h_1 \in V_r^H$:
    - **A.** Query $T(h_1, j)$ with $w' = H_1^N[i, h_1]$. Get the minimum last edge weight w'' of a non-decreasing path from h_1 to j in G_r, starting from any out-edge e of h_1 with $w(e) \geq w'$.
    - **B.** $R''[i,j] \leftarrow \min(R''[i,j], w'')$.
- **(S25)** $R \leftarrow \min(R', R'')$.

▶ **Observation 17.** The step S24 of Table 3 has the running time of $\tilde{O}(n_r \cdot n^{1-s} + n^{1-s} \cdot n^{2-t})$.

▶ **Theorem 18.** *Given a real edge-weighted digraph on n vertices, the APNP matrix can be computed in $\tilde{O}(n^{2+\omega/3}) = O(n^{2.791})$ time.*

Proof. We show the APNP matrix R is correctly extended from $G_{<r}$ to $G_{\leq r}$. Among all the non-decreasing paths in $G_{\leq r}$ from i to j with the minimum last edge weight, consider one path p with the least number of edges in G_r. We show p can be constructed by the algorithm. As in Observation 10 (Two Parts of A Path), the path p consists of two subpaths $p[i, k_1]$, $p[k_1, j]$, lying within $G_{<r}$, G_r resp. An illustration is given in Figure 1. If $p[k_1, j] = \varnothing$, then $R[i,j]$ is an old entry. Therefore p is already constructed. Consider the case in which $p[k_1, j] \neq \varnothing$. Let (k_1, k_2) be the first edge on $p[k_1, j]$. If there is no vertex in V_r^H on $p[k_2, j)$, the circumstance is similar to the simple case of Section 3. The step S21 successfully constructs p under this circumstance. The only case left is when there exists a vertex in V_r^H on $p[k_2, j)$. Consider the first such vertex h_1. Due to Observation 16, the subpath $p[i, h_1]$ is successfully constructed by the step S21. The last portion $p[h_1, j]$ is constructed by the step S24, as $X[i,j] = 1$ and $R[i,j] = \infty$ in this case.

Next we proceed to the analysis of the time complexity. The step S1 costs $O(n^2)$. According to Theorem 15, the step S21 needs the running time of $\tilde{O}(n^\omega + n_r \cdot n^{1-t+s})$ to compute R'. By Theorem 5 (Maximum First Edge Weight), the step S22 takes $\tilde{O}(n^{1-s} \cdot n^{2-t})$ time, as $|V_r^H| = O(n^{1-s})$ and G_r has $O(n^{2-t})$ edges. The step S23 involves the computation of $H_1^N \circledS H_2$, for which Theorem 4 (Sparse Dominance Product) is used. Due to Observation 16, the cost for it is $\tilde{O}(n^\omega + n_r \cdot n^{1-s})$. Note that the number of ($> -\infty$) entries of H_2 is no greater than n^{2-s}. Also H_1^N, H_2 are *expanded* to $n \times n$ matrices in Theorem 4. For easy understanding, one can do $\mathbf{rb}(H_2)$ rather than $\mathbf{cb}(H_1^N)$ in the proof of Theorem 4, which can be found in [6]. The step S24 takes $\tilde{O}(n_r \cdot n^{1-s} + n^{1-s} \cdot n^{2-t})$ time according to

Observation 17. The step S25 costs $O(n^2)$. Summing up all these costs, we have the total running time within $\tilde{O}(\cdot)$ as follows.

$$\begin{aligned}&\sum_{1\le r\le n^t}(n^\omega + n_r\cdot n^{1-t+s}) + n^{3-s}\\ &+\sum_{1\le r\le n^t}(n^\omega + n_r\cdot n^{1-s})\\ &+\sum_{1\le r\le n^t}(n_r\cdot n^{1-s} + n^{1-s}\cdot n^{2-t})\\ =&n^{t+\omega} + n^{3-t+s} + n^{3-s},\end{aligned}$$

where by choosing $t = 2s$, and $s = 1 - \frac{\omega}{3}$, we have the final result $n^{2+\omega/3}$. ◀

4.1 A Slight Improvement via Rectangular Matrix Multiplication

The algorithm in Table 3 builds auxiliary data structures of Theorem 6 for all vertices in V_r^H. We can lessen this cost a little bit at the price of considering more vertices like h_1 in Figure 1.

The Intuition: As illustrated in Figure 1, formerly we consider the first vertex h_1 in V_r^H on $p[k_2, j)$. Now we look at more vertices from V_r^H on $p[k_2, j)$. Consider the second such vertex h_2. Similarly the edges on $p(h_1, h_2]$ are the out-edges of *low* out-degree vertices. Also thanks to p having the least number of edges in G_r, entries (i, k') of the APNP matrix of $G_{\le r}$ for $k' \in p(h_1, h_2)$ are new w.r.t. the APNP matrix of $G_{<r}$. Therefore, if we had constructed the first edge of $p[h_1, h_2]$, the cost for constructing later edges on $p[h_1, h_2]$ could be charged to the step S22 in Table 2. The idea for the improvement is to look at the first n^q vertices from V_r^H on $p[k_2, j)$. Name them as $h_1, \ldots, h_{n^q}$. If $p[k_2, j)$ contains no more than n^q vertices from V_r^H, the path p is then fully constructed. If $p[k_2, j)$ contains more than n^q vertices from V_r^H, we could sample only $O(n^{1-s-q}\log n)$ vertices from V_r^H, such that w.h.p. at least one vertex in $\{h_1, \ldots, h_{n^q}\}$ is hit. Say one such vertex is h. Note that $p[i, h]$ is already constructed. We then use the same auxiliary data structures as in Table 3 to construct $p[h, j]$. Also we use the same algorithm as in Table 3 to first determine the existence of $p[h, j]$. The difference is that the construction of $p[h, j]$ only involves the computation on $O(n^{1-s-q}\log n)$ vertices, instead of $O(n^{1-s})$ vertices as before. The improvement comes from this difference.

The algorithm is given in Table 4, which needs to compute $H_1^N \circledcirc A_r^H$ for matrices H_1^N, A_r^H defined within. The time complexity of the computation is given as follows, for which the idea is similar to one in [6]. Note that Theorem 4 (Sparse Dominance Product) can be extended to handling two matrices of sizes $n \times n^{1-s}$ and $n^{1-s} \times n$. The time complexity similar to the one in Theorem 4 is $O(m_1 m_2/n^{1-s} + n^{\omega(1,1-s,1)})$.

▶ **Lemma 19.** *The computation for $H_1^N \circledcirc A_r^H$ takes $\tilde{O}(n^{\omega(1,1-s,1)+q} + n_r\cdot n^{1-t-q+s})$ time.*

Proof. Let $L(A_r^H)$ be the sorted list of $(<\infty)$ entries of A_r^H. We divide $L(A_r^H)$ into n^q parts, so each part contains at most $\lceil n^{2-t}/n^q\rceil$ entries. We have n^q matrices $A_{r,1}^H, \ldots, A_{r,n^q}^H$ corresponding to each part. The entries of H_1^N with values no greater than $\max\{e \mid e \in A_{r,n^q}^H\}$ are partitioned into (n^q+1) matrices $H_{1,0}^N, H_{1,1}^N, \ldots, H_{1,n^q}^N$. The matrix $H_{1,p}^N$ with $1 \le p \le n^q$ consists of the entries $H_1^N[i, j]$'s satisfying $\min\{e \mid e \in A_{r,p}^H\} < H_1^N[i, j] \le \max\{e \mid e \in A_{r,p}^H\}$. The matrix $H_{1,0}^N$ consists of the entries $H_1^N[i, j]$'s satisfying $H_1^N[i, j] \le \min\{e \mid e \in A_{r,1}^H\}$.

We compute $H_1^N \circledcirc A_{r,p}^H$ for $1 \le p \le n^q$. Note that the following holds.

$$H_1^N \circledcirc A_{r,p}^H = (H_{1,0}^N + \cdots + H_{1,p}^N) \circledcirc A_{r,p}^H,$$

Algorithm 4 A slightly faster algorithm via rectangular matrix multiplication.

(S1) An APNP matrix R is initialized as $R[i,i] = -\infty$, and $R[i,j] = \infty$ for $i \neq j$.

(S2) $\forall r, 1 \leq r \leq n^t$, the APNP matrix R of $G_{<r}$ is extended to the APNP matrix of $G_{\leq r}$. Split A_r into two sub-matrices A_r^H, A_r^L, representing out-edges of high, low out-degree vertices resp.

- **(S21)** Initialize a matrix R' as R.
  - **i.** Run the steps S21, S22 in Table 2 with inputs R', A_r and A_r^L. The step S22 of Table 2 only works on A_r^L instead of A_r.
- **(S22)** Run the following for n^q steps:
  - **i.** Group the entries of R' from V to V_r^H that are *new* w.r.t. R as an $n \times n^{1-s}$ matrix H_1^N.
  - **ii.** $R' \leftarrow \min(R', H_1^N \circledcirc A_r^H)$. Determine the set S of the entries of R' which get *strictly* smaller in this step.
  - **iii.** Run the step S22 of Table 2 with inputs R', S and A_r^L.
- **(S23)** Sample uniformly at random $O(n^{1-s-q} \cdot \log n)$ vertices from V_r^H. Run the steps S22–S25 in Table 3, but only on the sampled vertices, rather than V_r^H.

where $(H_{1,0}^N + \cdots + H_{1,p-1}^N) \circledcirc A_{r,p}^H$ is computable in $O(n^{\omega(1,1-s,1)})$ time, as the entries in $(H_{1,0}^N + \cdots + H_{1,p-1}^N)$ are no greater than $\min\{e \mid e \in A_{r,p}^H\}$. Use $|H_{1,p}^N|$ to represent the number of $(< \infty)$ entries in $H_{1,p}^N$. We use Theorem 4 (Sparse Dominance Product) to compute $H_{1,p}^N \circledcirc A_{r,p}^H$ in $\tilde{O}(n^{\omega(1,1-s,1)} + |H_{1,p}^N| \cdot n^{2-t-q}/n^{1-s})$ time, as $A_{r,p}^H$ only has n^{1-s} rows. The total time for computing $H_1^N \circledcirc A_{r,p}^H$ for $1 \leq p \leq n^q$ is $\tilde{O}(n^{\omega(1,1-s,1)} \cdot n^q + n_r \cdot n^{2-t-q}/n^{1-s})$, as $\sum_{1 \leq p \leq n^q} |H_{1,p}^N| \leq n_r$.

Let $((A_{1,p}^H)', (A_{1,p}^H)'') = \mathbf{cb}(A_{1,p}^H)$. The computation of $H_1^N \circledcirc (A_{r,p}^H)'$, $H_1^N \circledcirc (A_{r,p}^H)''$ for $1 \leq p \leq n^q$ also takes $\tilde{O}(n^{\omega(1,1-s,1)} \cdot n^q + n_r \cdot n^{2-t-q}/n^{1-s})$ time.

For an entry $(H_1^N \circledcirc A_r^H)[i,j]$ to be determined, we find the smallest p with $(H_1^N \circledcirc A_{r,p}^H)[i,j] > 0$. Then $(H_1^N \circledcirc A_r^H)[i,j] \in A_{r,p}^H$. The remaining steps are similar to Lemma 13 (The First Edge). The total time is $\tilde{O}(n^{\omega(1,1-s,1)} \cdot n^q + n_r \cdot n^{2-t-q}/n^{1-s} + n_r \cdot n^{2-t-q}/n)$, as $A_{r,p}^H$ has n columns and only the new entries of $H_1^N \circledcirc A_r^H$ are of interest, of which the number is at most n_r. ◀

The step S23 in Table 4 constructs an L-bridging set with $L = n^q$. We use Lemma 7 to compute it *deterministically* as follows. Consider p as in the intuition of Section 4.1. The N subsets in Lemma 7 are p's with exactly n^q vertices from V_r^H on $p[k_2, j)$'s, where $j \in V_r^H$. To get the subset related to p, we need to extract the vertices in V_r^H on $p[k_2, j)$. This is solvable in $O(n^q)$ time if we know the predecessors in V_r^H of the vertices on $p(k_2, j]$.

We notice that for a specific i, just before the Step 23 of Table 4, for the j's of which $R'[i,j]$'s are new w.r.t. R, the edges corresponding to these $R'[i,j]$ form a forest, i.e., we know the predecessors in V (rather than in V_r^H) of these j's. Given such a forest, a DFS will give us the required predecessors in V_r^H of these j's. The cost of DFS for different i's is $O(n_r)$, as there are no more than n_r new $R'[i,j]$'s w.r.t. R. Thus the following holds.

▶ **Observation 20.** An L-bridging set with $L = n^q$ in the step S23 of Table 4 can be constructed *deterministically* in $\tilde{O}(n^{1-s} \cdot n^q)$ time.

▶ **Theorem 21.** *The APNP matrix of a real edge-weighted digraph with n vertices is computable in deterministic $O(n^{2.78})$ time.*

Proof. The correctness proof is almost similar to the one in Theorem 18. There we consider the first vertex h_1 in V_r^H, whereas here we consider the vertex h in V_r^H hit by the bridging

set. According to Observation 20, the total time for the construction of the bridging sets is $\tilde{O}(n^{t+1-s+q}) = \tilde{O}(n^{2+t-2s})$ (as $q \leq 1-s$), which is absorbed by the other terms.

Now we turn to the analysis of time complexity. The step S21 is the same as before, taking $\tilde{O}(n^{\omega} + n_r \cdot n^{1-t+s})$ time. The second step of S22 takes $\tilde{O}(n^{\omega(1,1-s,1)+q} + n_r \cdot n^{1-t-q+s})$ time, as shown in Lemma 19. The third step of S22 in one iteration of the step S2 is charged to the step S21, resulting in $\tilde{O}(n_r \cdot n^{1-t+s})$ time. The other steps are similar to Theorem 18, for which the time is $\tilde{O}(n^{\omega} + n_r \cdot n^{1-s-q} + n^{1-s-q} \cdot n^{2-t})$. Summing up all of them, we have the total running time within $\tilde{O}(\cdot)$ as follows.

$$n^{t+\omega} + n^{3-t+s} + n^{3-s-q} + n^{t+\omega(1,1-s,1)+2q},$$

where $\omega(1, 1-s, 1) \leq 2 + (1-s)(\omega - 2)$ if rectangular matrix multiplications are reduced to square matrix multiplications. By setting $s = \frac{3-\omega}{\omega+1}$, $t = \frac{1}{2}(3 + s - \omega)$, and $q = t - 2s$, we have the final result $n^{\frac{1}{2}(3+\frac{3-\omega}{\omega+1}+\omega)} = O(n^{2.78})$. ◀

References

1. Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997.
2. Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of Baur-Strassen's theorem: Shortest cycles, diameter, and matchings. *J. ACM*, 62(4):28, 2015.
3. Artur Czumaj, Mirosław Kowaluk, and Andrzej Lingas. Faster algorithms for finding lowest common ancestors in directed acyclic graphs. *Theoretical Computer Science*, 380(1-2):37–46, 2007.
4. Artur Czumaj and Andrzej Lingas. Finding a heaviest triangle is not harder than matrix multiplication. In *Proceedings of the 18th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 986–994. SIAM, 2007.
5. Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.
6. Ran Duan and Seth Pettie. Fast algorithms for $(\max, \min)$-matrix multiplication and bottleneck shortest paths. In *Proceedings of the 20th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–391. SIAM, 2009.
7. Pavlos Eirinakis, Matthew Williamson, and K. Subramani. On the Shoshan-Zwick algorithm for the all-pairs shortest path problem. *J. Graph Algorithms Appl.*, 21(2):177–181, 2017.
8. Michael L. Fredman and Robert E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.
9. Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134(2):103–139, 1997.
10. Zvi Galil and Oded Margalit. All pairs shortest paths for graphs with small integer length edges. *Journal of Computer and System Sciences*, 54(2):243–254, 1997.
11. François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proceedings of the 29th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, 2018.
12. François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, pages 296–303. ACM, 2014.
13. George J. Minty. A variant on the shortest-route problem. *Operations Research*, 6(6):882–883, 1958.

14 Liam Roditty and Virginia Vassilevska Williams. Minimum weight cycles and triangles: Equivalences and algorithms. In *Proceedings of the 52nd annual IEEE symposium on Foundations of Computer Science*, pages 180–189. IEEE, 2011.

15 Piotr Sankowski and Karol Węgrzycki. Improved distance queries and cycle counting by Frobenius normal form. In *Proceedings of the 34th Symposium on Theoretical Aspects of Computer Science*, 2017.

16 R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995.

17 Asaf Shapira, Raphael Yuster, and Uri Zwick. All-pairs bottleneck paths in vertex weighted graphs. *Algorithmica*, 59(4):621–633, 2011.

18 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 40th annual IEEE symposium on Foundations of Computer Science*, pages 605–614. IEEE, 1999.

19 Virginia Vassilevska and Ryan Williams. Finding a maximum weight triangle in $n^{3-\delta}$ time, with applications. In *Proceedings of the 38th annual ACM Symposium on Theory of Computing*, pages 225–231. ACM, 2006.

20 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. Finding the smallest H-subgraph in real weighted graphs and related problems. In *Proceedings of the 33rd International Conference on Automata, Languages and Programming*, pages 262–273. Springer-Verlag, 2006.

21 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All pairs bottleneck paths and max-min matrix products in truly subcubic time. *Theory of Computing*, 5(9):173–189, 2009.

22 Virginia Vassilevska Williams. Nondecreasing paths in a weighted graph or: How to optimally read a train schedule. *ACM Trans. Algorithms*, 6(4):70:1–70:24, 2010.

23 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith–Winograd. In *Proceedings of the 44th annual ACM Symposium on Theory of Computing*, pages 887–898. ACM, 2012.

24 Raphael Yuster. A shortest cycle for each vertex of a graph. *Information Processing Letters*, 111(21):1057–1061, 2011.

25 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.

Edit Distance between Unrooted Trees in Cubic Time

Bartłomiej Dudek
Institute of Computer Science, University of Wrocław, Poland
bartlomiej.dudek@cs.uni.wroc.pl

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Abstract

Edit distance between trees is a natural generalization of the classical edit distance between strings, in which the allowed elementary operations are contraction, uncontraction and relabeling of an edge. Demaine et al. [ACM Trans. on Algorithms, 6(1), 2009] showed how to compute the edit distance between rooted trees on n nodes in $O(n^3)$ time. However, generalizing their method to unrooted trees seems quite problematic, and the most efficient known solution remains to be the previous $O(n^3 \log n)$ time algorithm by Klein [ESA 1998]. Given the lack of progress on improving this complexity, it might appear that unrooted trees are simply more difficult than rooted trees. We show that this is, in fact, not the case, and edit distance between unrooted trees on n nodes can be computed in $O(n^3)$ time. A significantly faster solution is unlikely to exist, as Bringmann et al. [SODA 2018] proved that the complexity of computing the edit distance between rooted trees cannot be decreased to $O(n^{3-\varepsilon})$ unless some popular conjecture fails, and the lower bound easily extends to unrooted trees. We also show that for two unrooted trees of size m and n, where $m \leq n$, our algorithm can be modified to run in $O(nm^2(1 + \log \frac{n}{m}))$. This, again, matches the complexity achieved by Demaine et al. for rooted trees, who also showed that this is optimal if we restrict ourselves to the so-called decomposition algorithms.

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

Keywords and phrases tree edit distance, dynamic programming, heavy light decomposition

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.45

Related Version A full version of the paper is available at [14], `https://arxiv.org/abs/1804.10186`.

1 Introduction

Computing the edit distance between two strings [30] is the most well-known example of dynamic programming. Thanks to the new fine-grained complexity paradigm, we know that this simple approach is essentially the best possible [1, 5], so the problem appears to be solved from the theoretical perspective. However, in many real-life applications we would like to operate on more complicated structures than strings. As a prime example, while primary structure of RNA can be seen as a string, computational biology is often interested in comparing also secondary structures. Second structure of RNA can be modeled as an ordered tree [17, 26], so we would like to generalize computing the edit distance between strings to computing the edit distance between ordered trees.

Tai [29] defined the edit distance between two ordered trees as the minimum total cost of a sequence of elementary operations that transform one tree into the other. For unrooted

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 45; pp. 45:1–45:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

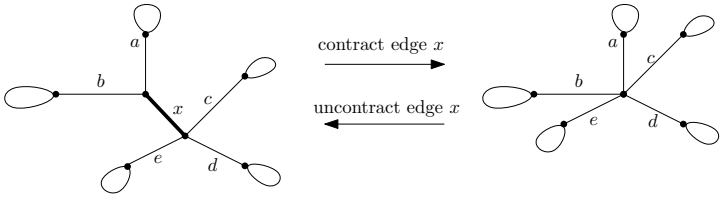


Figure 1 Contraction and uncontraction of the edge with label x costs $c_{del}(x) = c_{ins}(x)$.

trees, which are the focus of this paper, the trees are edge-labeled, and we have three elementary operations: contraction, uncontraction and relabeling of an edge. We think that the trees are embedded in the plane, i.e., there is a cyclic order on the neighbors of every node that is preserved by the contraction/uncontraction. See Figure 1. The cost of an operation depends on the label(s) of the edge(s): $c_{del}(\tau)$, $c_{ins}(\tau)$, $c_{match}(\tau_1, \tau_2)$, respectively. We assume that every operation has the same cost as its reverse counterpart: $c_{del}(\tau) = c_{ins}(\tau)$, $c_{match}(\tau_1, \tau_2) = c_{match}(\tau_2, \tau_1)$, and each edge participates in at most one elementary operation.

Computing the edit distance between trees is used as a measure of similarity in multiple contexts. The most obvious, given that some biological structures resemble trees, is computational biology [26]. Others include comparing XML data [10, 11, 16], programming languages [18]. Others, less obvious, include computer vision [6, 20, 22, 25], character recognition [24], automatic grading [3], and answer extraction [31]. See also the survey by Bille [7].

Tai [29] introduced the edit distance between rooted node-labeled trees on n nodes and designed an $O(n^6)$ algorithm. Zhang and Shasha [27] improved the time complexity to $O(n^4)$ by designing a recursive formula, which reduces computing the edit distance between two trees to computing the edit distance between two smaller trees. Then, Klein [21] considered the more general problem of computing the edit distance between unrooted edge-labeled trees and further improved the complexity to $O(n^3 \log n)$ using essentially the same formula, but applying it more carefully to restrict the number of different trees that appear in the whole process. This high-level idea of using the recursive formula can be formalized using the notion of decomposition strategy algorithms as done by Dulucq and Touzet [15]. Finally, Demaine et al. [13] further improved the complexity for rooted node-labeled trees to $O(n^3)$. For trees of different sizes m and n, where $m \leq n$, their algorithm runs in $O(nm^2(1 + \log \frac{n}{m}))$ time. At a very high level, the gist of their improvement was to apply the heavy path decomposition to both trees, while in Klein's algorithm only one tree is decomposed. This requires some care, as switching from being guided by the heavy path decomposition of the first tree to the second tree cannot be done too often.

Although Demaine et al. [13] showed that their algorithm is optimal among all decomposition strategies, it is not clear that any algorithm must be based on such a strategy. Nevertheless, there has been no progress on beating the best known $O(n^3)$ time worst-case bound for exact tree edit distance. Pawlik and Augsten [23] presented an experimental comparison of the known algorithms. Aratsu et al. [4], Akutsu et al. [2], and Ivkin [19] designed approximation algorithms. Only very recently a convincing explanation for the lack of progress on improving this worst-case complexity has been found by Bringmann et al. [9], who showed that a significant improvement on the cubic time complexity for rooted node-labeled trees is rather unlikely: an $O(n^{3-\varepsilon}$ algorithm for computing the edit distance between rooted trees on n nodes implies an $O(n^{3-\varepsilon})$ algorithm for APSP (assuming alphabet of size $\Theta(n)$) and an $O(n^{k(1-\varepsilon)})$ algorithm for Max-Weight k-Clique (assuming alphabet of sufficiently large but constant size).

Thus, the complexity of computing the edit distance between rooted trees seems well-understood by now. However, in multiple important applications, the trees are, in fact, unrooted. For example, Sebastian et al. [25] use unrooted trees to recognize shapes (in a paper with over 700 citations). Unfortunately, while the almost 20 years old algorithm presented by Klein works for unrooted trees in $O(n^3 \log n)$ time, it is not clear how to translate Demaine et al.'s improvement to the unrooted case. In fact, even if one of the trees is a rooted full binary tree and the other is a simple caterpillar, their approach appears to use $O(n^4)$ time, and it is not clear how to modify it. Given the lack of further progress, it might seem that unrooted trees are simply more difficult than rooted trees.

Our contribution. We present a new algorithm for computing the edit distance between unrooted trees which runs in $O(n^3)$ time and $O(n^2)$ space. For the case of trees of possibly different sizes n and m where $m \leq n$, it runs in $O(nm^2(1 + \log \frac{n}{m}))$ time and $O(nm)$ space. This matches the complexity of Demaine et al.'s algorithm for the rooted case and improves Klein's algorithm for the unrooted case. By a simple reduction, unrooted trees are as difficult as rooted trees, so our algorithm is optimal among all decomposition algorithms [13], and significantly faster approach is unlikely to exist unless some popular conjecture fails [9].

Our starting point is dynamic programming using the recursive formula of Zhang and Shasha, similarly as done by Klein and Demaine et al., but instead of presenting the computation in a top-down order, we prefer to work bottom-up. This gives us more control and allows us to be more precise about the details of the implementation. In the simpler $O(n^3 \log \log n)$ version of the algorithm, we apply the heavy path decomposition to both trees. As long as the first tree is sufficiently big, we proceed similarly as Klein, that is, look at its heavy path decomposition. However, if the first tree is small (roughly speaking) we consider the heavy path decomposition of the second tree and design a new divide and conquer strategy that is applied on every heavy path separately.

In the full version of the paper [14] we further improve the complexity to $O(n^3)$. Instead of a global parameter we modify the divide and conquer strategy so that the larger the first tree is the sooner the strategy terminates and switches to another approach. A careful analysis of such a modification leads to $O(nm^2(1 + \log^2 \frac{n}{m})) = O(n^3)$ running time. Then, we shave one $\log \frac{n}{m}$ by making the divide and conquer sensitive to the sizes of the subtrees attached to the heavy path instead of its length, that is, making some nodes more important than the other, reminiscing the so-called telescoping trick [8, 12]. All the improvements applied together decrease the overall complexity to $O(nm^2(1 + \log \frac{n}{m}))$, thus matching the running time of the algorithm by Demaine et al. for rooted trees [13].

▶ **Theorem 1.** *Edit distance between unrooted trees of size n and m, where $m \leq n$, can be computed in $O(nm^2(1 + \log \frac{n}{m})) = O(n^3)$ time and $O(nm)$ space.*

A straightforward reduction shows that computing edit distance between unrooted trees is at least as difficult as computing edit distance between rooted trees. Thus, invoking the lowerbound of Demaine et al. [13] we obtain that our algorithm is optimal if we restrict ourselves to the so-called decomposition algorithms, and by the result of Bringmann et al. [9] a significantly faster $O(n^{3-\varepsilon})$ algorithm is not possible assuming some popular conjecture.

Roadmap. In Section 2 we introduce the notation and the recursive formula that are then used to present Klein's algorithm adapted for the rooted case. Next, in Section 3 we return to the unrooted case, introduce new notation and transform both input trees by adding some auxiliary edges. Then, in Section 4 we present our new $O(n^3 \log \log n)$ algorithm for the

unrooted case which already improves the state-of-the-art Klein's algorithm and is essential for understanding our main $O(n^3)$ algorithm described in the full version of the paper. Both algorithms are described in a bottom-up fashion. In the simpler $O(n^3 \log\log n)$ version we first assume that one of the trees is a caterpillar and then generalize to arbitrary trees. In the more complicated $O(n^3)$ algorithm we start with an even more restricted case of one tree being a caterpillar and the other a rooted full binary tree. When analyzing both algorithms we only bound the total number of considered subproblems. As explained in the full version, this can be translated into an implementation with the same running time.

2 Preliminaries

We are given two unrooted trees T_1, T_2 with every edge labeled by an element of Σ and a cyclic order on the neighbors of every node. For every label $\alpha \in \Sigma$, we know the cost $c_{del}(\alpha) = c_{ins}(\alpha)$ of contracting or uncontracting of an edge with label α. For every $\alpha, \beta \in \Sigma$, $c_{match}(\alpha, \beta) = c_{match}(\beta, \alpha)$ is the cost of changing the label of an edge from α to β. All costs are non-negative and each edge can participate in at most one operation. Edit distance between T_1 and T_2 is defined as the minimum total cost of a sequence of the above operations transforming T_1 to T_2. Equivalently, it is the minimum cost of transforming both the trees to a common tree using only contracting and relabeling operations, as each operation has the same cost as its undo-counterpart. Note that for unrooted trees, edit distance is the minimum edit distance over all possible rootings of T_1 and T_2, where a rooting is uniquely determined by choice of the root and the leftmost edge from the root.

We first assume, that both trees are of equal size $n = |T_1| = |T_2|$, but later we will also address the case when one of them is significantly larger than the other. We start with the case when both trees are rooted, which is essential for the understanding of the unrooted case. Then, every node has its children ordered left-to-right. We also assume that both (rooted) trees are binary, as we can add $O(n)$ edges with a fresh label that costs 0 to contract and ∞ to relabel.

Naming convention. We use a similar naming convention as in [13]. We call main left and right edges of a (rooted) tree respectively the leftmost and rightmost edge from the root. For a given rooted tree T with at least 2 nodes, let r_T denote the right main edge of T and R_T denote the rooted subtree of T that is under (not including) r_T. By $T - r_T$ we denote a tree obtained from T by contracting edge r_T and by $T - R_T$ a tree obtained from T by contracting edge r_T and all edges from its subtree R_T. Thus the tree T consists of R_T, the edge r_T and edges $(T - R_T)$. l_T and L_T are defined analogously and T^v denotes subtree of T rooted at v. See Figure 2(a) and (b).

We define a pruned subtree of a tree T to be the tree obtained from T by a sequence of contractions of the left or right main edge. Note that every pruned subtree is uniquely represented by the pair of its left and right main edges. It also corresponds to an interval on the Euler tour of the tree started in the root when we remove from the interval each edge that occurs once. Thus we can completely represent a pruned subtree in $O(1)$ space by storing two edges. We can preprocess all the $O(n^2)$ pruned subtrees T' of a tree T to be able to obtain trees $R_{T'}, L_{T'}, T' - l_{T'}, T' - r_{T'}$ and edges $r_{T'}, l_{T'}$ in $O(1)$ time.

Dynamic programming. Zhang and Shasha [27] introduced the following recursive formula for computing the edit distance between two rooted trees:

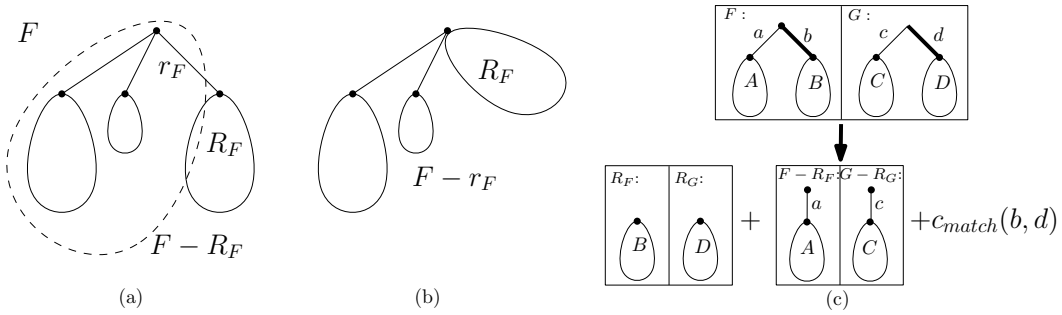


Figure 2 (a) Tree F with both r_F and R_G contracted. (b) F with its right main edge contracted. (c) When both right main edges are not contracted we obtain two independent problems.

▶ **Lemma 2.** *Let $\delta(F, G)$ be the edit distance between two pruned subtrees F and G of respectively T_1 and T_2. Then:*

- $\delta(\emptyset, \emptyset) = 0$
- $$\delta(F, G) = \min \begin{cases} \delta(F - r_F, G) + c_{del}(r_F) & \text{if } F \neq \emptyset \\ \delta(F, G - r_G) + c_{del}(r_G) & \text{if } G \neq \emptyset \\ \delta(R_F, R_G) + \delta(F - R_F, G - R_G) + c_{match}(r_F, r_G) & \text{if } F, G \neq \emptyset \end{cases}$$

The above recurrence also holds if we contract or match the left main edge.

It contracts the right main edge in one of the two trees or matches the right main edges of the two trees. In the latter case, we get two independent subproblems (R_F, R_G) and $(F - R_F, G - R_G)$ that must be transformed to equal trees. See Figure 2(c) for an illustration of this case.

To estimate time complexity of the algorithm, we only count different pairs (F, G) for which $\delta(F, G)$ is computed. Each such value is computed at most once and stored. Note that F is always a pruned subtree of T_1, while G is a pruned subtree of T_2, thus there are $O(n^4)$ possible pairs (F, G). In the worst case, all such pairs might be considered. The formula from Lemma 2 can be evaluated in constant time, and any previously computed value can be retrieved in constant time from a four-dimensional table.

The above algorithm always contracts or relabels the right main edge. A more deliberate choice of direction (whether to choose the left or right main edge) will lead to a different behavior of the algorithm which in turn might result in a smaller total number of considered pairs (F, G). Such a family of algorithms is called decomposition algorithms. When analyzing the time complexity of such an algorithm, we assume that any already computed $\delta(F, G)$ can be retrieved in constant time. If our goal is to compute significantly fewer than $O(n^4)$ subproblems, we cannot afford to allocate the four-dimensional table anymore. An obvious solution is to store the already computed values in a hash table, but this requires randomization. In the full version [14] we explain how to carefully arrange the order of the computation and store the partial results as to obtain deterministic algorithms with the same running time.

While the formula from Lemma 2 suggests a top-down strategy, we phrase the algorithms in a bottom-up perspective, which allows us to present the details of the computation more precisely. The aim of all the algorithms is to compute $\delta(T_1, T_2)$ knowing only $\delta(\emptyset, \cdot)$ and $\delta(\cdot, \emptyset)$, as the costs of contraction of an arbitrary pruned subtree are precomputed.

Algorithm 1 Klein's algorithm.

```
1: for each heavy path H in T_1 in the bottom-up order do
2:     let v_1, v_2, ..., v_{|H|} = H
3:     for i = |H| - 1, ..., 0 do
           ▷ avoiding the heavy child:
4:         ComputeFrom(δ(T_1^{v_i}, ·), δ(T_1^{v_{i+1}}, ·))
```

Klein's $O(n^3 \log n)$ algorithm. Klein's algorithm [21] uses heavy path decomposition [28] of T_1. The root is called light and every node calls its child with the largest subtree (and the leftmost in case of ties) heavy and all other children light. An edge is heavy if it leads to the heavy child.

While applying the dynamic formula from Lemma 2, Klein's algorithm uses a strategy that we call "avoiding the heavy child" in T_1. It chooses the direction (either left or right) in such a way that the edge leading to the heavy child of the root is contracted or relabeled as late as possible. Observe that contracting the main edge not leading to the heavy child of the root of a pruned subtree T, does not change the heavy child of the root of T, as its subtree is still the largest. Note that Klein's strategy does not depend on the considered pruned subtree of T_2.

Even though Klein uses top-down view to describe his algorithm, we find it more convenient to implement the computations in bottom-up order. Therefore the algorithm processes heavy paths of T_1 in the bottom-up order as shown in Algorithm 1. Consider a heavy path H with nodes $v_1, v_2, \ldots, v_{|H|}$ where v_1 is the closest node to the root and $v_{|H|}$ is a leaf. By $\delta(T_1^v, \cdot)$ we denote a table of $O(n^2)$ distances between tree T_1^v and all pruned subtrees of T_2. The algorithm considers all nodes on H also bottom-up. It starts from $\delta(T_1^{v_{|H|}}, \cdot) = \delta(\emptyset, \cdot)$, which is precomputed, and then iteratively computes $\delta(T_1^{v_i}, \cdot)$ from $\delta(T_1^{v_{i+1}}, \cdot)$ for decreasing values of i. We denote such a step by ComputeFrom subroutine. Note that in every step the strategy avoiding the heavy child always chooses the same direction (recall that the tree is binary) and visits altogether at most $O(n)$ pruned subtrees of T_1. Also when actually implementing the ComputeFrom step we proceed bottom-up. That is, suppose we have already computed $\delta(T_1^{v_{i+1}}, \cdot)$ and that v_{i+1} is the left child of v_i. Then the strategy avoiding the heavy child says R that is chooses first the right main edge to consider. We compute $\delta(T_1^{v_i}, \cdot)$ as follows. First we consider the tree $T_1^{v_{i+1}} \cup \{\{v_i, v_{i+1}\}\}$ (we call this uncontracting the heavy edge), next $T_1^{v_{i+1}} \cup \{\{v_i, v_{i+1}\}, \{v_i, w\}\}$ if exists a light child w of v_i and then uncontract the subsequent edges of T_1^w. This guarantees that while computing $\delta(F, G)$ the subtrees $F - r_F$ and $F - R_F$ have been already processed. Pruned subtrees of T_2 are also considered in the order of increasing sizes. Clearly, as argued for Zhang and Shasha's algorithm, the algorithm visits $O(n^2)$ pruned subtrees of T_2, so we need to bound the number of pruned subtrees of T_1.

▶ **Observation 3.** Consider an arbitrary tree T. Suppose that strategy avoiding the heavy child in T says R for a pruned subtree F. Then $F - R_F$ is also obtained by a sequence of contractions of the main edge according to the strategy.

The observation implies that in order to count the relevant intervals of T_1 we can only consider the trees obtained by contraction of the main edge according to the strategy, and trees of the form L_F and R_F. Note that the only trees of the form L_F or R_F that are not obtained in this way are rooted at a light node so will be counted separately for another heavy path.

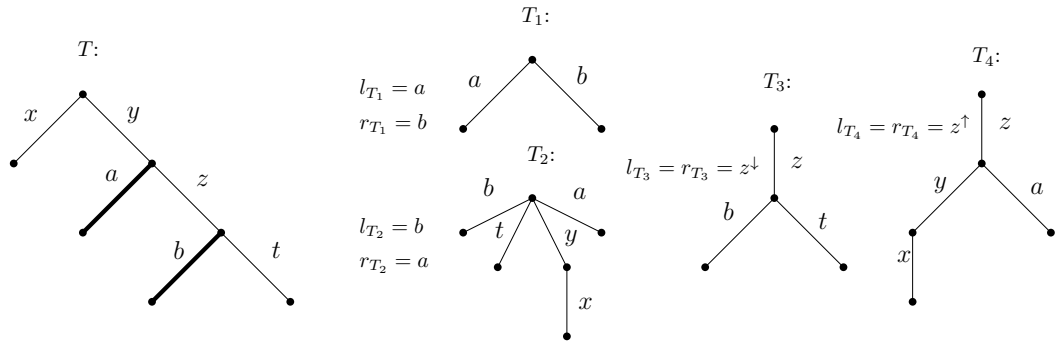


Figure 3 Every pruned subtree is uniquely represented by its left and right main edges or a dart.

We denote $\mathsf{apex}(F)$ to be the top node of the heavy path containing the lowest common ancestor of all endpoints of edges of F. In other words, $\mathsf{apex}(F)$ is the lowest light ancestor of all edges of F. Now grouping all the visited pruned subtrees by their apex-es we bound their total number:

▶ **Observation 4.** For an arbitrary tree T, there is $\sum_{v\text{: light node in } T} |T^v|$ pruned subtrees of T visited while applying strategy avoiding the heavy child of T.

Let the light-depth $\mathsf{ldepth}(u)$ of a node u be the number of light nodes that are ancestors of u (node is also an ancestor of itself). Because $\mathsf{ldepth}(u) \leq \log(n) + 1$, we obtain:

$$\sum_{v\text{: light node in } T_1} |T_1^v| = \sum_{v\text{: node in } T_1} \mathsf{ldepth}(v) \in O(n \log n) \tag{1}$$

Recalling that there are $O(n^2)$ relevant intervals of T_2 we conclude that Klein's algorithm visits $O(n^3 \log n)$ subproblems. As we assume the constant time memoization, it runs in $O(n^3 \log n)$ time.

3 Back to Unrooted Case

Recall that edit distance between two unrooted trees T_1 and T_2 is the minimum edit distance between T_1 and T_2 over all possible rootings of them, where rooting is determined by the root of the tree and its the left main edge. As Klein [21] mentioned, it is enough to choose an arbitrary rooting in one of the trees and try all possible rootings of the other to find an optimal setting. Observe, that we can treat the Euler tour of T_2 as a cyclic string and represent every pruned subtree of T_2 as an interval of it, for all possible rootings of T_2. Thus Klein's algorithm works in $O(n^3 \log n)$ time also for the edit distance between unrooted trees. Before we present our faster algorithm for this case, we need to introduce some new definitions. Recall, that even in the unrooted case, we first arbitrarily root both trees and the initial rooting remains unchanged throughout the algorithm.

Darts. We replace every edge e with two darts corresponding to two ways of traversing the edge, either down $e^{\downarrow}$ or up the tree $e^{\uparrow}$ (with respect to the fixed rooting). Subtree of a dart $\mathsf{subtree}((u, v))$ is defined as the subtree rooted at node v, when u is its parent. Note that $e^{\uparrow}$ and $e^{\downarrow}$ belong neither to $\mathsf{subtree}(e^{\uparrow})$ nor to $\mathsf{subtree}(e^{\downarrow})$. Every pruned subtree of (unrooted) tree is uniquely represented by its left and right main edges or a dart (if there is one edge from the root). See Figure 3.

Auxiliary edges for rootings. We observed that every rooting of T_2 corresponds to a subrange of a cyclic Euler tour E_{T_2}, but later it will be convenient to represent every rooting as a subtree of a dart. For this purpose, we add new edges labeled with a fresh label $\# \notin \Sigma$ which will be used only to denote a rooting. Setting $c_{del}(\#) = 0$ and $c_{match}(\#, \cdot) = \infty$ we force that these edges are only contracted. For every node v we add new edges alternating with the original ones. Thus in total, there are $2(n-1)$ edges added. Using these new edges we can compute edit distance between the unrooted trees from the values of $\delta(T_1, \mathsf{subtree}(d))$ for all darts d in T_2. Thus, our aim is to fill the table Δ where $\Delta[u, d] := \delta(T_1^u, \mathsf{subtree}(d))$.

Auxiliary edges to bound the degrees. As the last step, again we add $O(n)$ edges with appropriate costs as to ensure that the degree of every node is at most 3. Observe that the cost of the optimal solution for the modified trees is the same as for the initial ones and having a sequence of operations for the modified trees, we can easily obtain an optimal sequence for the original instance of the problem.

4 $O(n^3 \log\log n)$ Algorithm for Unrooted Case

After initial modifications both trees are binary and the algorithm needs to fill the table Δ where $\Delta[u, d] := \delta(T_1^u, \mathsf{subtree}(d))$ for all nodes $u \in T_1$ and darts $d \in T_2$. We first run Demaine et al.'s algorithm operating on labels on edges instead of nodes which computes $\delta(T_1^u, T_2^v)$ for all nodes $u \in T_1$ and $v \in T_2$ in $O(n^3)$ time and stores them in $\Delta[u, e_v^{\downarrow}]$ where $e_v^{\downarrow}$ is the dart to v from its parent. Now we need to fill the remaining fields $\Delta[u, e^{\uparrow}]$ for all darts $e^{\uparrow}$ up the tree T_2.

This is the main difficulty in the unrooted case, in which we need to handle many big subtrees which are significantly different from each other. Our approach is to successively reduce different subproblems to smaller ones, in a way that there are fewer subproblems to consider in the next step. We use divide and conquer paradigm, in which there is more and more sharing after every step.

In the beginning, we call each node of T_1 and T_2 light or heavy as in the Klein's algorithm and all the time the notion is with respect to the initial rootings. Similarly, the notion of traversing an edge up or down the tree is always with respect to the rooting. Recall that we denote $\mathsf{apex}(T)$ as the top node on the heavy path containing the lowest common ancestor of all edges of T. We first fix a global value b, which will be determined exactly later. On a high level, from the top-down perspective, the algorithm uses the following strategy to compute $\delta(F, G)$: if $|T_1^{\mathsf{apex}(F)}| > n/b$, then avoid the heavy child in F, and otherwise apply a new strategy based only on G and T_2.

Considering it bottom-up, the algorithm first fills values of $\Delta[u, e^{\uparrow}]$ for all nodes u such that $|T_1^{\mathsf{apex}(u)}| \leq n/b$ and all darts up T_2. For the remaining fields of Δ, it uses strategy avoiding the heavy child in T_1. As in the Klein's algorithm, in this phase, the algorithm needs to process heavy paths of T_1 in the bottom-up order. Note that for each light node v such that $|T_1^v| > n/b$ holds $\mathsf{ldepth}(v) < \log b + 1$. Thus there are $O(n^3 \log b)$ subproblems visited in total in this phase.

For the other phase note that there are $O(n^2/b)$ relevant subtrees in T_1, and now we need to carefully design and analyze the new strategy for T_2. It will be easier to think, that in this phase the algorithm needs to compute $\Delta[u, e^{\uparrow}]$ for all darts $e^{\uparrow}$ up T_2 and all nodes $u \in T_1$ such that $|T_1^u| \leq n/b$, call them interesting. Clearly, all subproblems in which there is a switch to the strategy based on T_2 are of this form.

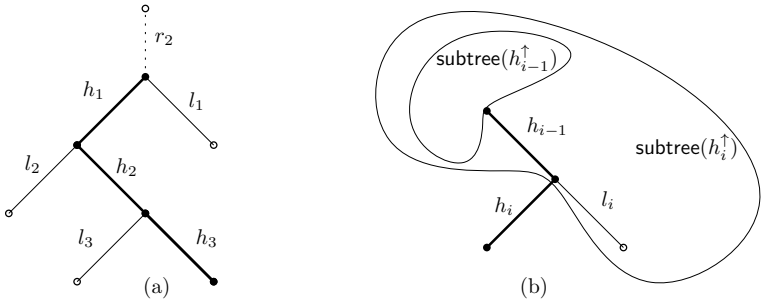


Figure 4 (a) A heavy path H with edge r_2 (dotted) denoting the rooting of T_2. (b) To compute $\delta(*, \mathsf{subtree}(h_i^{\uparrow}))$ we use $\delta(*, \mathsf{subtree}(h_{i-1}^{\uparrow}))$, first uncontract the edge h_{i-1} and then l_i (if exists).

As now the strategy will be more complex than before, we first describe it for the case when T_2 is a caterpillar: a heavy path with possibly single nodes connected to it. This example is already difficult in the unrooted case and will require divide and conquer approach to handle all the possible rootings of T_2 at once. Next, we will slightly modify the approach to handle arbitrary trees T_2.

4.1 Caterpillar T_2

Now we consider the case when T_2 is a heavy path H with possibly single nodes connected to it. Let h_i denote (heavy) edges on H, $r_2 = h_0$ be the edge denoting the initial rooting of T_2 and (if exists) l_i be the light edge connected to the i-th node on H. See Figure 4(a) for an example.

In the first step we compute values of $\delta(*, \mathsf{subtree}(h_i^{\uparrow}))$ for all heavy edges h_i, where $*$ denotes all pruned subtrees of T_1 of size at most n/b. The strategy is to avoid the parent, that is to contract the edge leading to the parent as late as possible. See Figure 4(b).

More precisely, in the beginning, we already know $\delta(*, \mathsf{subtree}(h_0^{\uparrow}))$, because it is the cost of contraction of the whole pruned subtree of T_1 (which is precomputed), as $h_0 = r_2$ and $\mathsf{subtree}(h_0^{\uparrow}) = \emptyset$. Then, having values of $\delta(*, \mathsf{subtree}(h_{i-1}^{\uparrow}))$ we compute $\delta(*, \mathsf{subtree}(h_i^{\uparrow}))$ by uncontracting first h_{i-1} and then l_i if it exists. It is an extension of the ComputeFrom subroutine, but now we do not have subtrees T^x and T^y, where x is the parent of y, but two edges h_i and h_{i-1} with a common endpoint. Note that in this step all uncontractions are from the same direction.

There are $O(n)$ pruned subtrees of T_2 obtained by uncontractions of a main edge according to the strategy, starting from the empty subtree. Now we need to show that the algorithm did not consider any other pruned subtree of T_2. Suppose it uncontracted the left main edge. Then $G - L_G \in \{\emptyset, G - l_G\}$, depending on whether l_G was the heavy edge leading to the parent or not. Also $L_G \in \{\emptyset, G - l_G\}$, so in both cases, all the obtained pruned subtrees are among the $O(n)$ described above. Finally, as there are $O(n^2/b)$ pruned subtrees of T_1, in total we computed and stored the edit distance of $O(n^3/b)$ subproblems. Now, using the computed values we fill $\Delta[u, h_i^{\uparrow}]$ for all interesting nodes $u \in T_1$ and heavy edges $h_i \in T_2$. Thus, later on, we do not have to consider the pruned subtrees of the form $\delta(L_F, L_G)$ or $\delta(R_F, R_G)$ as their values are already stored in Δ, because either one of them is empty or they are of the form $\delta(T_1^v, \mathsf{subtree}(dh))$ for an interesting node $u \in T_1$ and a dart dh from a heavy edge in T_2. We only have not computed values $\Delta[u, l^{\uparrow}]$ for darts from light edges up the tree, but in this phase of the algorithm, they never appear in $\delta(L_F, L_G)$ or $\delta(R_F, R_G)$ subproblem. However, we need to compute these values because they correspond to some rootings of T_2, so we will consider them in the following paragraph.

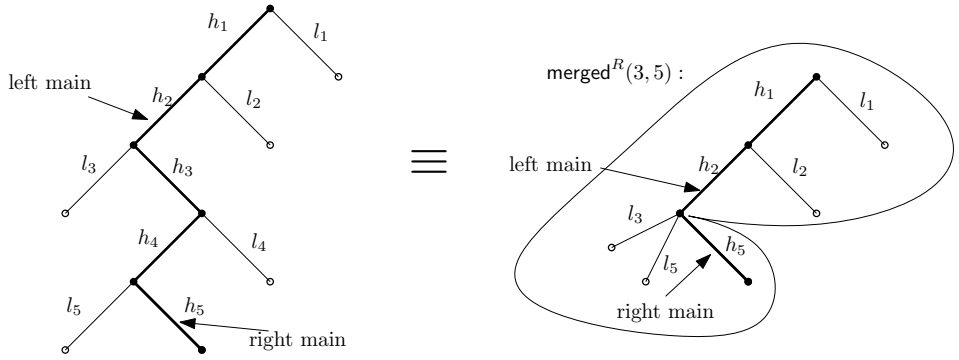


Figure 5 Pruned subtree $\mathsf{merged}^R(3,5)$ has the left main edge h_2 and right h_5.

Algorithm 2 Computes input tables needed for processing a heavy path H

1: **function** ProcessHeavyPath($\delta(*, \mathsf{subtree}(h_0^\uparrow))$)
2: **for** $i = 1..|H|$ **do**
▷ avoiding the parent:
3: ComputeFrom($\delta(*, \mathsf{subtree}(h_i^\uparrow)), \delta(*, \mathsf{subtree}(h_{i-1}^\uparrow))$)
4: fill $\Delta[u, h_i^\uparrow]$ for all interesting nodes u

▷ repeatedly uncontracting the left main edge:
5: ComputeFrom($\delta(*, \mathsf{merged}^R(1, |H|)), \delta(*, \mathsf{subtree}(h_0^\uparrow))$)
▷ repeatedly uncontracting the right main edge:
6: ComputeFrom($\delta(*, \mathsf{merged}^L(1, |H|)), \delta(*, \mathsf{subtree}(h_0^\uparrow))$)

7: Group($1, |H|, \mathsf{Data}(1, |H|)$)

Darts from light nodes up the tree. From now on, our algorithm processes heavy paths of T_2 one-by-one. In particular, in this subsection, we process the only heavy path H of T_2. Thus, unless explicitly stated otherwise all the notion is relative to the current heavy path H. First, we define $\mathsf{merged}^R(A, B)$ as the pruned subtree obtained by contraction of edges between the A-th and B-th node on H or to the right of H:

▶ **Definition 5.** Let H be a heavy path and A and B ($A \leq B$) denote indices of two nodes on H. Then $\mathsf{merged}^R(A, B)$ is a tree with the left main edge h_{A-1} and the right main edge h_B. $\mathsf{merged}^L(A, B)$ is a tree with the left main edge h_B and the right main edge h_{A-1}.

See Figure 5. Note that $\mathsf{subtree}(l_A^\uparrow)$ is either $\mathsf{merged}^R(A, A)$ or $\mathsf{merged}^L(A, A)$, depending on which side of H is l_A.

As explained earlier, in the beginning the algorithm computes $\delta(*, \mathsf{subtree}(h_i^\uparrow))$ for all heavy edges on H. Additionally, it calculates $\delta(*, \mathsf{merged}^L(1, |H|))$ and $\delta(*, \mathsf{merged}^R(1, |H|))$ from $\delta(*, \mathsf{subtree}(h_0^\uparrow))$ by repeatedly uncontracting respectively the right and left main edge. See Algorithm 2 for the summary of the whole preprocessing. Then it calls a recursive procedure Group($1, |H|, \mathsf{Data}(1, |H|)$). The final goal of this call is to fill $\Delta[u, l_i^\uparrow]$ for all light edges l_i connected to the heavy path H.

Group($A, B, \mathsf{Data}(A, B)$) is a procedure which considers an interval $[A, B]$ of indices on H given tables of values $\delta(*, \mathsf{subtree}(h_{A-1}^\uparrow)), \delta(*, \mathsf{subtree}(h_B^\downarrow)), \delta(*, \mathsf{merged}^L(A, B))$ and $\delta(*, \mathsf{merged}^R(A, B))$, which we denote as $\mathsf{Data}(A, B)$. Intuitively, $\mathsf{Data}(A, B)$ contains in-

Algorithm 3 Fills $\Delta[u, l_i^{\uparrow}]$ for light edges l_i connected to the heavy path H with $i \in [A, B]$.

```
1:  function GROUP(A, B, Data(A, B))
2:      if A = B then
3:          if there is a light edge l_A connected to H then
4:              fill Δ[u, l_A^↑] for interesting nodes u ∈ T_1
5:          return
6:      M := ⌊(A + B)/2⌋
7:      for i = (B − 1)..M do
            ▷ avoiding the heavy child:
8:          COMPUTEFROM(δ(∗, subtree(h_i^↓)), δ(∗, subtree(h_{i+1}^↓)))
        ▷ repeatedly uncontracting the right main edge:
9:      COMPUTEFROM(δ(∗, merged^R(A, M)), {δ(∗, merged^R(A, B)); δ(∗, subtree(h_{A−1}^↑))})
        ▷ repeatedly uncontracting the left main edge:
10:     COMPUTEFROM(δ(∗, merged^L(A, M)), {δ(∗, merged^L(A, B)); δ(∗, subtree(h_{A−1}^↑))})

11:     GROUP(A, M, Data(A, M))
12:     symmetric computations for interval [M + 1, B]
13:     GROUP(M + 1, B, Data(M + 1, B))
```

formation about subtrees "outside" the considered interval $[A, B]$ which are relevant during intermediate computations. Then, the procedure calls itself recursively for shorter intervals until it holds that $A = B$ when $\delta(*, \mathsf{merged}^L(A, A))$ or $\delta(*, \mathsf{merged}^R(A, A))$ contains the fields of $\Delta[u, l_A^{\uparrow}]$ for all interesting nodes u and then the recurrence stops.

In more detail, for an interval $[A, B]$, the procedure computes $\mathsf{Data}(A, M)$ and $\mathsf{Data}(M + 1, B)$ for $M = \lfloor \frac{A+B}{2} \rfloor$ and calls itself recursively for the smaller intervals. Note that for $\mathsf{Data}(A, M)$ it needs to compute tables $\delta(*, G)$ for trees $G = \mathsf{merged}^R(A, M), \mathsf{merged}^L(A, M)$ or $\mathsf{subtree}(h_M^{\downarrow})$ and can reuse table $\delta(*, \mathsf{subtree}(h_{A-1}^{\uparrow}))$ which is a part of $\mathsf{Data}(A, B)$. Similarly for interval $[M + 1, B]$. See Algorithm 3.

To analyze the complexity of the GROUP procedure, first note that in every step of the loop in line 7, it considers a constant number of pruned subtrees from T_2, so in total there are $O(B - M)$ of them. After this loop, we have $\delta(*, \mathsf{subtree}(h_M^{\downarrow}))$ computed.

The call of COMPUTEFROM in line 9 needs more input than the call in line 8, even though the strategy is always uncontracting the right main edge. Note that if the dynamic program only tried contracting the right main edge, it would be possible to compute $\delta(*, \mathsf{merged}^R(A, M))$ only from $\delta(*, \mathsf{merged}^R(A, B))$. However, it is not the case when the algorithm also matches edges. The first case when r_G is a light edge ($r_G = l_X$ for some value of X) is not problematic, because then $R_G = \emptyset$ and $G - R_G = G - r_G$, so this pruned subtree is already visited. Although, if r_G is a heavy edge then $R_G = \mathsf{subtree}(r_G^{\downarrow})$ and $G - R_G$ is a pruned subtree, which has not been considered yet. Observe that in this situation the pruned subtree can be obtained from $\mathsf{subtree}(h_{A-1}^{\uparrow})$ by a sequence of $O(B - A)$ contractions of the right main edge, so we need it as a separate input to the COMPUTEFROM subroutine. A similar reasoning applies to the edges to the left of H in line 10 and to the computations for interval $[M + 1, B]$.

To sum up, one call of GROUP(A, B) (not including recursive calls) visits $O(B - A)$ pruned subtrees of T_2. As we start from an interval of length $|H|$ and in every recursive call its length is roughly halved, the procedure considers in total $O(|H| \log |H|) = O(n \log n)$ pruned subtrees of T_2.

4.2 Arbitrary Tree T_2

Now we describe, how to modify the above algorithm to process not only a caterpillar, but an arbitrary tree T_2. In this case, there can be non-empty subtrees connected to the main heavy path.

Note that for an arbitrary heavy path H inside T_2, the ProcessHeavyPath procedure only needs to know $\delta(*, \mathsf{subtree}(h_0^{\uparrow}))$ to be able to compute all the remaining input parameters in $\mathsf{Data}(1, |H|)$, because $\delta(*, \mathsf{subtree}(h_{|H|}^{\downarrow})) = \delta(*, \emptyset)$ is precomputed. In the beginning, the algorithm calls $\textsc{ProcessHeavyPath}_{H^0}(\delta(*, \emptyset))$, where H^0 is the heavy path of T_2 containing the root of T_2. The only place we need to change inside the Group procedure to handle arbitrary trees T_2 is to not only fill $\Delta[u, l_A^{\uparrow}]$ in line 4 of Algorithm 3, but also recursively call $\textsc{ProcessHeavyPath}_{H'}(\delta(*, \mathsf{subtree}(l_A^{\uparrow})))$ where H' is the heavy path connected to the A-th node of the considered heavy path. As we pointed earlier, $\mathsf{subtree}(l_A^{\uparrow})$ is either $\mathsf{merged}^R(A, A)$ or $\mathsf{merged}^L(A, A)$, depending on which side of H is l_A. Now observe, that each subsequent pruned subtree that appears in the recursive formula is already visited and processed:

▶ **Observation 6.** In the modified Group procedure, during the call of ComputeFrom subroutine in line 9 of Algorithm 3, all the intermediate pruned subtrees of T_2 are obtained by a sequence of uncontractions of the right main edge from the root either from $\mathsf{merged}^R(A, B)$ or $\mathsf{subtree}(h_{A-1}^{\uparrow})$. A similar property holds for the other three calls of ComputeFrom in lines 10 and 12.

What changes in the analysis of the procedure is that now there are not $O(|H| \log |H|)$ pruned subtrees of T_2 but $O(|T_2^v| \log |H|) = O(|T_2^v| \log n)$, where v is the top node of H. In other words, the heavy path H itself might be short, but there might be big subtrees connected to it. However, every subtree connected to H is completely contracted (edge-by-edge) a constant number of times on every level of recursion of Group procedure and thus the bound.

Recall that top node of every heavy path is light, so using equation (1) we bound the overall number of subtrees of T_2 considered during this part of the algorithm:

$$\sum_{v\text{: top node of a heavy path in } T_2} |T_2^v| \cdot \log n = \sum_{v\text{: light node in } T_2} |T_2^v| \cdot \log n \in O(n \log^2 n)$$

4.3 Final Analysis

To conclude, the above algorithm computes $\Delta[u, e^{\uparrow}]$ for all nodes $u \in T_1$ such that $|T_1^u| \leq n/b$ and all darts up the tree T_2 by considering $O(n \log^2 n)$ pruned subtrees of T_2 and $O(n^2/b)$ of T_1. At the beginning of Section 4 we described the second phase of the algorithm, which avoids the heavy child in T_1 and fills the remaining fields of Δ considering $O(n \log b)$ pruned subtrees of T_1 and $O(n^2)$ of T_2. Thus, during the two phases, the whole algorithm visits $O(n^3 \frac{\log^2 n}{b} + n^3 \log b)$ subproblems. Setting $b = \log^2 n$ we obtain the overall complexity $O(n^3 \log \log n)$.

References

1 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *48th STOC*, pages 375–388, 2016.

2 Tatsuya Akutsu, Daiji Fukagawa, and Atsuhiro Takasu. Approximating tree edit distance through string edit distance. *Algorithmica*, 57(2):325–348, 2010.

3 Rajeev Alur, Loris D'Antoni, Sumit Gulwani, Dileep Kini, and Mahesh Viswanathan. Automated grading of DFA constructions. In *23rd IJCAI*, pages 1976–1982, 2013.

4 Taku Aratsu, Kouichi Hirata, and Tetsuji Kuboyama. Approximating tree edit distance through string edit distance for binary tree codes. *Fundam. Inf.*, 101(3):157–171, 2010.

5 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *47th STOC*, pages 51–58, 2015.

6 J. Bellando and R. Kothari. Region-based modeling and tree edit distance as a basis for gesture recognition. In *10th ICIAP*, pages 698–703, 1999.

7 Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3):217–239, 2005.

8 Norbert Blum and Kurt Mehlhorn. On the average number of rebalancing operations in weight-balanced trees. *Theor. Comput. Sci.*, 11(3):303–320, 1980.

9 Karl Bringmann, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Tree edit distance cannot be computed in strongly subcubic time (unless APSP can). In *29th SODA*, 2018.

10 Peter Buneman, Martin Grohe, and Christoph Koch. Path queries on compressed XML. In *29th VLDB*, pages 141–152, 2003.

11 Sudarshan S. Chawathe. Comparing hierarchical data in external memory. In *25th VLDB*, pages 90–101, 1999.

12 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *36th STOC*, pages 91–100, 2004.

13 Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Trans. Algorithms*, 6(1):2:1–2:19, 2009.

14 Bartłomiej Dudek and Paweł Gawrychowski. Edit distance between unrooted trees in cubic time. *CoRR*, abs/1804.10186, 2018. `arXiv:1804.10186`.

15 Serge Dulucq and Hélène Touzet. Decomposition algorithms for the tree edit distance problem. *J. Discrete Algorithms*, 3(2-4):448–471, 2005.

16 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1):4:1–4:33, 2009.

17 Matthias Höchsmann, Thomas Töller, Robert Giegerich, and Stefan Kurtz. Local similarity in RNA secondary structures. In *2nd CSB*, pages 159–168, 2003.

18 Christoph M. Hoffmann and Michael J. O'Donnell. Pattern matching in trees. *J. ACM*, 29(1):68–95, Jan. 1982.

19 Egor Ivkin. Approximating tree edit distance through string edit distance for binary tree codes. B.Sc. thesis, Charles University in Prague, 2012.

20 Philip Klein, Srikanta Tirthapura, Daniel Sharvit, and Ben Kimia. A tree-edit-distance algorithm for comparing simple, closed shapes. In *11th SODA*, pages 696–704, 2000.

21 Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *6th ESA*, pages 91–102, 1998.

22 Philip N. Klein, Thomas B. Sebastian, and Benjamin B. Kimia. Shape matching using edit-distance: An implementation. In *12th SODA*, pages 781–790, 2001.

23 Mateusz Pawlik and Nikolaus Augsten. Efficient computation of the tree edit distance. *ACM Trans. Database Syst.*, 40(1):3:1–3:40, 2015.

24 Juan Ramón Rico-Juan and Luisa Micó. Comparison of aesa and laesa search algorithms using string and tree-edit-distances. *Pattern Recogn. Lett.*, 24(9-10):1417–1426, jun 2003.

25 T. B. Sebastian, P. N. Klein, and B. B. Kimia. Recognition of shapes by editing their shock graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(5):550–571, May 2004.

26 B. A. Shapiro and K. Z. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Comput. Appl. Biosci.*, 6(4):309–318, Oct. 1990.

27 Dennis Shasha and Kaizhong Zhang. Fast algorithms for the unit cost editing distance between trees. *J. Algorithms*, 11(4):581–621, dec 1990.

28 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

29 Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

30 Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.

31 Xuchen Yao, Benjamin Van Durme, Chris Callison-Burch, and Peter Clark. Answer extraction as sequence tagging with tree edit distance. In *HLT-NAACL*, pages 858–867, 2013.

A Note on Two-Colorability of Nonuniform Hypergraphs

Lech Duraj
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland
lech.duraj@uj.edu.pl

Grzegorz Gutowski
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland
grzegorz.gutowski@uj.edu.pl
https://orcid.org/0000-0003-3313-1237

Jakub Kozik
Theoretical Computer Science Department, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland
jakub.kozik@uj.edu.pl

Abstract

For a hypergraph H, let $q(H)$ denote the expected number of monochromatic edges when the color of each vertex in H is sampled uniformly at random from the set of size 2. Let $s_{\min}(H)$ denote the minimum size of an edge in H. Erdős asked in 1963 whether there exists an unbounded function $g(k)$ such that any hypergraph H with $s_{\min}(H) \geqslant k$ and $q(H) \leqslant g(k)$ is two colorable. Beck in 1978 answered this question in the affirmative for a function $g(k) = \Theta(\log^* k)$. We improve this result by showing that, for an absolute constant $\delta > 0$, a version of random greedy coloring procedure is likely to find a proper two coloring for any hypergraph H with $s_{\min}(H) \geqslant k$ and $q(H) \leqslant \delta \cdot \log k$.

2012 ACM Subject Classification Mathematics of computing → Combinatorics, Mathematics of computing → Hypergraphs, Mathematics of computing → Probabilistic algorithms

Keywords and phrases Property B, Nonuniform Hypergraphs, Hypergraph Coloring, Random Greedy Coloring

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.46

Funding This work was partially supported by Polish National Science Center (2016/21/B/ST6/02165)

1 Introduction

A *hypergraph* $H = (V, E)$ is a finite set of vertices V and a set of edges E where each edge is a set of at least two vertices. A *two coloring* of H is an assignment of color blue or red to each vertex in H. A coloring is *proper* if each edge in H contains both a vertex colored blue and a vertex colored red. We say that H is *two colorable* if it admits a proper two coloring. Hypergraph H is *k-uniform* if every edge in H has size exactly k – we also say that H is a *k-graph*. For every $n \in \mathbb{N}$, the set $\{1, \ldots, n\}$ is denoted by $[n]$. We use standard O-notation to describe asymptotic properties of various functions.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 46; pp. 46:1–46:13


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

One of the most classical problems in the extremal combinatorics is to find the minimum number of edges $m(k)$ in a k-uniform hypergraph that is not two colorable. The research on this problem has been started in the 60s by Erdős and Hajnal [5], who used the term *Property B* for two colorability. Today, by the result of Radhakrishnan and Srinivasan [10], we know that $m(k) = \Omega((k/\log k)^{1/2}) \cdot 2^k$. The best known upper bound, proved by Erdős [4] in 1964, is $m(k) = O(k^2) \cdot 2^k$. This upper bound follows from the fact that a random k-graph with k^2 vertices and $O(k^2) \cdot 2^k$ edges is very unlikely to be two colorable. Interestingly, known deterministic constructions require much larger structures – the best one is by Gebauer [7] and gives a not two colorable k-graph with roughly $2^{k+k^{2/3}}$ edges.

Lovász [8] proved that for $k \geqslant 3$, the problem of deciding if a given k-graph is two colorable is NP-complete. For k-graphs with the number of edges smaller than $m(k)$ the decision problem is trivial – by the definition they are all two colorable. Nevertheless, constructing a two coloring of such k-graphs is not necessarily an easy task. Luckily, the known lower bounds for $m(k)$ are constructive. In fact, the bound of [10] is proved by showing that some randomized coloring procedure succeeds with high probability for the considered hypergraphs. Cherkashin and Kozik [2] showed that the same bound is obtained by the analysis of a random greedy algorithm (i.e., a procedure that colors the vertices of a hypergraph in a random order and assigns color blue to each vertex unless it is the last vertex of a monochromatic blue edge – only then color red is assigned).

For a hypergraph $H = (V, E)$, let $q(H)$ denote the expected number of monochromatic edges when the color of each vertex is sampled uniformly at random. Clearly, for a k-graph H, we have $q(H) = |E| \cdot 2^{-k+1}$, and determining the value of $m(k)$ is equivalent to finding a not two colorable k-graph H with the minimal possible value of $q(H)$. This formulation allows for a neat generalization of the question to hypergraphs with edges of arbitrary sizes (i.e., nonuniform hypergraphs). For a hypergraph $H = (V, E)$, let $s_{\min}(H) = \min_{e \in E} |e|$ and observe that $q(H) = \sum_{e \in E} 2^{-|e|+1}$. Erdős [3, 6] asked whether there exists an unbounded function g such that any hypergraph H with $s_{\min}(H) \geqslant k$ and $q(H) \leqslant g(k)$ is two colorable. A positive answer has been given in 1978 by Beck [1] who proved the result for $g(k) = \Theta(\log^*(k))$. This has not been improved since then. (In 2008 Lu [9] announced a proof of a bound $\Omega(\log(k)/\log\log(k))$, but it turned out to work only for some specific class of hypergraphs[1]. The class contains all simple hypergraphs, but for these Shabanov in [11] improved the bound to $\Omega(\sqrt{k})$.) In this paper we prove the same result for $g(k) = \Theta(\log(k))$. The random construction of a not two colorable k-graph by Erdős [4] shows that the best possible g is $O(k^2)$, even when restricted to uniform hypergraphs. Interestingly, there are no better nonuniform constructions known. Our main result is the following.

▶ **Theorem 1.** *There exists a constant $\delta > 0$ such that for all sufficiently large k, any hypergraph $H = (V, E)$ with $s_{\min}(H) \geqslant k$ and $q(H) \leqslant \delta \cdot \log k$ is two colorable.*

Moreover, we prove the theorem by showing that a version of a random greedy coloring procedure succeeds with positive probability for these hypergraphs.

[1] The proof in the preprint available on the authors web page is very close to the developments of Section 3.3 "Simple bound". In our opinion it is incorrect, since it does not take into account that a vertex of some edge f can be recolored because of some other edge e with $|e \cap f| > 1$ (despite irreducibility). We avoid this problem by considering the vertices in a random order and allowing only the last vertex of an initially monochromatic edge to be recolored, provided that the edge is not already repaired.

2 Basic notions and the coloring procedure

2.1 Tools

We start with a simple lemma on convex functions of random variables.

▶ **Lemma 2.** *Let X be a nonnegative random variable such that $0 \leqslant X \leqslant M$ and $\mathbb{E}[X] \leqslant \lambda M$ for some $M \geqslant 0$, and $0 \leqslant \lambda \leqslant 1$. Then, for any convex function $f : [0, M] \to [0, \infty)$ with $f(M) \geqslant f(0)$, the following inequality holds*

$$\mathbb{E}[f(X)] \leqslant \lambda f(M) + (1 - \lambda) f(0).$$

Proof. Consider another random variable $Y := \frac{X}{M} f(M) + (1 - \frac{X}{M}) f(0)$. From the convexity of f we have $f(X) \leqslant Y$. Therefore

$$\begin{aligned}\mathbb{E}[f(X)] &\leqslant \frac{\mathbb{E}[X]}{M} \cdot f(M) + f(0) - \frac{\mathbb{E}[X]}{M} \cdot f(0) \leqslant f(0) + (f(M) - f(0)) \cdot \frac{\mathbb{E}[X]}{M} \\ &\leqslant \lambda f(M) + (1 - \lambda) f(0),\end{aligned}$$

as desired. ◀

2.2 Preliminaries

Let $H = (V, E)$ be a hypergraph and let k denote the minimum size of an edge in H. For any $j \geqslant k$ we define

$$q_j := \sum_{e \in E, |e| = j} 2^{-j+1},$$

which is the expected number of monochromatic edges of size j when the color of each vertex is sampled uniformly at random. Let $q := q(H)$ and observe that $q = \sum_{j \geqslant k} q_j$.

We aim to prove that if $q = O(\log k)$ then the hypergraph is two colorable. In order to do that, we describe a random coloring procedure and with a careful analysis we bound the probability that a fixed edge is monochromatic after the procedure finishes. The obtained bound allows us to conclude that the expected number of monochromatic edges after the procedure finishes is smaller than one. Thus, the hypergraph is two colorable.

2.3 The coloring procedure

Our algorithm is based on the *random greedy coloring* and it works in two phases. In the first one, for every vertex v we sample uniformly and independently an *initial color* $ic(v)$ and a *weight* $w(v) \in (0, 1)$. Then, an edge in which all vertices get the same initial color is called *initially monochromatic*. For an edge e, the *heaviest* vertex in e is the one with maximum weight among vertices in e (we assume that no two vertices have the same weight). We define the weight $w(e)$ of e to be the weight of the heaviest vertex in e (i.e. $w(e) = \max_{v \in e} w(v)$). The procedure is defined as follows:

- **Phase 1: Initial coloring**
  For every vertex v sample independently the following two values:
  - $ic(v)$ – the *initial color* of v: blue or red, each with probability $\frac{1}{2}$,
  - $w(v)$ – the *weight* of v, sampled uniformly at random from the real interval $(0, 1)$.
- **Phase 2: Recoloring**
  Arrange the vertices in the order of increasing weights. Iterating over vertices in that order, for every vertex v, we assign $c(v)$ – the *color* of v according to the following rules:

- if v is the heaviest vertex of some initially monochromatic edge e, and for all other vertices w of e we have $c(w) = ic(w)$, we set $c(v)$ to be the color that is different than $ic(v)$,
- otherwise we put $c(v) = ic(v)$.

Observe that once the color $c(v)$ is assigned, it is never changed. We say that a vertex v is *recolored* if $c(v)$ is different than $ic(v)$. Note also that if v is the heaviest vertex in some initially monochromatic edge e, then, at the moment when we assign color $c(v)$ to v, the color of every other vertex in e is already defined. At that point, if none of the other vertices in e is recolored, we define $c(v)$ to be the color other than $ic(v)$ (i.e. we recolor v), and we say that e is a *reason* to recolor v. Note that there may be more than one reason to recolor v. On the other hand, if there is no reason to recolor v we simply assign $c(v) = ic(v)$. Observe that eventually every initially monochromatic edge gets one of the vertices recolored.

2.4 Main result

For a better exposition of the argument, we first prove a statement slightly weaker than Theorem 1. In Section 3.3 we give a proof of the following result about the coloring procedure.

▶ **Proposition 3.** *If $q = O(\frac{\log k}{\log\log k})$ then, for any edge e, the probability that all vertices in e are colored red does not exceed $\frac{1}{3q2^{|e|-1}}$.*

This immediately implies that the expected number of monochromatic edges is at most $2 \cdot \sum_{e\in E} \frac{1}{3\cdot q\cdot 2^{|e|-1}} = \frac{2}{3}$ and thus, not only H is two colorable but also that our coloring procedure succeeds with probability at least $\frac{1}{3}$. In Section 3.4 we introduce more technical details to the argument and improve the bound.

▶ **Proposition 4.** *If $q = O(\log k)$ then, for any edge e, the probability that all vertices in e are colored red does not exceed $\frac{1}{3q2^{|e|-1}}$.*

This immediately implies Theorem 1.

3 Analysis

3.1 Bad events

The proof focuses on bounding the probability that one, fixed edge becomes monochromatic red. Nevertheless, we want to first exclude some problematic but unlikely events from happening. The simplest example is that we don't want two different vertices to receive the same weight. The probability of this event is zero and we want to simply assume that it doesn't happen. To be more precise, we allow our coloring procedure to fail during the initial phase of coloring. We give a few different reasons to fail and we argue that the probability that any of those bad events happens is small. Then, for the rest of the proof, we assume that none of the bad events happens.

3.1.1 Event $\mathcal{A}$ – too many initially monochromatic edges

The expected number of initially monochromatic edges is q. For a constant $\alpha_{\mathcal{A}}$ (to be fixed later) let $\mathcal{A}$ denote the event that there are more than $\alpha_{\mathcal{A}} \cdot q$ initially monochromatic edges. Markov's inequality gives that $\Pr[\mathcal{A}] < 1/\alpha_{\mathcal{A}}$.

3.1.2 Event $\mathcal{B}$ – a light monochromatic edge

For a constant $\alpha_{\mathcal{B}}$ (to be fixed later) and every j we define

$$p_j := \frac{\ln(\alpha_{\mathcal{B}} q)}{j}.$$

An edge f of size j is *light* if it is initially monochromatic and the weight of every vertex in f is smaller than $1 - p_j$. The expected number of light monochromatic edges of size j is

$$q_j 2^{j-1} \cdot (1-p_j)^j \cdot 2^{-j+1} < q_j \cdot \exp(-p_j \cdot j) = \frac{q_j}{\alpha_{\mathcal{B}} q}.$$

Therefore, the expected total number of light edges (of any size) is at most $1/\alpha_{\mathcal{B}}$. Let $\mathcal{B}$ denote the event that there is a light monochromatic edge. Clearly $\Pr[\mathcal{B}] < 1/\alpha_{\mathcal{B}}$.

3.1.3 Event $\mathcal{C}$ – too many almost monochromatic edges

An edge f is *almost monochromatic* if there is a vertex $v \in f$ such that all vertices in $f - v$ have the same initial color (in particular, an initially monochromatic edge is also an almost monochromatic edge). With every almost monochromatic edge f we can injectively associate a *certifying pair* $(f, v) \in E \times V$ for which $v \in f$ and $f - v$ is initially monochromatic.

Let Q_j be a random variable that denotes the number of almost monochromatic edges of size j. Since the number of such edges cannot exceed the number of certifying pairs associated with edges of size j, we get $\mathbb{E}[Q_j] \leqslant q_j 2^{j-1} \cdot j \cdot 2^{-j+2} = 2j \cdot q_j$. We define random variable

$$Y := \sum_j \frac{Q_j}{j},$$

and get that $\mathbb{E}[Y] \leqslant 2q$. Let $\mathcal{C}$ denote the event that $Y > \alpha_{\mathcal{C}} q$. Markov's inequality gives $\Pr[\mathcal{C}] < 2/\alpha_{\mathcal{C}}$.

For any fixed $\varepsilon > 0$ we can choose constants $\alpha_{\mathcal{A}}, \alpha_{\mathcal{B}}, \alpha_{\mathcal{C}}$ so that $1/\alpha_{\mathcal{A}} + 1/\alpha_{\mathcal{B}} + 2/\alpha_{\mathcal{C}} < \varepsilon$. Denote by $\mathcal{G}$ the intersection $\overline{\mathcal{A}} \cap \overline{\mathcal{B}} \cap \overline{\mathcal{C}}$ and observe that $\Pr[\mathcal{G}] > 1 - \varepsilon$. That is, with arbitrarily high probability none of the bad events happens. For any event $\mathcal{V}$, we denote $\Pr[\mathcal{V} \cap \mathcal{G}]$ by $\Pr_{\mathcal{G}}[\mathcal{V}]$ and similarly by $\Pr_{\mathcal{G}}[\mathcal{V}|\mathcal{C}]$ we mean $\Pr[\mathcal{V} \cap \mathcal{G}|\mathcal{C}]$.

3.2 e-focused coloring

For the rest of this section and the next section we fix an arbitrary edge e in E. Let s denote the size of e. The event "e *becomes red*" denotes the situation that all vertices in e are colored red by the coloring procedure. First observation is that if e is initially monochromatic red, then at least one vertex in e gets recolored and e can't become red in the end. Thus, if e becomes red then e contains some initially blue vertices and each of them gets recolored. In particular, every initially blue vertex in e is the heaviest vertex in some initially monochromatic blue edge. Additionally, it needs to happen that none of the initially red vertices in e gets recolored, but this condition seems impossible to use.

Taking into account the bad events we aim to prove that for a proper q we have:

$$\Pr_{\mathcal{G}}[e \text{ becomes red}] < \frac{1}{3} \cdot \frac{1}{q 2^{s-1}}.$$

3.2.1 The threat hypergraph

In what follows we try to understand better which initially blue vertices in e are recolored to red. The important observation is that if edges f and e have more than one vertex in common and f is a reason to recolor any of the common vertices, then e does not become red. To see that, let v be the heaviest vertex in f, and let w be any vertex in $f \cap e$ other than v. If f is a reason to recolor v then f is initially monochromatic blue and w is not recolored. Thus, w retains the initial blue color, and edge e does not become red.

This motivates the following construction of the *threat hypergraph* H_e. We define the vertex set of H_e to be $V \setminus e$. For each edge f in E that has exactly one common vertex with e (i.e., $|f \cap e| = 1$), let $f_e = f \setminus e$. We define the edge set of H_e to be $\{f_e : f \in E, |f \cap e| = 1\}$. Observe that for different edges $f \neq f'$ in E it might happen that $f_e = f'_e$. Thus, H_e is a multihypergraph. For each edge f_e of H_e we call f to be the *extension edge* of f_e and we call the only vertex in $f \cap e$ to be the *extension vertex* of f_e.

For the sake of our analysis, we reveal the outcomes of the random experiments used in the coloring procedure in four steps. In the first step we reveal the initial colors of the vertices in H_e. In the second step, we reveal the initial colors of the vertices in e. Then, we reveal the weights of vertices in H_e. Finally, we reveal the weights of the vertices in e. It is crucial to understand that this does not influence the coloring procedure in any way.

After the first step, some edges in H_e are monochromatic blue. For every such edge f_e, let v be the extension vertex of f_e, and we say that v is *endangered* by f_e. Observe that if e is to become red, then among vertices in e, only the endangered ones can be recolored from blue to red. For every endangered vertex v in e we define the *severity* of v to be the minimum size $|f|$ of an edge such that v is endangered by f_e. We define $\mathcal{R}^e_j$ to be the set of all vertices in e that are endangered and with severity j. Let $R^e_j := |\mathcal{R}^e_j|$. Note that both R^e_j and $\mathcal{R}^e_j$ are random variables which are determined after the first step (i.e., by the initial colors of the vertices in H_e).

Thus, a necessary condition for e to become red is that in the second step only the endangered vertices get initial color blue. Consider an endangered vertex with severity j which is initially blue and which is to become red. There is an edge f_e that endangers v, and v becomes the heaviest vertex in the extension of f_e. In particular, since the size of f is at least j, the weight of v (revealed in the fourth step) has to be at least $1 - p_j$. Otherwise, the edge f is a light monochromatic edge and bad event $\mathcal{B}$ happens.

Observe that there are no more vertices recolored than there are initially monochromatic edges. As we assume that bad event $\mathcal{A}$ does not happen, there are at most $\alpha_{\mathcal{A}} q$ vertices recolored in total. Let us sum up the observed necessary conditions for the edge e to become red:

1. at least one and at most $\alpha_{\mathcal{A}} q$ vertices in e are initially blue,
2. every initially blue vertex v in e is endangered. If severity of v is j then $w(v) \geqslant 1 - p_j$.

We use these conditions to obtain an upper bound on the probability of e becoming red.

3.3 Simple bound

We define, mainly for technical convenience, a random variable

$$X := \sum_j R^e_j \cdot p_j.$$

Observe that X is determined after the first step and that X takes only a finite number of possible values. For the rest of this section whenever we condition on event $X = x$ we always assume that the value x is such that $\Pr[X = x] > 0$. Recall that s is the size of e. The bound will follow from the following result:

▶ **Proposition 5.**

$$\Pr_{\mathcal{G}}[e \text{ becomes red } | X = x] < \frac{\exp(x) - 1}{2^s}.$$

Proof. Assume that we are after the first step and the values of variables $\mathcal{R}^e_j$, R^e_j, and X are determined. For each $j \geqslant k$, let $r_j := R^e_j$. With this assumption, we compute the probability of e becoming red. We claim that

$$\Pr_{\mathcal{G}}[e \text{ becomes red } | \text{the first step}] \leqslant \frac{1}{2^{s-\sum_j r_j}} \sum_{1 \leqslant c_k + c_{k+1} + \ldots \leqslant \alpha_{\mathcal{A}} q} \prod_j \binom{r_j}{c_j} \left(\frac{p_j}{2}\right)^{c_j} \left(\frac{1}{2}\right)^{r_j - c_j}.$$

The first factor corresponds to the not endangered vertices in e – each of them needs to be initially colored red. The sum spans over the values $c_k, c_{k+1}, \ldots$, where c_j corresponds to the number of initially blue vertices in R^e_j. There are exactly $\sum_j c_j$ initially blue elements and we know that this number is at least 1 and at most $\alpha_{\mathcal{A}} q$.

Once the number of initially blue elements in each $\mathcal{R}^e_j$ is fixed, there are $\binom{r_j}{c_j}$ possibilities to choose these elements from $\mathcal{R}^e_j$. Finally, all the chosen elements have to be initially colored blue and their weight has to be at least $1 - p_j$. The remaining elements of $\mathcal{R}^e_j$ have to be initially colored red.

Observe that the expression depends not on a particular result of the first phase, but rather only on the values of R^e_j. We use the fact that $\binom{r_j}{c_j} \leqslant \frac{r_j^{c_j}}{c_j!}$, rearrange the terms, and obtain:

$$\begin{aligned}\Pr_{\mathcal{G}}[e \text{ becomes red } | (R^e_j = r_j)_{j \geqslant k}] &\leqslant \frac{1}{2^s} \sum_{c=1}^{\alpha_{\mathcal{A}} q} \sum_{c_k + c_{k+1} + \ldots = c} \prod_j \binom{r_j}{c_j} p_j^{c_j} \\ &\leqslant \frac{1}{2^s} \sum_{c=1}^{\alpha_{\mathcal{A}} q} \frac{1}{c!} \left(\sum_j r_j \cdot p_j \right)^c.\end{aligned}$$

Let $x := \sum_j r_j \cdot p_j$, recall that $X = \sum_j R^e_j \cdot p_j$, and observe that the last expression depends not on the particular values of R^e_j, but rather only on the value of X.

$$\Pr_{\mathcal{G}}[e \text{ becomes red } | X = x] \leqslant \frac{1}{2^s} \sum_{c=1}^{\alpha_{\mathcal{A}} q} \frac{x^c}{c!} \tag{1}$$

$$< \frac{\exp(x) - 1}{2^s}. \tag{2}$$

◀

Proof of Proposition 3. Recall that the random variable Q_j denotes the number of almost monochromatic edges of size j, while R^e_j is the number of endangered vertices in e with severity j. For every such vertex v we have an initially blue edge f in H_e for which v is the extension vertex. Since the extension edge of f is almost blue we obtain that $R^e_j \leqslant Q_j$. This implies:

$$X = \sum_j R^e_j \cdot p_j = \ln(\alpha_{\mathcal{B}} q) \cdot \sum_j \frac{R^e_j}{j} \leqslant \ln(\alpha_{\mathcal{B}} q) \cdot \sum_j \frac{Q_j}{j} = \ln(\alpha_{\mathcal{B}} q) \cdot Y. \tag{3}$$

Therefore $X \leqslant \ln(\alpha_{\mathcal{B}} q) \cdot \alpha_{\mathcal{C}} q$ unless bad event $\mathcal{C}$ happens. We now have:

$$\begin{aligned}\Pr_{\mathcal{G}}[e \text{ becomes red}] &= \sum_{x \leqslant \ln(\alpha_{\mathcal{B}} q) \cdot \alpha_{\mathcal{C}} q} \Pr[X = x] \cdot \Pr_{\mathcal{G}}[e \text{ becomes red } | X = x] \\ &< \sum_{x \leqslant \ln(\alpha_{\mathcal{B}} q) \cdot \alpha_{\mathcal{C}} q} \Pr[X = x] \cdot \frac{\exp(x) - 1}{2^s} \\ &\leqslant \frac{1}{2^s} \cdot \mathbb{E}\left[\exp(X) - 1 \middle| X \leqslant \ln(\alpha_{\mathcal{B}} q) \cdot \alpha_{\mathcal{C}} q\right].\end{aligned}$$

Inequality (3) also yields

$$\mathbb{E}[X] \leqslant \sum_j q_j \cdot p_j \leqslant \frac{q \ln(\alpha_{\mathcal{B}} q)}{k}.$$

We apply Lemma 2 for $f(x) = \exp(x) - 1$, $M = \alpha_{\mathcal{C}} q \ln(\alpha_{\mathcal{B}} q)$ and $\lambda = \frac{1}{\alpha_{\mathcal{C}} k}$, and obtain:

$$\mathbb{E}[\exp(X) - 1] \leqslant \frac{\exp(\ln(\alpha_{\mathcal{B}} q) \cdot \alpha_{\mathcal{C}} q)}{\alpha_{\mathcal{C}} k}.$$

Hence

$$\Pr_{\mathcal{G}}[e \text{ becomes red}] < \frac{1}{2^s} \cdot \frac{\exp(\ln(\alpha_{\mathcal{B}} q) \cdot \alpha_{\mathcal{C}} q)}{\alpha_{\mathcal{C}} k}.$$

Let $\alpha > \max\{\alpha_{\mathcal{B}}, \alpha_{\mathcal{C}}\}$. Now, suppose that:

$$q \leqslant \frac{1}{\alpha} \cdot \frac{\ln k}{\ln \ln k}.$$

For k large enough, $\ln(\alpha_{\mathcal{B}} q) \leqslant \ln \ln(k)$ which yields:

$$\frac{\exp(\ln(\alpha_{\mathcal{B}} q) \cdot \alpha_{\mathcal{C}} q)}{\alpha_{\mathcal{C}} k} \leqslant \frac{1}{\alpha_{\mathcal{C}} k} \cdot \exp\left(\frac{\alpha_{\mathcal{C}} \ln k}{\alpha \ln \ln k} \cdot \ln \ln k\right) \leqslant \frac{k^{(\alpha_{\mathcal{C}}/\alpha)-1}}{\alpha_{\mathcal{C}}}.$$

For k large enough, the last term is less than $\frac{1}{6q}$, which implies:

$$\Pr_{\mathcal{G}}[e \text{ becomes red}] < \frac{1}{3q \cdot 2^{s-1}}$$

and completes the proof. ◀

An astute reader may have realised that we did not use bad event $\mathcal{A}$ in any essential way. Currently, the only reason to introduce $\mathcal{A}$ is that it makes the proof slightly easier. We could, however, use $\mathcal{A}$ to improve bound (2) for the values of x greater than q, leading to a slightly better condition $q = O(\frac{\log k}{\log \log \log k})$. We do not elaborate on that since the argument in Section 3.4 already gives an even better result.

3.4 Improved bound

In order to obtain an improved bound we introduce one more bad event.

3.4.1 Event $\mathcal{D}$ – large second weight deficit

For every edge f in E which is initially monochromatic, we define its *second weight deficit* as $d_2(f) := (|f|+1)\cdot(1-w_2(f))$, where $w_2(f)$ is the weight of the second heaviest vertex in f. For an edge f that is not initially monochromatic, $d_2(f)$ is defined to be 0.

Note that, conditioned on f being initially monochromatic, the variable $1-w_2(f)$ has mean $\frac{2}{|f|+1}$. In particular $\mathbb{E}[d_2(f)|\ f \text{ is monochromatic}] = 2$ and hence $\mathbb{E}[d_2(f)] = \frac{2}{2^{|f|-1}}$. Let $D_2 := \sum_{f\in E} d_2(f)$ and observe that we have

$$\mathbb{E}[D_2] = \sum_j 2q_j = 2q.$$

Event $\mathcal{D}$ is defined as $D_2 > \alpha_{\mathcal{D}} q$. By Markov's inequality, we get $\Pr[\mathcal{D}] < 2/\alpha_{\mathcal{D}}$ and we can chose $\alpha_{\mathcal{D}}$ so that this probability is arbitrarily small.

3.4.2 Analysis

In the first step of e-focused coloring we reveal the initial colors of all the vertices from $V\setminus e$. This step determines the endangered vertices in e – we denote their set by $\mathcal{R}$. For every value of $c = 1, 2, \dots \alpha_{\mathcal{A}} q$ and every c-subset $S = \{v_1, \dots, v_c\}$ of $\mathcal{R}$ we consider an event that S contains exactly the vertices in e which become recolored. Thus, these are the only initially blue vertices in e. Once we fix the subset S, the probability that S is the set of initially blue vertices in e is precisely 2^{-s}. This event is determined after the second step of e-focused coloring – when the initial colors of vertices in e are revealed. In order to be recolored, every vertex v_j must receive a weight that makes it heavier than some edge that endangers it. Let us reveal the weights of the vertices in $V\setminus e$ (third step of e-focused coloring). The vertex v_j is endangered by some edges $f^1_{v_j}, \dots, f^t_{v_j}$ of H_e, and let f_{v_j} be the lightest of these edges (i.e. the edge whose heaviest vertex is the lightest among the heaviest vertices of $f^1_{v_j}, \dots, f^t_{v_j}$). Clearly in order for vertex v_j to be recolored, it has to get a weight greater than $w(f_{v_j})$ – this happens with probability $1-w(f_{v_j})$. We choose a parametrization that takes into account the size of f_{v_j} and denote the value $1-w(f_{v_j})$ by $\frac{\delta_j}{|f_{v_j}|+2}$. Now, conditioned on the result of the first three steps, the probability that all vertices $\{v_1, \dots, v_c\}$ are heavy enough is

$$\prod_{j=1}^{c} \frac{\delta_j}{|f_{v_j}|+2} < \prod_{j=1}^{c} \frac{\delta_j}{|f_{v_j}|+1}. \tag{4}$$

The edge f_{v_j} together with v_j forms an edge of H, which we denote by h_{v_j}. Although the value of $d_2(h_{v_j})$ is not determined until we reveal the weight of v_j (in the fourth step), we already know at this point that $\delta_j \leqslant d_2(h_{v_j})$ (it becomes an equality when v_j becomes the heaviest vertex in h_{v_j}). Assuming the bad event $\mathcal{D}$ does not happen, we have $\sum_{j=1}^{c} \delta_j \leqslant \alpha_{\mathcal{D}} q$. Using the AM-GM inequality we deduce that $\prod_{j=1}^{c} \delta_j \leqslant \left(\frac{\alpha_{\mathcal{D}} q}{c}\right)^c$, which bounds the value of (4):

$$\prod_{j=1}^{c} \frac{\delta_j}{|f_{v_j}|+1} \leqslant \left(\frac{\alpha_{\mathcal{D}} q}{c}\right)^c \prod_{j=1}^{c} \frac{1}{|f_{v_j}|+1}.$$

Summing over all c-subsets of $\mathcal{R}$ we get that the probability that some c-subset contains all initially blue vertices in e and they are all recolored does not exceed

$$\sum_{S\in\binom{\mathcal{R}}{c}} \frac{1}{2^s}\left(\frac{\alpha_{\mathcal{D}} q}{c}\right)^c \prod_{v\in S} \frac{1}{|f_v|+1} \leqslant \frac{1}{2^s}\left(\frac{\alpha_{\mathcal{D}} q}{c}\right)^c \frac{1}{c!}\left(\sum_{v\in\mathcal{R}} \frac{1}{|f_v|+1}\right)^c.$$

Define random variable

$$Y_e := \sum_{f_e \text{ in } H_e,\ f_e \text{ is blue}} \frac{1}{|f_e|+1},$$

which gets determined after the first step of e-focused coloring. For each endangered vertex v in e, all the edges, including the lightest one, that endanger v are blue and thus are taken into the sum defining Y_e. Therefore, $Y_e \geqslant \sum_{v\in\mathcal{R}} \frac{1}{|f_v|+1}$. On the other hand, the extension edge of every blue edge in H_e is an almost monochromatic edge in H. As Y counts the number of almost monochromatic edges in H, we get $Y_e \leqslant Y \leqslant \alpha_{\mathcal{C}} q$ unless bad event $\mathcal{C}$ happens. We can also bound the expected value of Y_e:

$$\mathbb{E}[Y_e] = \sum_{f\in H_e} \frac{2^{-|f|}}{|f|+1} < \frac{1}{k} \sum_{f'\in H} 2^{-|f'|+1} = \frac{q}{k}.$$

Note that Y_e takes only a finite number of possible values. For any value y such that $\Pr[Y_e = y] > 0$, we get the following bound:

$$\begin{aligned}
\Pr_{\mathcal{G}}[e \text{ becomes red } | Y_e = y] &\leqslant \frac{1}{2^s} \sum_{c=1}^{\alpha_{\mathcal{A}} q} \left(\frac{\alpha_{\mathcal{D}} q}{c}\right)^c \frac{y^c}{c!} \\
&\leqslant \frac{1}{2^s} \sum_{c=1}^{\alpha_{\mathcal{A}} q} \frac{(\alpha_{\mathcal{D}} q y)^c}{c! \cdot c^c} \\
&\leqslant \frac{1}{2^s} \sum_{c=1}^{\alpha_{\mathcal{A}} q} \frac{(2\alpha_{\mathcal{D}} q y)^c}{(2c)!}
\end{aligned}$$

as $\frac{(2c)!}{2^c} \leqslant \frac{c!\cdot(2c)^c}{2^c} = c! \cdot c^c$. For any x, we have $\sum_{c=0}^{\infty} \frac{x^{2c}}{(2c)!} = \frac{\exp(x)+\exp(-x)}{2} = \cosh(x)$. Therefore

$$\Pr_{\mathcal{G}}[e \text{ becomes red } | Y_e = y] \leqslant \frac{1}{2^s}(\cosh\left(\sqrt{2\alpha_{\mathcal{D}} q y}\right) - 1),$$

and

$$\Pr_{\mathcal{G}}[e \text{ becomes red}] \leqslant \mathbb{E}\left[\frac{1}{2^s}(\cosh\left(\sqrt{2\alpha_{\mathcal{D}} q Y_e}\right) - 1)\middle| Y_e \leqslant \alpha_{\mathcal{C}} q\right].$$

Observe that for any $a > 0$, the function $\cosh(a\sqrt{x})$ is convex and increasing in $[0, \infty)$. Therefore, we apply Lemma 2 for $f(x) = \cosh(\sqrt{2\alpha_{\mathcal{D}} q x}) - 1$, $M = \alpha_{\mathcal{C}} q$ and $\lambda = \frac{1}{\alpha_{\mathcal{C}} k}$, and obtain:

$$\Pr_{\mathcal{G}}[e \text{ becomes red}] \leqslant \frac{1}{2^s}\frac{1}{\alpha_{\mathcal{C}} k}(\cosh\left(\sqrt{2\alpha_{\mathcal{D}}\alpha_{\mathcal{C}} q q}\right) - 1) \leqslant \frac{1}{\alpha_{\mathcal{C}} k 2^s} \exp\left(\sqrt{2\alpha_{\mathcal{D}}\alpha_{\mathcal{C}}} \cdot q\right).$$

The obtained value is smaller than $\frac{1}{3q2^{s-1}}$ whenever

$$\frac{3q \exp\left(\sqrt{2\alpha_{\mathcal{D}}\alpha_{\mathcal{C}}} \cdot q\right)}{2\alpha_{\mathcal{C}} k} \leqslant 1.$$

The last inequality is easily seen to hold for $q \leqslant \frac{0.9}{\sqrt{2\alpha_{\mathcal{D}}\alpha_{\mathcal{C}}}} \ln k$ and all large enough k.

4 Remarks

4.1 Bounded maximal size

We can derive better bounds when the size of the maximum edge is not much larger than k. Suppose that $\max_{e\in E}|e| \leqslant K$. We apply the proof strategy from [2] and analyze the random greedy coloring procedure (i.e. we arrange the vertices in random order and color consecutive vertices blue if this does not create a monochromatic edge, otherwise we color it red). As a technical convenience, instead of sampling a random ordering of the vertices, for every vertex we choose uniformly a weight from the real interval $(0,1)$. We color vertices greedily in the order of increasing weights. We choose (with foresight) parameter $p := \ln(4q)/k$. An edge is called *light* if the weight of its heaviest vertex is at most $(1-p)/2$. Similarly an edge f is *heavy* if every verex in f has weight at least $(1+p)/2$. The probability that there exists a light edge is bounded by the expected number of such:

$$\sum_{f\in E}\left(\frac{1-p}{2}\right)^{|f|} \leqslant (1-p)^k \cdot q.$$

The same bound holds for *heavy* edges. It is easy to see that in order for the procedure to fail there must exist a pair of edges f_1, f_2 such that the heaviest vertex of f_1 is the lightest vertex of f_2. Such a pair is called *conflicting*. Therefore for the procedure to fail it is necessary that either there exists a conflicting pair f_1, f_2 for which the weight of the unique common vertex belongs to $((1-p)/2, (1+p)/2)$ or there exists a light or heavy edge. The expected number of such conflicting pairs is at most

$$\begin{aligned}
&\sum_{f_1,f_2\in E}\int_{-p/2}^{p/2}\left(\frac{1}{2}+x\right)^{|f_1|-1}\left(\frac{1}{2}-x\right)^{|f_2|-1}dx\\
&\leqslant \sum_{f_1,f_2\in E} 2^{-|f_1|-|f_2|+2}\cdot p\cdot \max_{x\in(-p/2,p/2)}(1+2x)^{|f_1|-1}(1-2x)^{|f_2|-1}\\
&\leqslant p\cdot\sum_{f_1,f_2\in E} 2^{-|f_1|-|f_2|+2}\cdot \max_{x\in(-p/2,p/2)}(1+2x)^{|f_1|-|f_2|}\\
&= p\cdot\sum_{f_1,f_2\in E} 2^{-|f_1|-|f_2|+2}\cdot (1+p)^{|f_1|-|f_2|}\\
&\leqslant p\cdot(1+p)^{K-k}\sum_{f_1,f_2\in E} 2^{-|f_1|-|f_2|+2} = p(1+p)^{K-k}q^2.
\end{aligned}$$

Altogether the probability of failure is at most

$$p(1+p)^{K-k}q^2 + 2(1-p)^k q \sim pq^2\exp(p(K-k)) + 2q\exp(-pk).$$

Plugging in the value of p we get

$$\frac{\ln(4q)q^2(4q)^{K/k-1}}{k} + 1/2 \leqslant \frac{\ln(k)(4q)^{K/k+1}}{k} + 1/2$$

where we additionally assumed that $q \leqslant k$. As long as this value is below 1 we can be sure that random greedy coloring strategy succeeds with positive probability. For $k = K$ we recover the result of [2]. When K is bounded by a linear function of k, e.g. $K \leqslant \alpha k$ it is sufficient that q does not exceed

$$\frac{1}{5}\left(\frac{k}{\ln(k)}\right)^{\frac{1}{\alpha+1}}$$

The resulting bound for q starts to be worse than the one from Theorem 1 when K is roughly of the order $k\log(k)$.

4.2 Uniform case

It is instructive to observe how our analysis works for uniform hypergraphs. We focus on modifications in the proof of our simple bound, since the ideas used for the improved bound do not help in the uniform case. Using an obvious bound $R_k^e \leqslant k$, we improve inequality (3) to $X \leqslant \ln(\alpha_{\mathcal{B}} q)$. Then we apply Lemma 2 with $M = \ln(\alpha_{\mathcal{B}} q)$ and $\lambda = \frac{q}{k}$ obtaining

$$\Pr_{\mathcal{G}}[e \text{ becomes red}] < 2^{-k}\frac{q}{k}\exp(\ln(\alpha_{\mathcal{B}} q)) = 2^{-k}\frac{q^{1+\alpha_{\mathcal{B}}}}{k}.$$

Since in this case the only bad event that we use is $\mathcal{B}$, we can afford to set $\alpha_{\mathcal{B}} = 1+\varepsilon$, for any small $\varepsilon > 0$. We get that $\mathcal{B}$ does not happen with probability at least $\frac{\varepsilon}{1+\varepsilon}$. Then

$$\Pr_{\mathcal{G}}[e \text{ becomes red}] < 2^{-k}\frac{q^{2+\varepsilon}}{k}$$

and in order for this value to be at most $\frac{1}{2^k q}\cdot\frac{\varepsilon}{1+\varepsilon}$ it suffices that

$$q \leqslant k^{\frac{1}{3+\varepsilon}}\cdot\left(\frac{\varepsilon}{1+\varepsilon}\right)^{\frac{1}{3+\varepsilon}}.$$

This way we obtain a result analogous to that of Beck from [1] (i.e. $m(k) \geqslant k^{1/3-o(1)}2^k$). Incorporating the ideas from [10] or [2] that allowed to derive a bound $m(k) = \Omega(\sqrt{k/\log(k)})\cdot 2^k$ does not bring any significant improvement of our main result.

4.3 Hypergraphs with random-like characteristics

The weakest points of our analysis are the places where we apply Lemma 2. The lemma works for any bounded non-negative random variable X. It is clear from the bound that the worst case distribution of X is the one that assumes only values 0 and M. The variables for which we apply the lemma are related to the numbers of initially monochromatic edges in hypergraphs H_e. If these variables exhibit sufficiently strong concentration around their mean (like in the case of random hypergraphs) we may get a much stronger bound than the one of Lemma 2 and obtain results that are much closer to the case of uniform hypergaphs.

References

1 József Beck. On 3-chromatic hypergraphs. *Discrete Mathematics*, 24(2):127–137, 1978. doi:10.1016/0012-365X(78)90191-7.

2 Danila D. Cherkashin and Jakub Kozik. A note on random greedy coloring of uniform hypergraphs. *Random Structures & Algorithms*, 47(3):407–413, 2015. doi:10.1002/rsa.20556.

3 Paul Erdős. On a combinatorial problem. *Nordisk Matematisk Tidskrift*, 11:5–10, 40, 1963.

4 Paul Erdős. On a combinatorial problem. II. *Acta Mathematica Academiae Scientiarum Hungaricae*, 15:445–447, 1964.

5 Paul Erdős and András Hajnal. On a property of families of sets. *Acta Mathematica Academiae Scientiarum Hungaricae*, 12:87–123, 1961.

6 Paul Erdős and László Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. In *Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to P. Erdős on his 60th birthday), Vol. II*, volume 10 of *Colloquia Mathematica Societatis János Bolyai*, pages 609–627. North-Holland, Amsterdam, 1975.

7 Heidi Gebauer. On the construction of 3-chromatic hypergraphs with few edges. *Journal of Combinatorial Theory. Series A*, 120(7):1483–1490, 2013. doi:10.1016/j.jcta.2013.04.007.

8 László Lovász. Coverings and coloring of hypergraphs. In *Proceedings of the Fourth Southeastern Conference on Combinatorics, Graph Theory, and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1973)*, pages 3–12, 1973.

9 Linyuan Lu. On a problem of Erdős and Lovász on coloring non-uniform hypergraphs, 2008. URL: http://people.math.sc.edu/lu/papers/propertyB.pdf.

10 Jaikumar Radhakrishnan and Aravind Srinivasan. Improved bounds and algorithms for hypergraph 2-coloring. *Random Structures & Algorithms*, 16(1):4–32, 2000. doi:10.1002/(SICI)1098-2418(200001)16:1<4::AID-RSA2>3.3.CO;2-U.

11 Dmitry A. Shabanov. Around Erdős-Lovász problem on colorings of non-uniform hypergraphs. *Discrete Mathematics*, 338(11):1976–1981, 2015. doi:10.1016/j.disc.2015.04.017.

Additive Non-Approximability of Chromatic Number in Proper Minor-Closed Classes

Zdeněk Dvořák[1]
Charles University, Malostranske namesti 25, 11800 Prague, Czech Republic
rakdver@iuuk.mff.cuni.cz
https://orcid.org/0000-0002-8308-9746

Ken-ichi Kawarabayashi[2]
National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
k_keniti@nii.ac.jp
https://orcid.org/0000-0001-6056-4287

Abstract

Robin Thomas asked whether for every proper minor-closed class $\mathcal{G}$, there exists a polynomial-time algorithm approximating the chromatic number of graphs from $\mathcal{G}$ up to a constant additive error independent on the class $\mathcal{G}$. We show this is not the case: unless P = NP, for every integer $k \geq 1$, there is no polynomial-time algorithm to color a K_{4k+1}-minor-free graph G using at most $\chi(G)+k-1$ colors. More generally, for every $k \geq 1$ and $1 \leq \beta \leq 4/3$, there is no polynomial-time algorithm to color a K_{4k+1}-minor-free graph G using less than $\beta\chi(G) + (4 - 3\beta)k$ colors. As far as we know, this is the first non-trivial non-approximability result regarding the chromatic number in proper minor-closed classes.

We also give somewhat weaker non-approximability bound for K_{4k+1}-minor-free graphs with no cliques of size 4. On the positive side, we present an additive approximation algorithm whose error depends on the apex number of the forbidden minor, and an algorithm with additive error 6 under the additional assumption that the graph has no 4-cycles.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases non-approximability, chromatic number, minor-closed classes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.47

1 Introduction

The problem of determining the chromatic number of a graph, or even of just deciding whether a graph is colorable using a fixed number $c \geq 3$ of colors, is NP-complete [7], and thus it cannot be solved in polynomial time unless P = NP. Even the approximation version of the problem is hard: for every $\varepsilon > 0$, Zuckerman [16] proved that unless P = NP, there exists no polynomial-time algorithm approximating the chromatic number of an n-vertex graph within multiplicative factor $n^{1-\varepsilon}$.

There are more restricted settings in which the graph coloring problem becomes more tractable. For example, the well-known Four Color Theorem implies that deciding c-colorability of a planar graph is trivial for any $c \geq 4$; still, 3-colorability of planar graphs is NP-complete [7]. From the approximation perspective, this implies that chromatic number of planar graphs can be approximated in polynomial time up to multiplicative factor of 4/3 (but not better), and additively up to 1.

[1] Supported by project 17-04611S (Ramsey-like aspects of graph coloring) of Czech Science Foundation.
[2] Supported by JST ERATO Grant Number JPMJER1305, Japan.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 47; pp. 47:1–47:12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Let Σ be a fixed surface, and consider a graph G embedded in Σ. If G is large enough, Euler's formula implies that its average degree is less than 7, and thus G contains a vertex v of degree at most 6. Removing v and repeating this observation, we conclude that the vertex set of G can be partitioned in parts A and B such that A has bounded size (depending only on Σ) and $G[B]$ is 6-degenerate. We can now by brute force find an optimal coloring of $G[A]$ and color $G[B]$ greedily using at most 7 additional colors, thus obtaining a proper coloring of G using at most $\chi(G[A]) + 7 \leq \chi(G) + 7$ colors. That is, for any fixed surface Σ, there exists a linear-time algorithm to approximate chromatic number of graphs embedded in Σ with additive error at most 7, independent of the surface (this additive error can be improved to 2 using the fact that c-colorability of embedded graphs can be decided in polynomial-time for $c \geq 5$ based on a deep result of Thomassen [14]).

If a graph can be drawn in a given surface, all its minors can be drawn there as well. Hence, it is natural to also consider the coloring problem in the more general setting of proper minor-closed classes. Motivated by the simple algorithm for embedded graphs from the previous paragraph (and the minor structure theorem [12], which relates proper minor-closed classes to embedded graphs), Robin Thomas [13] conjectured that such an additive approximation algorithm exists for graphs from any proper minor-closed class.

▶ **Conjecture 1.** *For some fixed constant $\alpha \geq 0$, the following holds: for every proper minor-closed class $\mathcal{G}$, there exists a polynomial-time algorithm taking as an input a graph $G \in \mathcal{G}$ and returning an integer c such that $\chi(G) \leq c \leq \chi(G) + \alpha$.*

The fact that we do not allow α depend on the class $\mathcal{G}$ is important: any K_k-minor-free graph is $O(k\sqrt{\log k})$-colorable [10], and thus its chromatic number can be trivially approximated with additive error $O(k\sqrt{\log k})$.

Since it is easy to decide whether $\chi(G) < 3$, Conjecture 1 would imply that chromatic number in proper minor-closed classes can be approximated up to a fixed multiplicative factor (at most $1 + \alpha/3$). And indeed, this weaker statement holds: As shown by DeVos et al. [4] and algorithmically by Demaine et al. [2], for every proper minor-closed class $\mathcal{G}$, there exists a constant $\gamma_{\mathcal{G}}$ such that the vertex set of any graph $G \in \mathcal{G}$ can be partitioned in polynomial time into two parts A and B with both $G[A]$ and $G[B]$ having tree-width at most $\gamma_{\mathcal{G}}$. Consequently, $\chi(G[A])$ and $\chi(G[B])$ can be determined exactly in linear time [1], and we can color $G[A]$ and $G[B]$ using disjoint sets of colors, obtaining a coloring of G using at most $\chi(G[A]) + \chi(G[B]) \leq 2\chi(G)$ colors.

In the light of this evidence, Conjecture 1 seems quite plausible. Kawarabayashi et al. [9] conjectured that it actually holds even in a stronger form, in the list coloring setting. As our main result, we show that Conjecture 1 is actually false, even if we allow further multiplicative error by a factor smaller than 4/3. A graph F is *t-apex* if there exists a set X of at most t vertices of F such that $F - X$ is planar.

▶ **Theorem 2.** *Let k_0 be a positive integer, let F be a $(4k_0 - 3)$-connected graph that is not $(4k_0 - 4)$-apex, and let $1 \leq \beta \leq 4/3$ be a real number. Unless $P = NP$, there is no polynomial-time algorithm taking as an input an F-minor-free graph G and returning an integer c such that $\chi(G) \leq c < \beta\chi(G) + (4 - 3\beta)k_0$.*

In particular, in the special case of $\beta = 1$ and F being a clique, we obtain the following.

▶ **Corollary 3.** *Let k_0 be a positive integer. Unless $P = NP$, there is no polynomial-time algorithm taking as an input a K_{4k_0+1}-minor-free graph G and returning an integer c such that $\chi(G) \leq c \leq \chi(G) + k_0 - 1$.*

On the positive side, Kawarabayashi et al. [9] showed it is possible to approximate the chromatic number of K_k-minor free graphs in polynomial time additively up to $k-2$. We leave open the question whether a better additive approximation (of course above the bound $\approx k/4$ given by Corollary 3) is possible. Another positive result was given by Demaine et al. [3], who proved that if H is a 1-apex graph, then the chromatic number of H-minor-free graphs can be approximated additively up to 2. Let us also remark that if H is 0-apex (i.e., planar), then H-minor-free graphs have bounded tree-width [11], and thus their chromatic number can be determined exactly in linear time [1].

The *apex number* of a graph F is the minimum t such that F is t-apex. Theorem 2 shows that the additive error of any approximation algorithm for chromatic number of F-minor-free graphs must depend on the apex number of F. On the positive side, we show that such algorithm exists, generalizing the results from the previous paragraph.

▶ **Theorem 4.** *Let t be a positive integer and let H be a t-apex graph. There exists a polynomial-time algorithm taking as an input an H-minor-free graph G and returning an integer c such that $\chi(G) \leq c \leq \chi(G) + t + 3$.*

The construction we use to establish Theorem 2 results in graphs with large clique number (on the order of k_0). On the other hand, forbidding triangles makes the coloring problem for embedded graphs more tractable—all triangle-free planar graphs are 3-colorable [8] and there exists a linear-time algorithm to decide 3-colorability of a triangle-free graph embedded in any fixed surface [6]. It is natural to ask whether Conjecture 1 could not hold for triangle-free graphs, and this question is still open. On the negative side, we show that forbidding cliques of size 4 is not sufficient.

▶ **Theorem 5.** *Let β and d be real numbers such that $1 \leq \beta < 4/3$ and $d \geq 0$. Let $m = \lceil d/(4-3\beta) \rceil$. There exists a positive integer $k_0 = O\big(m^4 \log^2 m\big)$ such that the following holds. Let F be a $(4k_0-1)$-connected graph with at least $4k_0+8$ vertices that is not $(4k_0-4)$-apex. Unless $P = NP$, there is no polynomial-time algorithm taking as an input an F-minor-free graph G with $\omega(G) \leq 3$ and returning an integer c such that $\chi(G) \leq c < \beta\chi(G) + d$.*

In particular, in the special case of $\beta = 1$ and F being a complete graph, we get the following.

▶ **Corollary 6.** *For every positive integer k_0, there exists an integer $d = \Omega(k_0^{1/4}/\log^{1/2} k_0)$ as follows. Unless $P = NP$, there is no polynomial-time algorithm taking as an input a K_{4k_0+8}-minor-free graph G with $\omega(G) \leq 3$ and returning an integer c such that $\chi(G) \leq c \leq \chi(G) + d$.*

On the positive side, we offer the following small improvement to the additive error of Theorem 4.

▶ **Theorem 7.** *Let t be a positive integer and let H be a t-apex graph. There exists a polynomial-time algorithm taking as an input an H-minor-free graph G with no triangles and returning an integer c such that $\chi(G) \leq c \leq \chi(G) + \lceil (13t+172)/14 \rceil$.*

What about graphs of larger girth? It turns out that Conjecture 1 holds for graphs of girth at least 5, with $\alpha = 6$. Somewhat surprisingly, it is not even necessary to forbid triangles to obtain this result, just forbidden 4-cycles are sufficient. Indeed, we can show the following stronger result.

▶ **Theorem 8.** *Let $a \leq b$ be positive integers and let $\mathcal{G}$ be a proper minor-closed class of graphs. There exists a polynomial-time algorithm taking as an input a graph $G \in \mathcal{G}$ not containing $K_{a,b}$ as a subgraph and returning an integer c such that $\chi(G) \leq c \leq \chi(G) + a + 4$.*

Let us remark that the multiplicative 2-approximation algorithm of Demaine et al. [2] can be combined with the algorithms of Theorems 4, 7, and 8 by returning the minimum of their results. E.g., if H is a t-apex graph, then there is a polynomial-time algorithm coloring an H-minor-free graph G using at most $\min(2\chi(G), \chi(G)+t+3) \leq \beta\chi(G)+(2-\beta)(t+3)$ colors, for any β such that $1 \leq \beta \leq 2$; the combined multiplicative-additive non-approximability bounds of Theorems 2 and 5 are also of interest in this context.

The rest of the paper is organized as follows. In Section 2, we present a graph construction which we exploit to obtain the non-approximability results in Section 3. The approximation algorithms are presented in Section 4.

2 Tree-like product of graphs

Let G and H be graphs, and let $|V(G)| = n$ and $V(H) = \{u_1, \ldots, u_k\}$. Let $T_{n,k}$ denote the rooted tree of depth $k+1$ such that each vertex at depth at most k has precisely n children (the *depth* of the tree is the number of vertices of a longest path starting with its root, and the depth of a vertex x is the number of vertices of the path from the root to x; i.e., the root has depth 1). For each non-leaf vertex $x \in V(T_{n,k})$, let G_x be a distinct copy of the graph G and let θ_x be a bijection from $V(G_x)$ to the children of x in $T_{n,k}$. If $v \in V(G_x)$, y is a non-leaf vertex of the subtree of $T_{n,k}$ rooted in $\theta_x(v)$, and $z \in V(G_y)$, then we say that v is a *progenitor* of z. The *level* of v is defined to be the depth of x in $T_{n,k}$. Note that a vertex at level j has exactly one progenitor at level i for all positive $i < j$. The graph $T(G,H)$ is obtained from the disjoint union of the graphs G_x for non-leaf vertices $x \in V(T_{n,k})$ by, for each edge $u_iu_j \in E(H)$ with $i < j$, adding all edges from vertices of $T(G,H)$ at level j to their progenitors at level i. Note that the graph $T(G,H)$ depends on the ordering of the vertices of H, which we consider to be fixed arbitrarily.

Let Q_1 and Q_2 be graphs, containing cliques K_1 and K_2 of the same size k. A graph Q is a *clique-sum* of graphs Q_1 and Q_2 on cliques K_1 and K_2 if Q is obtained from Q_1 and Q_2 by identifying the two cliques and possibly removing some edges of the resulting clique. We use the following well-known observation: If F is a $(k+1)$-connected graph and F is a minor of Q, then F is also a minor of Q_1 or Q_2.

▶ **Lemma 9.** *Let G and H be graphs with $|V(G)| = n \geq 2$ and $V(H) = \{u_1, \ldots, u_k\}$. Let $q \leq k$ be the maximum integer such that $\{u_{k-q+1}, \ldots, u_k\}$ is an independent set in H. The graph $T(G,H)$ has $O(n^k)$ vertices and $\omega(T(G,H)) = \omega(G) + \omega(H) - 1$. Furthermore, if F is a minor of $T(G,H)$ and F is $(k-q+1)$-connected, then there exists a set $X \subseteq V(F)$ of size at most $k-q$ such that $F - X$ is a minor of G.*

Proof. The tree $T_{n,k}$ has $1 + n + n^2 + \ldots + n^{k-1} \leq 2n^{k-1}$ non-leaf vertices, and thus $|V(T(G,H))| \leq 2n^k$.

Consider a clique K in $T(G,H)$, and let v be a vertex of K of largest level. Let x be the vertex of $T_{n,k}$ such that $v \in V(G_x)$. Note that all vertices of $K \setminus V(G_x)$ are progenitors of v, and the vertices of H corresponding to their levels are pairwise adjacent. Consequently, $|K \setminus V(G_x)| \leq \omega(H) - 1$ and $|K \cap V(G_x)| \leq \omega(G)$. Therefore, each clique in $T(G,H)$ has size at most $\omega(G) + \omega(H) - 1$. A converse argument shows that cliques K_G in G and K_H in H give rise to a clique in $T(G,H)$ of size $|K_G| + |K_H| - 1$, implying that $\omega(T(G,H)) = \omega(G) + \omega(H) - 1$.

For $i = 0, \ldots, k-q$, let G_i denote the graph obtained from G by adding i universal vertices. Observe that $T(G,H)$ is obtained from copies of $G_0, \ldots, G_{k-q}$ by clique-sums on cliques of size at most $k-q$ (corresponding to the progenitors whose level is most $k-q$). Hence, each $(k-q+1)$-connected minor F of $T(G,H)$ is a minor of one of $G_0, \ldots, G_{k-q}$, and thus a minor of G can be obtained from F by removing at most $k-q$ vertices. ◀

For an integer $p \geq 1$, the *p-blowup* of a graph H_0 is the graph H obtained from H_0 by replacing every vertex u by an independent set S_u of p vertices, and by adding all edges zz' such that $z \in S_u$ and $z' \in S_{u'}$ for some $uu' \in E(H_0)$. For the purposes of constructing the graph $T(G, H)$, we order the vertices of H so that for each $u \in V(H_0)$, the vertices of S_u are consecutive in the ordering. Note that $\chi(H) \leq \chi(H_0)$, since we can assign all the vertices of S_u the color of u.

The *strong p-blowup* of H_0 is obtained from the p-blowup by making the sets S_u into cliques for each $u \in V(H_0)$. For integers $a \geq b \geq 1$, an $(a : b)$-*coloring* of H_0 is a function φ that to each vertex of H_0 assigns a subset of $\{1, \ldots, a\}$ of size b such that $\varphi(u) \cap \varphi(v) = \emptyset$ for each edge uv of H_0. The *fractional chromatic number* $\chi_f(H_0)$ is the infimum of $\{a/b : H_0 \text{ has an } (a : b)\text{-coloring}\}$. Note that if H is the strong p-blowup of a graph H_0, then a c-coloring of H gives a $(c : p)$-coloring of H_0. Consequently, we have the following.

▶ **Observation 10.** *If H is the strong p-blowup of a graph H_0, then $\chi(H) \geq p\chi_f(H_0)$.*

We now state a key result concerning the chromatic number of the graph $T(G, H)$.

▶ **Lemma 11.** *Let $p, c \geq 1$ be integers and let G be a graph. Let H_0 be a graph such that $\chi(H_0) = \chi_f(H_0) = c$, and let H be the p-blowup of H_0. Then*

$$\chi(T(G, H)) \leq c\chi(G)$$

and if $\chi(G) \geq p$, then

$$\chi(T(G, H)) \geq cp.$$

Proof. Let $V(H) = \{u_1, \ldots, u_k\}$, where $k = p|V(H_0)|$. Note that $\chi(H) \leq \chi(H_0) = c$. Let φ_H be a proper coloring of H using c colors. Let $C_1, \ldots, C_c$ be pairwise disjoint sets of $\chi(G)$ colors. For each non-leaf vertex x of $T_{n,k}$ of depth i, color G_x properly using the colors in $C_{\varphi_H(u_i)}$. Observe that this gives a proper coloring of $T(G, H)$ using at most $c\chi(G)$ colors, and thus $\chi(T(G, H)) \leq c\chi(G)$.

Suppose now that $\chi(G) \geq p$ and consider a proper coloring φ of $T(G, H)$. We construct a path $P = x_1 x_2 \ldots x_{k+1}$ in the tree $T_{n,k}$ from its root x_1 to one of the leaves and a coloring ψ of H, as follows. Suppose that we already selected $x_1, \ldots, x_i$ for some $i \leq k$. Let Z_i denote the set of progenitors of level at least $i - p + 1$ of the vertices of G_{x_i}. Since $|Z_i| \leq p - 1$ and φ uses at least $\chi(G) \geq p$ distinct colors on G_{x_i}, there exists $v \in V(G_{x_i})$ such that $\varphi(v)$ is different from the colors of all vertices of Z_i. We define $x_{i+1} = \theta_{x_i}(v)$ be the child of x_i in $T_{n,k}$ corresponding to v, and set $\psi(u_i) = \varphi(v)$.

Note that ψ is a proper coloring of H such that for each $u \in V(H_0)$, ψ assigns vertices in S_u pairwise distinct colors. Consequently, ψ is a proper coloring of the strong p-blowup of H_0, and thus ψ (and φ) uses at least $p\chi_f(H_0) = cp$ distinct colors by Observation 10. We conclude that $\chi(T(G, H)) \geq cp$. ◀

For positive integers p and k, let $K_{k\times p}$ denote the p-blowup of K_k, i.e., the complete k-partite graph with parts of size p. Let us summarize the results of this section in the special case of the graph $T(G_0, K_{k\times 4})$ with G_0 planar.

▶ **Corollary 12.** *Let G_0 be a planar graph with n vertices and let k_0 be a positive integer. Let $G = T(G_0, K_{k_0\times 4})$. The graph G has $O(n^{4k_0})$ vertices. If G_0 is 3-colorable, then $\chi(G) \leq 3k_0$, and otherwise $\chi(G) \geq 4k_0$. Furthermore, every $(4k_0 - 3)$-connected graph appearing as a minor in G is $(4k_0 - 4)$-apex.*

Proof. Note that $\chi(K_{k_0}) = \chi_f(K_{k_0}) = k_0$, $|V(K_{k_0\times 4})| = 4k_0$ and the last 4 vertices of $K_{k_0\times 4}$ form an independent set. The claims follow from Lemma 9 (with $H = K_{k_0\times 4}$, $k = 4k_0$ and $q = 4$, using the fact that every minor of a planar graph is planar) and Lemma 11 (with $H_0 = K_{k_0}$, $p = 4$ and $c = k_0$). ◀

3 Non-approximability

The main non-approximability result is a simple consequence of Corollary 12 and NP-hardness of testing 3-colorability of planar graphs.

Proof of Theorem 2. Suppose for a contradiction that there exists such a polynomial-time algorithm $\mathcal{A}$, taking as an input an F-minor-free graph G and returning an integer c such that $\chi(G) \leq c < \beta\chi(G) + (4 - 3\beta)k_0$.

Let G_0 be a planar graph, and let $G = T(G_0, K_{k_0\times 4})$. By Corollary 12, the size of G is polynomial in the size of G_0 and G is F-minor-free. Furthermore, if G_0 is 3-colorable, then $\chi(G) \leq 3k_0$, and otherwise $\chi(G) \geq 4k_0$. Hence, if G_0 is 3-colorable, then the value returned by the algorithm $\mathcal{A}$ applied to G is less than $\beta\chi(G) + (4 - 3\beta)k_0 \leq 4k_0$, and if G_0 is not 3-colorable, then the value returned is at least $\chi(G) \geq 4k_0$. This gives a polynomial-time algorithm to decide whether G_0 is 3-colorable.

However, it is NP-hard to decide whether a planar graph is 3-colorable [7], which gives a contradiction unless P = NP. ◀

Note that the graphs $T(G_0, K_{k_0\times 4})$ used in the proof of Theorem 2 have large cliques (of size greater than k_0). This turns out not to be essential—we can prove somewhat weaker non-approximability result even for graphs with clique number 3. To do so, we need to apply the construction with both G and H_0 being triangle-free. A minor issue is that testing 3-colorability of triangle-free planar graphs is trivial by Grötzsch' theorem [8]. However, this can be worked around easily.

▶ **Lemma 13.** *Let $\mathcal{G}$ denote the class of graphs such that all their 3-connected minors with at least 12 vertices are planar. The problem of deciding whether a triangle-free graph $G \in \mathcal{G}$ is 3-colorable is NP-hard.*

Proof. Let R_0 be the Grötzsch graph (R_0 is a triangle-free graph with 11 vertices and chromatic number 4, and all its proper subgraphs are 3-colorable). Let R be a graph obtained from R_0 by removing any edge uv. Note that R is 3-colorable and the vertices u and v have the same color in every 3-coloring.

Let G_1 be a planar graph. Let G_2 be obtained from G_1 by replacing each edge xy of G_1 by a copy of R whose vertex u is identified with x and an edge is added between v and y (i.e., G_2 is obtained from G_1 by a sequence of Hajós sums with copies of R_0). Clearly, G_2 is triangle-free, it is 3-colorable if and only if G_1 is 3-colorable, and $|V(G_2)| = |V(G_1)| + 10|E(G_1)|$. Furthermore, G_2 is obtained from a planar graph by clique-sums with R_0 on cliques of size two, and thus every 3-connected minor of G_2 is either planar or a minor of R_0 (and thus has at most 11 vertices). Hence, G_2 belongs to $\mathcal{G}$.

Since testing 3-colorability of planar graphs is NP-hard, it follows that testing 3-colorability of triangle-free graphs from $\mathcal{G}$ is NP-hard. ◀

We also need a graph H_0 which is triangle-free and its fractional chromatic number is large and equal to its ordinary chromatic number. Such a graph can be obtained by a standard probabilistic argument (omitted).

▶ **Lemma 14.** *For every positive integer m, there exists a triangle-free graph H_m with $O(m^4 \log^2 m)$ vertices such that $\chi(H_m) = \chi_f(H_m) = m$.*

Theorem 5 now follows in the same way as Theorem 2, using the graphs from Lemma 14 instead of cliques.

Proof of Theorem 5. Suppose for a contradiction that there exists such a polynomial-time algorithm $\mathcal{A}$, taking as an input an F-minor-free graph G with $\omega(G) \leq 3$ and returning an integer c such that $\chi(G) \leq c < \beta\chi(G) + d$. Recall that $m = \lceil d/(4-3\beta) \rceil$. Let H_m be the graph from Lemma 14. Let $k_0 = |V(H_m)|$ and let H be the 4-blowup of H_m. Let $\mathcal{G}$ be the class of graphs such that all their 3-connected minors with at least 12 vertices are planar.

Consider a triangle-free graph $G_0 \in \mathcal{G}$, and let $G = T(G_0, H)$. By Lemma 9, the size of G is polynomial in the size of G_0. Consider any $(4k_0 - 1)$-connected minor F' of G. By Lemma 9, there exists a set X of size at most $4k_0 - 4$ such that $F' - X$ is a minor of G_0. Since $F' - X$ is 3-connected and $G_0 \in \mathcal{G}$, we conclude that either $|V(F')| \leq |X| + 11 \leq 4k_0 + 7$ or $F' - X$ is planar. Consequently, $F' \neq F$, and thus G does not contain F as a minor. Furthermore, $\omega(G) \leq \omega(G_0) + \omega(H) - 1 \leq 3$.

Recall that $\chi(H_m) = \chi_f(H_m) = m$. By Lemma 11, if G_0 is 3-colorable, then $\chi(G) \leq 3m$, and otherwise $\chi(G) \geq 4m$. Hence, if G_0 is 3-colorable, then the value returned by the algorithm $\mathcal{A}$ applied to G is less than $\beta\chi(G) + d \leq 3\beta m + d \leq 4m$, and if G_0 is not 3-colorable, then the value returned is at least $\chi(G) \geq 4m$. This gives a polynomial-time algorithm to decide whether G_0 is 3-colorable, in contradiction to Lemma 13 unless P = NP. ◀

4 Approximation algorithms

Let us now turn our attention to the additive approximation algorithms. The algorithms we present use ideas similar to the ones of the 2-approximation algorithm of Demaine et al. [2] and of the additive approximation algorithms of Kawarabayashi et al. [9] and Demaine et al. [3]. We find a partition of the vertex set of the input graph G into parts L and C such that $G[L]$ has bounded tree-width (and thus its chromatic number can be determined exactly) and $G[C]$ has bounded chromatic number, and color the parts using disjoint sets of colors. The existence of such a decomposition is proved using the minor structure theorem [12], in the variant limiting the way apex vertices attach to the surface part of the decomposition. The proof of this stronger version can be found in [5]. Let us now introduce definitions necessary to state this variant of the structure theorem.

A *tree decomposition* of a graph G is a pair (T, β), where T is a tree and β is a function assigning to each vertex of T a subset of $V(G)$, such that for each $uv \in E(G)$ there exists $z \in V(T)$ with $\{u, v\} \subseteq \beta(z)$, and such that for each $v \in V(G)$, the set $\{z \in V(T) : v \in \beta(z)\}$ induces a non-empty connected subtree of T. The *width* of the tree decomposition is $\max\{|\beta(z)| : z \in V(T)\} - 1$, and the *tree-width* of a graph is the minimum of the widths of its tree decompositions.

The decomposition is *rooted* if T is rooted. For a rooted tree decomposition (T, β) and a vertex z of T distinct from the root, if w is the parent of z in T, we write $\beta \uparrow z := \beta(z) \cap \beta(w)$ and $\beta \downarrow z := \beta(z) \setminus \beta(w)$. If z is the root of T, then $\beta \uparrow z := \emptyset$ and $\beta \downarrow z := \beta(z)$. The *torso expansion* of a graph G with respect to its rooted tree decomposition (T, β) is the graph obtained from G by adding a clique on $\beta \uparrow z$ for all $z \in V(T)$.

A *path decomposition* is a tree decomposition (T, β) where T is a path. A *vortex with boundary sequence $v_1, \ldots, v_s$ and depth d* is a graph with a path decomposition $(p_1 p_2 \ldots p_s, \beta)$ such that $|\beta(p_i)| \leq d + 1$ and $v_i \in \beta(p_i)$ for $i = 1, \ldots, s$. An embedding of a graph in a surface is *2-cell* if each face of the embedding is homeomorphic to an open disk.

▶ **Theorem 15** (Dvořák and Thomas [5]). *For every non-negative integer t and a t-apex graph H, there exists a constant a_H such that the following holds. For every H-minor-free graph G, there exists a rooted tree decomposition (T, β) of G with the following properties. Let G' denote the torso expansion of G with respect to (T, β). For every $v \in V(T)$, there exists a set $A_v \subseteq \beta(v)$ of size at most a_H with $\beta \uparrow v \subseteq A_v$, a set $A'_v \subseteq A_v$ of size at most $t-1$, and subgraphs G_v, $G_{v,1}$, ..., $G_{v,m}$ of $G'[\beta(v) \setminus A_v]$ for some $m \leq a_H$ such that*

- **(a)** *$G'[\beta(v) \setminus A_v] = G_v \cup G_{v,1} \cup \ldots \cup G_{v,m}$, and for $1 \leq i < j \leq m$, the graphs $G_{v,i}$ and $G_{v,j}$ are vertex-disjoint and G' contains no edges between $V(G_{v,i})$ and $V(G_{v,j})$,*
- **(b)** *the graph G_v is 2-cell embedded in a surface Σ_v in which H cannot be drawn,*
- **(c)** *for $1 \leq i \leq m$, $G_{v,i}$ is a vortex of depth a_H intersecting G_v only in its boundary sequence, and this sequence appears in order in the boundary of a face $f_{v,i}$ of G_v, and $f_{v,i} \neq f_{v,j}$ for $1 \leq i < j \leq m$,*
- **(d)** *vertices of G_v have no neighbors in $A_v \setminus A'_v$, and*
- **(e)** *if w is a child of v in T and $\beta(w) \cap V(G_v) \neq \emptyset$, then $\beta \uparrow w \subseteq V(G_v) \cup A'_v$.*

Furthermore, the tree decomposition and the sets and subgraphs as described can be found in polynomial time.

Informally, the graph G is a clique-sum of the graphs $G'[\beta(v)]$ for $v \in V(T)$, and $\beta(v)$ contains a bounded-size set A_v of apex vertices such that $G'[\beta(v)] - A_v$ can be embedded in Σ_v up to a bounded number of vortices of bounded depth. Furthermore, at most $t-1$ of the apex vertices (forming the set A'_v) can have neighbors in the part G_v of $G'[\beta(v)] - A_v$ drawn in Σ_v, or in the other bags of the decomposition that intersect G_v. Note that it is also possible that Σ_v is the null surface, and consequently $A_v = \beta(v)$.

We need the following observation on graphs embedded up to vortices.

▶ **Lemma 16.** *Let Σ be a surface of Euler genus g and let a be a non-negative integer. Let G be a graph and let G_0, G_1, ..., G_m be its subgraphs such that $G = G_0 \cup G_1 \cup \ldots \cup G_m$, the subgraphs G_1, ..., G_m are pairwise vertex-disjoint and G contains no edges between them, G_0 is 2-cell embedded in Σ, and there exist pairwise distinct faces f_1, ..., f_m of this embedding such that for $1 \leq i \leq m$, G_i intersects G_0 only in a set of vertices contained in the boundary of f_i. If the graphs G_1, ..., G_m have tree-width at most a, then there exists a subset L_0 of vertices of G_0 such that $G_0 - L_0$ is planar and the graph $G_0[L_0] \cup G_1 \cup \ldots \cup G_m$ has tree-width at most $26g + 9m + a$.*

Proof. If Σ is the sphere, then we can set $L_0 = \emptyset$; hence, we can assume that $g \geq 1$. Let G'_0 be the graph obtained from G_0 by, for $1 \leq i \leq m$, adding a new vertex r_i drawn inside f_i and joined by edges to all vertices of $V(G_0) \cap V(G_i)$. Note that G'_0 has a 2-cell embedding in Σ extending the embedding of G. Let F be a subgraph of G'_0 such that the embedding of F in Σ inherited from the embedding of G'_0 is 2-cell and $|V(F)| + |E(F)|$ is minimum. Then F has only one face, since otherwise it is possible to remove an edge separating distinct faces of F, and F has minimum degree at least two, since otherwise we can remove a vertex of degree at most 1 from F. By generalized Euler's formula, we have $|E(F)| = |V(F)| + g - 1$, and thus F contains at most $2(g-1)$ vertices of degree greater than two. By considering the graph obtained from F by suppressing vertices of degree two, we see that F is either a cycle (if $g = 1$) or a subdivision of a graph with at most $3(g-1)$ edges.

Let M_0 be the set of vertices of F of degree at least three and their neighbors. We claim that each vertex of $V(F) \setminus M_0$ is adjacent in G'_0 to only two vertices of $V(F)$. Indeed, suppose that a vertex $x \in V(F) \setminus M_0$ has at least three neighbors in G'_0 belonging to $V(F)$. Let w and y be the neighbors of x in F, and let z be a vertex distinct from w and y adjacent to x in G'_0. The graph $F + xz$ has two faces, and by symmetry, we can assume that the edge xy separates them. Since $x \notin M_0$, the vertex y has degree two in F, and thus the embedding of $F + xz - y$ is 2-cell, contradicting the minimality of F.

Let M_1 be the set of vertices of F at distance at most 4 from a vertex of degree greater than two. Note that $|M_1| \leq 26g$. For $1 \leq i \leq m$, let N_i denote the set of vertices of $F - M_1$ that are in G_0' adjacent to a vertex of $V(G_i) \cap V(G_0) \setminus V(F)$. We claim that $|N_i| \leq 9$. Indeed, suppose for a contradiction that $|N_i| \geq 10$ and consider a path $w_4w_3w_2w_1xy_1y_2y_3y_4$ in F such that x belongs to N_i (vertices x, w_1, …, w_4, y_1, …, y_4 have degree two in F, since $x \notin M_1$). If $r_i \in V(F)$, then let Q be a path in G_0' of length two between x and r_i through a vertex of $V(G_i) \cap V(G_0) \setminus V(F)$; note that $r_i \notin \{w_1, w_2, y_1, y_2\}$, since r_i has at least 10 neighbors in $V(F)$ and belongs to M_0 by the previous paragraph and $x \notin M_1$. If $r_i \notin V(F)$, then there exists a vertex $z \in N_i \setminus \{x, w_1, \ldots, w_4, y_1, \ldots, y_4\}$; we let Q be a path of length at most four between x and z passing only through their neighbors in $V(G_i) \cap V(G_0) \setminus V(F)$ and possibly through r_i. By symmetry, we can assume that the edge xy_1 separates the two faces of $F + Q$, and the graph $F + Q - \{y_1, y_2\}$ if $r_i \in V(F)$ or $F + Q - \{y_1, y_2, y_3, y_4\}$ if $r_i \notin V(F)$ contradicts the minimality of F.

Let $M = (M_1 \cup \bigcup_{i=1}^{m} N_i) \cap V(G_0)$; we have $|M| \leq 26g + 9m$. Let $L_0 = V(F) \cap V(G_0)$. Clearly, $G_0 - L_0 \subseteq G_0' - V(F)$ is planar. Let G' be the graph obtained from $G_1 \cup \ldots \cup G_m$ by adding vertices of M as universal ones, adjacent to all other vertices of G'. The tree-width of G' is at most $a + |M| \leq 26g + 9m + a$. Note that each vertex of $L_0 \setminus V(G')$ has degree two in $G_0[L_0] \cup G_1 \cup \ldots \cup G_m$, and thus $G_0[L_0] \cup G_1 \cup \ldots \cup G_m$ is a subgraph of a subdivision of G'. We conclude that the tree-width of $G_0[L_0] \cup G_1 \cup \ldots \cup G_m$ is also at most $26g + 9m + a$. ◀

Note that the set L_0 can be found in polynomial time. For the clarity of presentation of the proof we selected F with $|V(F)| + |E(F)|$ minimum; however, it is sufficient to start with an arbitrary inclusionwise-minimal subgraph with exactly one 2-cell face (obtained by repeatedly removing edges that separate distinct faces and vertices of degree at most 1) and repeatedly perform the reductions described in the proof until each vertex of $V(F) \setminus M_0$ is adjacent in G_0' to only two vertices of $V(F)$ and until $|N_i| \leq 9$ for $1 \leq i \leq m$.

For positive integers t and a, we say that a rooted tree decomposition (T, β) of a graph G is (t, a)-*restricted* if for each vertex v of T, the subgraph of the torso expansion of G induced by $\beta \downarrow v$ is planar, $|\beta \uparrow v| \leq t - 1$, and each vertex of $\beta \downarrow v$ has at most $a - 1$ neighbors in G that belong to $\beta \uparrow v$. Using the decomposition from Theorem 15, we now partition the considered graph into a part of bounded tree-width and a (t, t)-restricted part.

▶ **Theorem 17.** *For every positive integer t and a t-apex graph H, there exists a constant c_H with the following property. The vertex set of any H-minor-free graph G can be partitioned in polynomial time into two parts L and C such that $G[L]$ has tree-width at most c_H and $G[C]$ has a (t, t)-restricted rooted tree decomposition. Additionally, for any such graph H and positive integers $a \leq b$, there exists a constant $c_{H,a,b}$ such that if G is H-minor-free and does not contain $K_{a,b}$ as a subgraph, then L and C can be chosen so that $G[L]$ has tree-width at most $c_{H,a,b}$ and $G[C]$ has a (t, a)-restricted rooted tree decomposition.*

Proof. Let a_H be the constant from Theorem 15 for H, and let g be the maximum Euler genus of a surface in which H cannot be embedded. Let $c_H = 26g + 11a_H$ and $c_{H,a,b} = c_H + \binom{t-1}{a}(b-1)$.

Since G is H-minor-free, we can in polynomial time find its rooted tree decomposition (T, β), its torso expansion G', and for each $v \in V(T)$, find A_v, A_v', G_v, $G_{v,1}$, …, $G_{v,m}$, and Σ_v as described in Theorem 15. Let L_v' be the set of vertices obtained by applying Lemma 16 to G_v, $G_{v,1}$, …, $G_{v,m}$; i.e., $G_v - L_v'$ is planar and the graph $G_v[L_v'] \cup G_{v,1} \cup \ldots \cup G_{v,m}$ has tree-width at most $26g + 10a_H$. When considering the case that G does not contain $K_{a,b}$ as a subgraph, let S_v be the set of vertices of G_v that have at least a neighbors in G belonging to A_v (and thus to A_v'); otherwise, let $S_v = \emptyset$. Since there are at most $\binom{t-1}{a}$ ways how to choose a set of a neighbors in A_v' and no b vertices can have the same set of a neighbors in A_v', we have $|S_v| \leq \binom{t-1}{a}(b-1)$. Let $L_v = (A_v \setminus \beta \uparrow v) \cup V(G_{v,1} \cup \ldots \cup G_{v,m}) \cup L_v' \cup S_v$.

We define $L = \bigcup_{v \in V(T)} L_v$. Note that $L \cap \beta(v) \subseteq L_v \cup (\beta \uparrow v) \subseteq L_v \cup A_v$. Consequently, $G'[L \cap \beta(v)]$ is obtained from $G_v[L'_v] \cup G_{v,1} \cup \ldots \cup G_{v,m}$ by adding S_v and some of the vertices of A_v, and consequently $G'[L \cap \beta(v)]$ has tree-width at most $c_{H,a,b}$ when considering the case that G does not contain $K_{a,b}$ as a subgraph and at most c_H otherwise. The graph $G[L]$ is a clique-sum of the graphs $G'[L \cap \beta(v)]$ for $v \in V(T)$, and thus the tree-width of $G[L]$ is also at most $c_{H,a,b}$ or c_H.

Let $C = V(G) \setminus L$, and consider the graph $G[C]$. For $v \in V(T)$, let $\beta'(v) = \beta(v) \cap C$. Then (T, β') is a rooted tree decomposition of $G[C]$ such that for every $v \in V(T)$, the graph $G'[\beta' \downarrow v] \subseteq G_v - L'_v$ is planar, all vertices of $\beta' \uparrow v$ adjacent in G to a vertex of $\beta' \downarrow v$ belong to A'_v (and thus there are at most $t-1$ such vertices), and when considering the case that G does not contain $K_{a,b}$ as a subgraph, each vertex of $\beta' \downarrow v$ has at most $a-1$ neighbors in G belonging to $\beta' \uparrow v$.

Note that $\beta' \uparrow v$ can contain vertices not belonging to A'_v, and thus $\beta' \uparrow v$ can have size larger than $t-1$, and the tree decomposition (T, β') is not necessarily (t,t)-restricted. However, by the condition (e) from the statement of Theorem 15, the vertices of $(\beta' \uparrow v) \setminus A'_v$ can only be contained in the bags of descendants of v which are disjoint from $V(G_v)$, and thus we can fix up this issue as follows.

If w is a child of v and $\beta'(w) \cap V(G_v) = \emptyset$, we say that the edge vw is *skippable*; note that in that case $\beta' \uparrow w \subseteq \beta' \uparrow v$. For each vertex w of T, let $f(w)$ be the nearest ancestor of w such that the first edge on the path from $f(w)$ to w in T is not skippable. Let T' be the rooted tree with vertex set $V(T)$ where the parent of each vertex w is $f(w)$. Observe that (T', β') is a tree decomposition of $G[C]$. Furthermore, denoting by z the child of $f(w)$ on the path from $f(w)$ to w in T, note that if a vertex $x \in \beta' \uparrow f(w)$ is contained in $\beta'(w)$, then $x \in \beta(z)$, and since the edge $f(w)z$ is not skippable, the condition (e) from the statement of Theorem 15 implies that $x \in A'_{f(w)}$.

Hence, letting $\beta''(v) = (\beta' \downarrow v) \cup (A'_v \cap C)$ for each vertex v of T', we conclude that (T', β'') is a rooted tree decomposition of $G[C]$ which is (t,t)-restricted, and when considering the case that G does not contain $K_{a,b}$ as a subgraph, the decomposition is (t,a)-restricted. ◀

Let us now consider the chromatic number of graphs with a (t,a)-restricted tree decomposition.

▶ **Lemma 18.** *Let a and t be positive integers. Let G be a graph with a (t,a)-restricted rooted tree decomposition (T, β). The chromatic number of G is at most $\min(t+3, a+4)$. Additionally, if G is triangle-free, then the chromatic number of G is at most $\lceil (13t+172)/14 \rceil$.*

Proof. We can color G using $t+3$ colors, starting from the root of the tree decomposition, as follows. Suppose that we are considering a vertex $v \in V(T)$ such that $\beta \uparrow v$ is already colored. Since $|\beta \uparrow v| \leq t-1$, this leaves at least 4 other colors to be used on $G[\beta \downarrow v]$. Hence, we can extend the coloring to $G[\beta \downarrow v]$ by the Four Color Theorem.

We can also color G using $a+4$ colors, starting from the root of the tree decomposition, as follows. For each vertex x of $\beta \downarrow v$, at most $a-1$ colors are used on its neighbors in $\beta \uparrow v$, leaving x with at least 5 available colors not appearing on its neighbors. Since $G[\beta \downarrow v]$ is planar, we can color it from these lists of size at least 5 using the result of Thomassen [15], again extending the coloring to $G[\beta \downarrow v]$.

Finally, suppose that G is triangle-free. Let G' be the torso expansion of G with respect to (T, β), and let $c = \lceil (13t+172)/14 \rceil$. We again color G starting from the root of the tree decomposition using at most c colors. Additionally, we choose the coloring so that the following invariant is satisfied: ($\star$) for each vertex w of T and for each independent set I in $G[\beta(w)]$ such that $I \cap \beta \downarrow w$ induces a clique in G', at most $c-6$ distinct colors are used on I.

Let v be a vertex of T. Suppose we have already colored $\beta \uparrow v$, and we want to extend the coloring to $\beta \downarrow v$. Note that the choice of this coloring may only affect the validity of the invariant $(\star)$ at v and at descendants of v in T. Consider any descendant w of T. Coloring $\beta \downarrow v$ can only assign color to vertices of $\beta \uparrow w$, and since G' is the torso expansion of G, the set $\beta \uparrow w \cap \beta \downarrow v$ induces a clique in G'. Consequently, the validity of $(\star)$ at v implies the validity at w (until more vertices of G are assigned colors), and thus when choosing the coloring of $\beta \downarrow v$, we only need to ensure that $(\star)$ holds at v.

The graph $G'[\beta \downarrow v]$ is planar, and thus it is 5-degenerate; i.e., there exists an ordering of its vertices such that each vertex is preceded by at most 5 of its neighbors. Let us color the vertices of $\beta \downarrow v$ according to this ordering, always preserving the validity of $(\star)$ at v. Suppose that we are choosing a color for a vertex $x \in V(\beta \downarrow v)$. Let P_x consist of the neighbors of x in G' belonging to $\beta \downarrow v$ that precede it in the ordering; we have $|P_x| \leq 5$. Note that all cliques in $G'[\beta \downarrow v]$ containing x and with all other vertices already colored are subsets of $P_x \cup \{x\}$. Let $Q_x = P_x \cup \beta \uparrow v$; we have $|Q_x| \leq t + 4$.

Let N_x consist of vertices of Q_x that are adjacent to x in G. We say that a color a is forbidden at x if there exists an independent set $A_a \subseteq Q_x \setminus N_x$ of G such that $A_a \cap P_x$ is a clique in G' and $c - 6$ colors distinct from a appear on A_a. Observe that assigning x a color which neither appears on N_x nor is forbidden results in a proper coloring that preserves the invariant $(\star)$ at v.

Suppose first that no color is forbidden at x. Since G is triangle-free, $N_x \setminus P_x$ is an independent set in $G[\beta \uparrow v]$, and by $(\star)$, at most $c - 6$ colors appear on $N_x \setminus P_x$. Since $|P_x| \leq 5$, it follows that some color does not appear on N_x, as required.

Hence, we can assume that some color is forbidden at x, and thus there exists an independent set $Z_1 \subseteq Q_x \setminus N_x$ of size at least $c - 6$ such that vertices of Z_1 are assigned pairwise distinct colors. Since $|Q_x| \leq t + 4$, at most $t + 4 - (c - 6) = t + 10 - c$ of these colors appear at least twice on Q_x, and thus there exists a set $Z_2 \subseteq Z_1$ of size at least $c - 6 - (t + 10 - c) = 2c - t - 16$ such that the color of each vertex of Z_2 appears exactly once on Q_x (and thus does not appear on N_x). Let $Z = Z_2 \setminus P_x$; we have $|Z| \geq 2c - t - 21$. We claim that not all colors appearing on Z are forbidden; since such colors do not appear on N_x, we can use them to color x.

For contradiction, assume that colors of all vertices of Z are forbidden at x. Let $Z = \{z_1, \ldots, z_m\}$ for some $m \geq 2c - t - 21$, and for $a = 1, \ldots, m$, let a be the color of z_a. Since a is forbidden at x, there exists an independent set $A_a \subseteq Q_x \setminus N_x$ such that $A_a \cap P_x$ is a clique in G' and $c - 6$ colors distinct from a appear on A_a. Note that $A_a \cup \{z_a\}$ is not an independent set, as otherwise this set contradicts the invariant $(\star)$ at v. Hence, we can choose a neighbor $f(a)$ of z_a in A_a. Since Z is an independent set, we have $f(a) \notin Z$. Furthermore, we claim that the preimage in f of each vertex has size at most 6: if say $f(z_1) = \ldots = f(z_7) = y$, then for $i = 1, \ldots, 7$, the vertex z_i would have a neighbor y in the independent set A_1, and thus $z_1, \ldots, z_7 \notin A_1$; however, the only appearance of colors $1, \ldots, 7$ in Q_x is on the vertices $z_1, \ldots, z_7$, and thus at most $c - 7$ colors would appear on A_1. We conclude that $|f(Z)| \geq |Z|/6$, and thus $t + 4 \geq |Q_x| \geq |Z| + |f(Z)| \geq \frac{7}{6}|Z| \geq \frac{7}{6}(2c - t - 21) \geq (6t + 25)/6$. This is a contradiction. ◀

Combining Lemma 18 with Theorem 17 (coloring $G[L]$ using at most $\chi(G)$ colors in linear time [1], using the fact that $G[L]$ has bounded tree-width, and coloring $G[C]$ using a disjoint set of colors), we obtain Theorems 4, 7, and 8.

References

1. Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23:11–24, 1989.
2. Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 637–646. IEEE, 2005.
3. Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Approximation algorithms via structural results for apex-minor-free graphs. *Automata, Languages and Programming*, pages 316–327, 2009.
4. Matt DeVos, Guoli Ding, Bogdan Oporowski, Daniel Sanders, Bruce Reed, Paul Seymour, and Dirk Vertigan. Excluding any graph as a minor allows a low tree-width 2-coloring. *J. Comb. Theory, Ser. B*, 91:25–41, 2004.
5. Zdeněk Dvořák and Robin Thomas. List-coloring apex-minor-free graphs. *arXiv*, 1401.1399, 2014.
6. Zdeněk Dvořák, Daniel Král', and Robin Thomas. Three-coloring triangle-free graphs on surfaces VII. A linear-time algorithm. *ArXiv*, 1601.01197, 2016.
7. Michael Garey and David Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman & Co. New York, NY, USA, 1979.
8. Herbert Grötzsch. Ein Dreifarbensatz für dreikreisfreie Netze auf der Kugel. *Math.-Natur. Reihe*, 8:109–120, 1959.
9. Ken-ichi Kawarabayashi, Erik D. Demaine, and MohammadTaghi Hajiaghayi. Additive approximation algorithms for list-coloring minor-closed class of graphs. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1166–1175. SIAM, 2009.
10. Alexandr Kostochka. Lower bound of the Hadwiger number of graphs by their average degree. *Combinatorica*, 4:307–316, 1984.
11. Neil Robertson and Paul D. Seymour. Graph Minors. V. Excluding a planar graph. *J. Combin. Theory, Ser. B*, 41:92–114, 1986.
12. Neil Robertson and Paul D. Seymour. Graph Minors. XVI. Excluding a non-planar graph. *J. Combin. Theory, Ser. B*, 89(1):43–76, 2003.
13. Robin Thomas. Conference Graph Theory 2008 at Sandbjerg Manor; slides at `http://people.math.gatech.edu/~thomas/SLIDE/beyondgrot.pdf`.
14. Carsten Thomassen. Five-Coloring Maps on Surfaces. *J. of Combin. Theory, Ser. B*, 59:89–105, 1993.
15. Carsten Thomassen. Every planar graph is 5-choosable. *J. Combin. Theory, Ser. B*, 62:180–181, 1994.
16. David Zuckerman. Linear degree extractors and the inapproximability of Max Clique and Chromatic Number. *Theory of Computing*, 3:103–128, 2007.

How to Navigate Through Obstacles?

Eduard Eiben[1]
Department of Informatics, University of Bergen, Bergen, Norway
eduard.eiben@uib.no
https://orcid.org/0000-0003-2628-3435

Iyad Kanj
School of Computing, DePaul University, Chicago, USA
ikanj@cs.depaul.edu

Abstract

Given a set of obstacles and two points in the plane, is there a path between the two points that does not cross more than k different obstacles? This is a fundamental problem that has undergone a tremendous amount of work by researchers in various areas, including computational geometry, graph theory, wireless computing, and motion planning. It is known to be NP-hard, even when the obstacles are very simple geometric shapes (*e.g.*, unit-length line segments). The problem can be formulated and generalized into the following graph problem: Given a planar graph G whose vertices are colored by color sets, two designated vertices $s, t \in V(G)$, and $k \in \mathbb{N}$, is there an s-t path in G that uses at most k colors? If each obstacle is connected, the resulting graph satisfies the color-connectivity property, namely that each color induces a connected subgraph.

We study the complexity and design algorithms for the above graph problem with an eye on its geometric applications. We prove a set of hardness results, among which a result showing that the color-connectivity property is crucial for any hope for fixed-parameter tractable (FPT) algorithms, as without it, the problem is W[SAT]-hard parameterized by k. Previous results only implied that the problem is W[2]-hard. A corollary of this result is that, unless W[2] = FPT, the problem cannot be approximated in FPT time to within a factor that is a function of k. By describing a generic plane embedding of the graph instances, we show that our hardness results translate to the geometric instances of the problem.

We then focus on graphs satisfying the color-connectivity property. By exploiting the planarity of the graph and the connectivity of the colors, we develop topological results that allow us to prove that, for any vertex v, there exists a set of paths whose cardinality is upper bounded by a function of k, that "represents" the valid s-t paths containing subsets of colors from v. We employ these structural results to design an FPT algorithm for the problem parameterized by both k and the treewidth of the graph, and extend this result further to obtain an FPT algorithm for the parameterization by both k and the length of the path. The latter result generalizes and explains previous FPT results for various obstacle shapes, such as unit disks and fat regions.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms, Theory of computation → Computational geometry, Mathematics of computing → Graph algorithms

Keywords and phrases parameterized complexity and algorithms, motion planning, barrier coverage, barrier resilience, colored path, minimum constraint removal, planar graphs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.48

Related Version A full version of the paper is available at [8], `https://arxiv.org/abs/1712.04043`.

[1] Supported by Pareto-Optimal Parameterized Algorithms (ERC Starting Grant 715744)

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 48; pp. 48:1–48:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

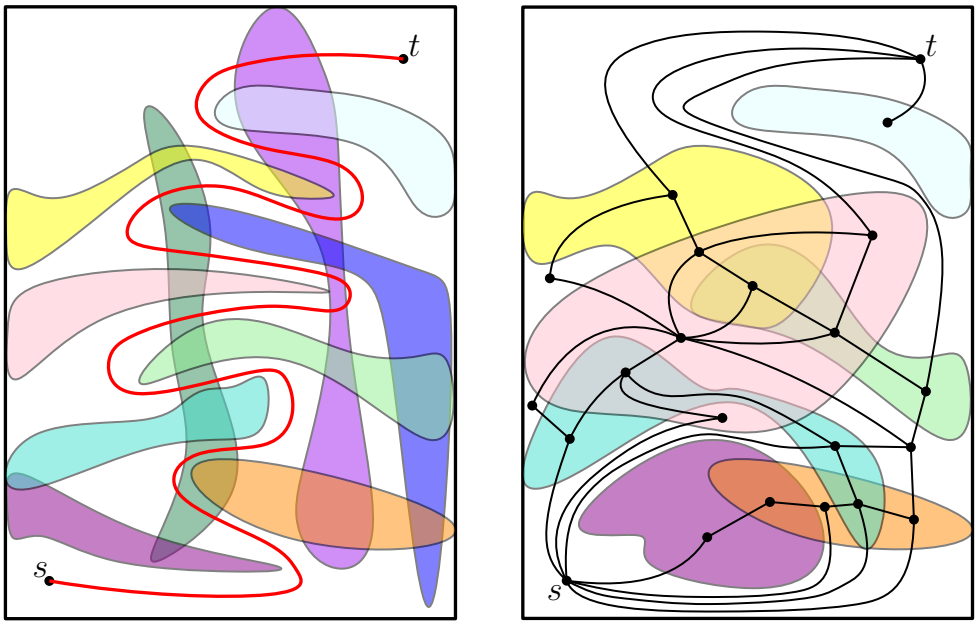


Figure 1 Illustration of instances of the problem under consideration drawn within a bounding box. The left figure shows an instance in which the optimal path crosses two obstacles, zigzagging between the other obstacles. The right figure shows an instance and its auxiliary plane graph.

1 Introduction

We consider the following problem: Given a set of obstacles and two designated points in the plane, is there a path between the two points that does not cross more than k obstacles? Equivalently, can we remove k obstacles so that there is an obstacle-free path between the two designated points? We refer to this problem as Obstacle Removal, and to its restriction to instances in which each obstacle is connected as Connected Obstacle Removal.

By considering the auxiliary plane graph that is the dual of the plane subdivision determined by the obstacles, Obstacle Removal was formulated and generalized into the following graph problem, referred to as Colored Path (see Figure 1 for illustrations):

Colored Path

Given: A planar graph G; a set of colors C; $\chi : V \longrightarrow 2^C$; two designated vertices $s, t \in V(G)$; and $k \in \mathbb{N}$

Question: Does there exist an s-t path in G that uses at most k colors?

Denote by Colored Path-Con the restriction of Colored Path to instances in which each color induces a connected subgraph of G.

As we discuss next, Connected Obstacle Removal and Colored Path are fundamental problems that have undergone a tremendous amount of work, albeit under different names and contexts, by researchers in various areas, including computational geometry, graph theory, wireless computing, and motion planning.

1.1 Related Work

In motion planning, the goal is generally to move a robot from a starting position to a final position, while avoiding collision with a set of obstacles. This is usually referred to as the *piano-mover*'s problem. Obstacle Removal is a variant of the piano-mover's

problem, in which the obstacles are in the plane and the robot is represented as a point. Since determining if there is an obstacle-free path for the robot in this case is solvable in polynomial time, if no such path exists, it is natural to seek a path that intersects as few obstacles as possible. Motivated by planning applications, Connected Obstacle Removal and Colored Path were studied under the name Minimum Constraint Removal [7, 9, 10, 11]. Connected Obstacle Removal has also been studied extensively, motivated by applications in wireless computing, under the name Barrier Coverage or Barrier Resilience [1, 2, 12, 13, 15, 16]. In such applications, we are given a field covered by sensors (usually simple shapes such as unit disks), and the goal is to compute a minimum set of sensors that need to fail before an entity can move undetected between two given sites.

Kumar *et al.* [13] were the first to study Connected Obstacle Removal. They showed that for unit-disk obstacles in some restricted setting, the problem can be solved in polynomial time. The complexity of the general case for unit-disk obstacles remains open. Several works showed the NP-hardness of the problem, even when the obstacles are very simple geometric shapes such as line segments (*e.g.*, see [1, 15, 16]). The complexity of the problem when each obstacle intersects a constant number of other obstacles is open [9, 11].

Bereg and Kirkpatrick [2] designed approximation algorithms when the obstacles are unit disks by showing that the length, referred to as the *thickness* [2] (*i.e.*, number of regions visited), of a shortest path that crosses k disks is at most $3k$; this follows from the fact that a shortest path does not cross a disk more than a constant number of times.

Korman *et al.* [12] showed that Connected Obstacle Removal is FPT parameterized by k for unit-disk obstacles, and extended this result to similar-size fat-region obstacles with a constant *overlapping number*, which is the maximum number of obstacles having nonempty intersection. Their result draws the observation, which was also used in [2], that for unit-disk (and fat-region) obstacles, the length of an optimal path can be upper bounded by a linear function of the number of obstacles crossed (*i.e.*, the parameter). This observation was then exploited by a branching phase that decomposes the path into subpaths in (simpler) restricted regions, enabling a similar approach to that of Kumar *et al.* [13].

Motivated by its applications to networking, among other areas, the problem of computing a minimum-colored path in a graph received considerable attention (*e.g.*, see [3, 17]). The problem was shown to be NP-hard in several works [3, 4, 11, 17]. Most of the NP-hardness reductions start from Set Cover, and result in instances of Colored Path (*i.e.*, planar graphs), as was also observed by [2]. These reductions are FPT-reductions, implying the W[2]-hardness of Colored Path. Moreover, these reductions imply that, unless P = NP, the minimization version of Colored Path cannot be approximated to within a factor of $c \lg n$, for any constant $c < 1$. Hauser [11], and Gorbenko and Popov [10], implemented exact and heuristic algorithms for the problem on general graphs. Very recently, Eiben *et al.* [7] designed exact and heuristic algorithms for Colored Path and Obstacle Removal, and proved computational lower bounds on their subexponential-time complexity, assuming the Exponential Time Hypothesis.

1.2 Our Results and Techniques

We study the complexity and parameterized complexity of Colored Path and Colored Path-Con, eyeing the implications on their geometric counterparts Obstacle Removal and Connected Obstacle Removal, respectively. The proofs of the hardness results we obtain are too long and technical to be included, and for those we refer to the full paper [8].

Clearly, Colored Path is in the parameterized class XP. We show that the color-connectivity property is crucial for any hope for an FPT-algorithm, since even very restricted instances and combined parameterizations of Colored Path are W[1]-complete:

▶ **Theorem 1.1.** Colored Path*, restricted to instances of pathwidth at most 4, and in which each vertex contains at most one color and each color appears on at most 2 vertices, is* W[1]-complete parameterized by k.

▶ **Theorem 1.2.** Colored Path*, parameterized by both k and the length of the sought path ℓ, is* W[1]-complete.

Without restrictions, the problem sits high in the parameterized complexity hierarchy:

▶ **Theorem 1.3.** Colored Path*, parameterized by k, is* W*[SAT]-hard and is in* W[P].

A corollary of Theorem 1.3 is that, unless W[2] = FPT, Colored Path cannot be approximated in FPT time to within a factor that is a function of k.

We can produce a generic construction [8] that can be used to realize any graph instance of Colored Path as a geometric instance of Obstacle Removal. Using this generic construction, the hardness results in Theorems 1.1–1.3, and the inapproximability result discussed above, translate to Obstacle Removal. Previously, Colored Path was only known to be W[2]-hard, via the standard reduction from Set Cover [3, 11, 17]. Our results refine the parameterized complexity and approximability of Colored Path and Obstacle Removal.

As it turns out, the color-connectivity property without planarity is hopeless: We can tradeoff planarity for color-connectivity by adding a single vertex that serves as a color-connector, thus establishing the W[SAT]-hardness of the problem on apex graphs.

The above hardness results show that we can focus our attention on Colored Path-Con. We show the following algorithmic result:

▶ **Theorem 1.4** (Theorem 4.7)**.** Colored Path-Con*, parameterized by both k and the treewidth ω of the input graph, is* FPT*.*

We remark that bounding the treewidth does not make Colored Path-Con much easier, as we can show that Colored Path-Con is NP-hard even for 2-outerplanar graphs of pathwidth at most 3 [8].

The folklore dynamic programming approach based on tree decomposition, used for the Hamiltonian Path/Cycle problems, does not work for Colored Path-Con to prove the result in Theorem 1.4 for the following reasons. As opposed to the Hamiltonian Path/Cycle problems, where it is sufficient to keep track of how the path/cycle interacts with each bag in the tree decomposition, this is not sufficient in the case of Colored Path-Con because we also need to keep track of which color sets are used on both sides of the bag. Although (by color connectivity) any subset of colors appearing on both sides of a bag must appear on vertices in the bag as well, there can be too many such subsets (up to $|C|^k$, where C is the set of colors), and certainly we cannot afford to enumerate all of them if we seek an FPT algorithm. To overcome this issue, we develop in Section 3 topological structural results that exploit the planarity of the graph and the connectivity of the colors to show the following. For any vertex $w \in V(G)$, and for any pair of vertices $u, v \in V(G)$, the set of (valid) u-v paths in $G - w$ that use colors appearing on vertices in the face of $G - w$ containing w can be "represented" by a minimal set of paths whose cardinality is a function of k.

In Section 4, we extend the notion of a minimal set of paths w.r.t. a single vertex to a "representative set" of paths w.r.t. a specific bag, and a specific enumerated configuration for the bag, in a tree decomposition of the graph. This enables us to use the upper bound on the size of a minimal set of paths, derived in Section 3, to upper bound the size of a

representative set of paths w.r.t. a bag and a configuration. This, in turn, yields an upper bound on the size of the table stored at a bag, in the dynamic programming algorithm, by a function of both k and the treewidth of the graph, thus yielding the desired result.

In Section 5, we extend the FPT result for Colored Path-Con in Theorem 1.4 w.r.t. the parameters k and ω, to the parameterization by both k and the length ℓ of the path:

▶ **Theorem 1.5** (Theorem 5.1). *Colored Path-Con, and hence Connected Obstacle Removal, parameterized by both k and ℓ is FPT.*

The dependency on both ℓ and k is *essential* for the result in Theorem 1.5, as we can show that if we parameterize only by k, or only by ℓ, then the problem becomes W[1]-hard [8].

The result in Theorem 1.5 generalizes and explains Korman *et al.*'s results [12] showing that Connected Obstacle Removal is FPT parameterized by k for unit-disk obstacles, which they also generalized to similar-size fat-region obstacles with bounded overlapping number. Their results exploit the obstacle shape to upper bound the length of the path by a linear function of k, and then use branching to reduce the problems to a simpler setting. Our result directly implies that, regardless of the (connected) obstacle shapes, as long as the path length is upper bounded by some function of k (Corollary 5.2), the problem is FPT. The FPT result in Theorem 1.5 also implies that:

▶ **Corollary 1.6** (Corollary 5.3). *For any computable function h, Colored Path-Con restricted to instances in which each color appears on at most $h(k)$ vertices, is FPT parameterized by k.*

The result in Corollary 1.6 has applications to Connected Obstacle Removal, in particular, to the interesting case when the obstacles are convex polygons, each intersecting a constant number of other polygons. The question about the complexity of this problem was posed in [9, 11], and remains open. The result in Corollary 1.6 implies that this problem is FPT.

We finally mention that it remains open whether Colored Path-Con and Connected Obstacle Removal are FPT parameterized by k only.

2 Preliminaries

We assume familiarity with graph theory and parameterized complexity. We refer the reader to the standard books [5, 6].

For a set S, we denote by 2^S the power set of S. Let $G = (V, E)$ be a graph, let $C \subset \mathbb{N}$ be a finite set of colors, and let $\chi : V \longrightarrow 2^C$. A vertex v in V is *empty* if $\chi(v) = \emptyset$. A color c *appears on*, or is *contained in*, a subset S of vertices if $c \in \bigcup_{v \in S} \chi(v)$. For two vertices $u, v \in V(G)$, $\ell \in \mathbb{N}$, a u-v path $P = (u = v_0, \ldots, v_r = v)$ in G is *ℓ-valid* if $|\bigcup_{i=0}^{r} \chi(v_i)| \leq \ell$; that is, if the total number of colors appearing on the vertices of P is at most ℓ. A color $c \in C$ is *connected* in G, or simply *connected*, if $\bigcup_{c \in \chi(v)} \{v\}$ induces a connected subgraph of G. The graph G is *color-connected*, if for every $c \in C$, c is connected in G.

For an instance (G, C, χ, s, t, k) of Colored Path or Colored Path-Con, if s and t are nonempty vertices, we can remove their colors and decrement k by $|\chi(s) \cup \chi(t)|$ because their colors appear on every s-t path. If afterwards k becomes negative, then there is no k-valid s-t path in G. Moreover, if s and t are adjacent, then the path (s, t) is a path with the minimum number of colors among all s-t paths in G. Therefore, we will assume:

▶ **Assumption 2.1.** *For an instance (G, C, χ, s, t, k) of Colored Path or Colored Path-Con, we can assume that s and t are nonadjacent empty vertices.*

3 Structural Results

Let G be a color-connected plane graph, C a set of colors, and $\chi : V \longrightarrow 2^C$. In this section, we present structural results that are the cornerstone of the FPT-algorithm for Colored Path-Con presented in the next section. We start by giving an intuitive description of the plan for this section.

As mentioned in Section 1, the main issue facing a dynamic programming algorithm based on tree decomposition, is how to upper bound, by a function of k and the treewidth, the number of k-valid paths between (any) two vertices u and v that use color sets contained in a certain bag. As it turns out, this number cannot be upper bounded as desired. Instead, we "represent" those paths using a minimal set $\mathcal{P}$ of k-valid u-v paths, in the sense that any k-valid u-v path can be replaced by a path from $\mathcal{P}$ that is not "worse" than it. To do so, it suffices to represent the k-valid u-v paths that use color sets contained in a third vertex w, by a set whose cardinality is a function of k. This will enable us to extend the notion of a minimal set of k-valid u-v paths w.r.t. a single vertex to a representative set for the whole bag, which is the key ingredient of the dynamic programming FPT-algorithm – based on tree decomposition – in the next section.

As it turns out, the paths that matter are those that use "external" colors w.r.t. w (defined below), since those colors have the potential of appearing on both sides of a bag containing w. Therefore, the ultimate goal of this section is to define a notion of a minimal set $\mathcal{P}$ of k-valid u-v paths with respect to w (Definition 3.5), and to upper bound $|\mathcal{P}|$ by a function of k. Upper bounding $|\mathcal{P}|$ by a function of k turns out to be quite challenging, and requires ideas and topological results that will be discussed later in this section.

Throughout this section, we shall assume that G is color-connected. We start with the following simple observation that holds because of this assumption:

▶ **Observation 3.1.** *Let $x, y \in V(G)$ be such that there exists a color $c \in \mathcal{C}$ that appears on both x and y. Then any x-y vertex-separator in G contains a vertex on which c appears.*

Let G' be a plane graph, let $w \in V(G')$, and let f be the face in $G' - w$ such that w is interior to f; we call f the *external face* w.r.t. w in $G' - w$, and the vertices incident to f *external vertices* w.r.t. w in $G' - w$. A color $c \in C$ is an *external color* w.r.t. w in $G' - w$, or simply *external* to w in $G' - w$, if c appears on an external vertex w.r.t. w in $G' - w$; otherwise, c is *internal* to w in $G' - w$. The following observation is easy to see:

▶ **Observation 3.2.** *Let G be a color-connected graph, and let $w \in V(G)$. Let H be any subgraph of $G - w$. If c is an external color to w in $G - w$ and c appears on some vertex in H, then c is an external color to w in H. This also implies that the set of internal colors to w in H is a subset of the set of internal colors to w in $G - w$.*

▶ **Definition 3.3.** Let s, t be two designated vertices in G, and let x, y be two adjacent vertices in G such that $\chi(x) = \chi(y)$. Define the following operation to x and y, referred to as a *color contraction* operation, that results in a graph G', a color function χ', and two designated vertices s', t' in G', obtained as follows:

- G' is obtained from G by contracting the edge xy, which results in a new vertex z;
- $s' = s$ (resp. $t' = t$) if $s \notin \{x, y\}$ (resp. $t \notin \{x, y\}$), and $s' = z$ (resp. $t' = z$) otherwise; and
- $\chi' : V(G') \longrightarrow 2^C$ is defined as $\chi'(w) = \chi(w)$ if $w \neq z$, and $\chi'(z) = \chi(x) = \chi(y)$.

G is *irreducible* if there does not exist two vertices in G to which the color contraction operation is applicable.

▶ **Lemma 3.4.** *Let G be a color-connected plane graph, C a color set, $\chi : V \longrightarrow 2^C$, $s,t \in V(G)$, and $k \in \mathbb{N}$. Suppose that the color contraction operation is applied to two vertices in G to obtain G', χ', s',t', as described in Definition 3.3. Then G' is a color-connected plane graph, and there is a k-valid s-t path in G if and only if there is a k-valid s'-t' path in G'.*

▶ **Definition 3.5.** Let $u,v,w \in V(G)$. A set $\mathcal{P}$ of k-valid u-v paths in $G-w$ is said to be *minimal* w.r.t. w if:

- **(i)** There does not exist two paths $P_1, P_2 \in \mathcal{P}$ such that $\chi(P_1) \cap \chi(w) = \chi(P_2) \cap \chi(w)$;
- **(ii)** there does not exist two paths $P_1, P_2 \in \mathcal{P}$ such that $\chi(P_1) \subseteq \chi(P_2)$; and
- **(iii)** for any $P \in \mathcal{P}$, there does not exist a u-v path P' in $G-w$ such that $\chi(P') \subsetneq \chi(P)$.

Clearly, for any $u,v,w \in V(G)$, a minimal set of k-valid u-v paths in $G-w$ exists.

▶ **Observation 3.6.** *Let $u,v,w \in V(G)$. Any set of u-v paths that is minimal w.r.t. w contains at most one path whose vertices contain* only *internal colors w.r.t. w in $G-w$.*

To derive an upper bound on the cardinality of a minimal set $\mathcal{P}$ of k-valid u-v paths w.r.t. a vertex w, we select a maximal set $\mathcal{M}$ of color-disjoint paths in $\mathcal{P}$. We first upper bound $|\mathcal{M}|$ by a function of k, which requires developing several results of topological nature. The key ingredient for upper bounding $|\mathcal{M}|$ is showing that the subgraph induced by the paths in $\mathcal{M}$ has a u-v vertex-separator of cardinality $O(k)$ (Lemma 3.10). We then upper bound $|\mathcal{M}|$ (Lemma 3.12) by upper bounding the number of different traces of the paths of $\mathcal{M}$ on this small separator, and inducting on both sides of the separator. Finally, we show (Theorem 3.13) that $|\mathcal{P}|$ is upper bounded by a function of $|\mathcal{M}|$, which proves the desired upper bound on $|\mathcal{P}|$. We proceed to the details.

For the rest of this section, let $u,v,w \in V(G)$, and let $\mathcal{P}$ be a set of minimal k-valid u-v paths in $G-w$. Let $\mathcal{M}$ be a set of minimal k-valid color-disjoint u-v paths in $G-w$, and let M be the subgraph of $G-w$ induced by the edges of the paths in $\mathcal{M}$.

▶ **Observation 3.7.** *If $P \in \mathcal{M}$ contains a color c that is external to w in M, then c appears on a vertex in P that is incident to the external face to w in M.*

▶ **Lemma 3.8.** *Let G' be a plane graph with a face f, let $u,v \in V(G')$, and let $u_1,\ldots,u_r$, $r \geq 3$, be the neighbors of u. Suppose that, for each $i \in [r]$, there exists a u-v path P_i containing u_i and a vertex incident to f different from v, and such that P_i does not contain any u_j, $j \in [r]$, $j \neq i$. Then there exist two paths P_i, P_j, $i,j \in [r]$, $i \neq j$, such that $V(P_i) \cup V(P_j) - \{v\}$ is a vertex-separator separating $\{u_1,\ldots,u_r\} \setminus \{u_i,u_j\}$ from v.*

▶ **Lemma 3.9.** *Let x,y be two vertices in an irreducible subgraph G' of G, and let f be a face in G'. Then there are at most two color-disjoint x-y paths in G' that contain only colors that appear on f.*

▶ **Lemma 3.10.** *Suppose that M is irreducible, then there exist paths $P_1,P_2,P_3 \in \mathcal{M}$ such that $M - P_1 - P_2 - P_3$ has a u-v vertex-separator of cardinality at most $2k+3$.*

Proof. By Observation 3.6 and Observation 3.2, $\mathcal{M}$ contains at most one path that contains only internal colors w.r.t. w in M. Therefore, it suffices to show that $\mathcal{M}$ contains two paths P_1,P_2 such that $M - P_1 - P_2$ has a u-v vertex-separator of cardinality at most $2k+3$, assuming that every path in $\mathcal{M}$ contains an external color w.r.t. w in M.

By Observation 3.7, every path in $\mathcal{M}$ passes through an external vertex w.r.t. w in M that contains a color external to w in M. Because the paths in $\mathcal{M}$ are pairwise color-disjoint

and u and v are empty vertices, every path in $\mathcal{M}$ passes through a vertex on the external face of w in M that is different from u and v. Let $u_1, \ldots, u_q$ be the neighbors of u in M, and note that since u is empty and M is irreducible, each u_i, $i \in [q]$, contains a color. Let $P_1, \ldots, P_q$ be the paths in $\mathcal{M}$ containing $u_1, \ldots, u_q$, respectively, and note that since the paths in $\mathcal{M}$ are color-disjoint, no P_i passes through u_j, for $j \neq i$. By Lemma 3.8, there are two paths in $P_1, \ldots, P_q$, say P_1, P_2 without loss of generality, such that $V_{12} = V(P_1) \cup V(P_2) - \{v\}$ is a vertex-separator that separates $\{u_3, \ldots, u_q\}$ from v.

We proceed by contradiction and assume that $M^- = M - P_1 - P_2$ does not have a u-v vertex-separator of cardinality $2k + 3$. By Menger's theorem [5], there exists a set $\mathcal{D}$ of $r' \geq 2k + 3$ vertex-disjoint u-v paths in M^-. Since V_{12} separates $\{u_3, \ldots, u_q\}$ from v in M, every u-v path in M^- intersects at least one of P_1, P_2 at a vertex other than v. It follows that there exists a path in $\{P_1, P_2\}$, say P_1, that intersects at least $k + 2$ paths in $\mathcal{D}$ at vertices other than v. Since the paths in $\mathcal{D}$ are vertex-disjoint and incident to u, we can order the paths in $\mathcal{D}$ that intersect P_1 around u (in counterclockwise order) as $\langle Q_1, \ldots, Q_r \rangle$, where $r \geq k + 2$, and Q_{i+1} is counterclockwise from Q_i, for $i \in [r - 1]$. P_1 intersects each path Q_i, $i \in [r]$, possibly multiple times. Moreover, since the paths in $\mathcal{M}$ are pairwise color-disjoint, each intersection between P_1 and a path Q_i, $i \in [r]$, must occur at an empty vertex. We choose $r - 1$ subpaths, $P_1^1, \ldots, P_1^{r-1}$, of P_1 satisfying the property that the endpoints of P_1^i are on Q_i and Q_{i+1}, for $i = 1, \ldots, r - 1$, and the endpoints of P_1^i are the only vertices on P_1^i that appear on a path Q_j, for $j \in [r]$. It is easy to verify that the subpaths $P_1^1, \ldots, P_1^{r-1}$ of P_1 can be formed by following the intersection of P_1 with the sequence of (ordered) paths $Q_1, \ldots, Q_r$.

Recall that the endpoints of $P_1^1, \ldots, P_1^{r-1}$ are empty vertices. Since M is irreducible, no two empty vertices are adjacent, and hence, each subpath P_1^i must contain an internal vertex v_i that contains at least one color. We claim that no two vertices v_i, v_j, $1 \leq i < j \leq r - 1$, contain the same color. Suppose not, and let v_i, v_j, $i < j$, be two vertices containing a color c. Since v_i, v_j are internal to P_1^i and P_1^j, respectively, $Q_1, \ldots, Q_r$ are vertex-disjoint u-v paths, and by the choice of the subpaths $P_1^1, \ldots, P_1^{r-1}$, the paths Q_i and Q_{i+1} form a Jordan curve, and hence a vertex-separator in G, separating v_i from v_j.

By Observation 3.1, color c must appear on a vertex in Q_p, $p \in \{i, i + 1\}$, and this vertex is clearly not in P_1 since P_1 intersects Q_p at empty vertices. Since every vertex in M appears on a path in $\mathcal{M}$, and c appears on $P_1 \in \mathcal{M}$ and on a vertex not in P_1, this contradicts that the paths in $\mathcal{M}$ are pairwise color-disjoint, and proves the claim.

Since no two vertices v_i, v_j, $1 \leq i < j \leq r$, contain the same color, the number $r - 1$ of subpaths $P_1^1, \ldots, P_1^{r-1}$ is upper bounded by the number of distinct colors that appear on P_1, which is at most k. It follows that r is at most $k + 1$, contradicting our assumption above and proving the lemma. ◀

▶ **Lemma 3.11.** *Let S be a minimal u-v vertex-separator in M. Let M_u, M_v be a partition of $M - S$ containing u and v, respectively, and such that there is no edge between M_u and M_v. For any vertex $x \in S$, M_u is contained in a single face of $M_v + x$.*

▶ **Lemma 3.12.** *$|\mathcal{M}| \leq g(k)$, where $g(k) = \mathcal{O}(c^k k^{2k})$, for some constant $c > 1$.*

Proof (sketch). By Observation 3.6, there is at most one path in $\mathcal{M}$ that contains only internal colors w.r.t. w in $G - w$. Therefore, it suffices to upper bound the number of paths in $\mathcal{M}$ that contain at least one external color to w in $G - w$. By Observation 3.2, every such path in $\mathcal{M}$ contains a color that is external to w in M.

The proof is by induction on k, over every color-connected plane graph G, every triplet of vertices u, v, w in G, and every minimal set $\mathcal{M}$ w.r.t. w of k-valid pairwise color-disjoint

u-v paths in $G - w$. If $k = 1$, any path in $\mathcal{M}$ contains exactly one external color w.r.t. w in M. By Lemma 3.9, at most two paths in $\mathcal{M}$ contain only external colors. Assume by the inductive hypothesis that, for each $1 \leq i < k$, we have $|\mathcal{M}| \leq g(i)$. We can assume that M is irreducible; otherwise, we can apply the color contraction operation and replace $\mathcal{M}$ with a set of paths satisfying the same properties.

By Lemma 3.10, there are (at most) 3 paths in $\mathcal{M}$, such that the subgraph of M induced by the remaining paths of $\mathcal{M}$ has a u-v vertex-separator S satisfying $|S| \leq 2k + 3$. Remove these 3 paths from $\mathcal{M}$, and now we can assume that M has a u-v vertex-separator S satisfying $|S| \leq 2k + 3$; we will add 3 to the upper bound on $|\mathcal{M}|$ at the end. We can assume that S is minimal. S separates M into two subgraphs M_u and M_v such that $u \in V(M_u)$, $v \in V(M_v)$, and there is no edge between M_u and M_v. We partition $\mathcal{M}$ into the following groups, where each group excludes the paths satisfying the properties of the groups defined before it: (1) The set of paths in $\mathcal{M}$ that contain a nonempty vertex in S; (2) the set of paths $\mathcal{M}_u^k$ consisting of each path P in $\mathcal{M}$ such that all colors on P appear on vertices in M_u (these colors could still appear on vertices in M_v as well); (3) the set of paths $\mathcal{M}_v^k$ consisting of each path P in $\mathcal{M}$ such that all colors on P appear on vertices in M_v; and (4) the set $\mathcal{M}^{<k}$ of remaining paths in $\mathcal{M}$, satisfying that each path contains a nonempty external vertex to w in M and contains less than k colors from each of M_u and M_v.

Since the paths in $\mathcal{M}$ are pairwise color-disjoint, no nonempty vertex in S appears on two distinct paths from group (1). Therefore, the number of paths in group (1) is at most $|S| \leq 2k + 3$. Observe that the vertices in S contained in any path from groups (2)-(4) are empty vertices. To upper bound the number of paths in group (2), for each path P, there is a last vertex x_P (*i.e.*, farthest from u) in P that is in S. Fix a vertex $x \in S$, and let us upper bound the number of paths P in group (2) for which $x = x_P$. Let P_v be the subpath of P from x to v. Note that since v is empty and all the vertices in S that are contained in paths in group (2) are empty, and since M is irreducible, P_v must contain at least one color. Since all colors appearing on P appear on vertices in M_u, all colors appearing on P_v appear in M_u. By Lemma 3.11, M_u is contained in a single face f of $M_v + x$. Since f is a vertex-separator that separates $V(M_u)$ from $V(P_v)$ in G, by Observation 3.1, every color that appears on P_v appears on f. By Lemma 3.9, there are at most two x-v paths that contain only colors that appear on f. This shows that there are at most two paths in group (2) for which x is the last vertex in S. Since $|S| \leq 2k + 3$, this upper bounds the number of paths in group (2) by $2(2k + 3) = 4k + 6$. By symmetry, the number of paths in group (3) is upper bounded by $4k + 6$.

To upper bound the number of paths in group (4), let $S = \{s_2, \ldots, s_{r-1}\}$, and extend S by adding the two vertices $s_1 = u$ and $s_r = v$ to form the set $A = \{s_1, s_2, \ldots, s_r\}$. For every two (distinct) vertices $s_j, s_{j'} \in A$, we define a set of paths $\mathcal{P}_{jj'}$ in $G - w$ whose endpoints are s_j and $s_{j'}$ as follows. For each path P in group (4), partition (the edges in) P into subpaths $P_1, \ldots, P_q$ satisfying the property that the endpoints of each P_i, $i \in [q]$, are in A, and no internal vertex to P_i is in A. For each P_i, $i \in [q]$, such that P_i contains a vertex that contains an external color to w in $G - w$, let P_i' (possibly P_i) be a subpath in $G - w$ between the endpoints of P_i satisfying that $\chi(P_i') \subseteq \chi(P_i)$ and $\chi(P_i')$ is minimal w.r.t. containment. Since P contains a vertex that contains an external color to w in $G - w$, it is easy to see that there exists an $i \in [q]$ such that P_i' contains a vertex containing an external color to w in $G - w$. Pick any $i \in [q]$ satisfying that P_i' contains a vertex containing an external color to w in $G - w$, associate P with P_i', and assign P_i' to the set of paths $\mathcal{P}_{jj'}$ such that s_j and $s_{j'}$ are the endpoints of P_i'. The map that takes each P to its P_i' is clearly a bijection.

Therefore, it suffices to upper bound the number of paths assigned to the sets $\mathcal{P}_{jj'}$. Fix a set $\mathcal{P}_{jj'}$. The paths in $\mathcal{P}_{jj'}$ have $s_j, s_{j'}$ as endpoints, and are pairwise color-disjoint. It is not difficult to show that $\mathcal{P}_{jj'}$ is a minimal set of $(k-1)$-valid s_j-$s_{j'}$ paths in $G-w$ w.r.t. w. By the inductive hypothesis, we have $|\mathcal{P}_{jj'}| \leq g(k-1)$. Since the number of sets $\mathcal{P}_{jj'}$ is at most $\binom{2k+5}{2}$, the number of paths in group (4) is $\mathcal{O}(k^2) \cdot g(k-1)$.

It follows that $|\mathcal{M}| \leq g(k)$, where $g(k)$ satisfies:

$$g(k) \leq 3 + (2k+3) + 2(4k+6) + \mathcal{O}(k^2) \cdot g(k-1) = \mathcal{O}(k^2) \cdot g(k-1),$$

where 3 accounts for the 3 paths removed from $\mathcal{M}$. Solving the aforementioned recurrence relation gives $g(k) = \mathcal{O}(c^k k^{2k})$, where $c > 1$ is a constant. ◀

Applying Lemma 3.12 to a maximal set $\mathcal{M}$ of color-disjoint paths in $\mathcal{P}$, and using an inductive proof, we can show the following theorem:

▶ **Theorem 3.13.** *Let G be a plane color-connected graph, and let $w \in V(G)$. Let G' be a subgraph of $G-w$, and let $u, v \in V(G')$. Every set $\mathcal{P}$ of minimal k-valid u-v paths in G' w.r.t. w satisfies $|\mathcal{P}| \leq h(k)$, where $h(k) = \mathcal{O}(c^{k^2} k^{2k^2+k})$, for some constant $c > 1$.*

4 The Algorithm

In this section, we highlight how the FPT algorithm for Colored Path-Con, parameterized by both k and the treewidth of the input graph works. As pointed out in Section 3, there can be too many (*i.e.*, more than FPT-many) subsets of colors that appear in a bag, and hence, that the algorithm may need to store/remember. To overcome this issue, we extend the notion of a minimal set of k-valid u-v paths w.r.t. a vertex – from the previous section – to a "representative set" of paths w.r.t. a specific bag and a specific enumerated configuration for the bag. This allows us to upper bound the size of the table, in the dynamic programming algorithm, stored at a bag by a function of both k and the treewidth of the graph.

Let (G, C, χ, s, t, k) be an instance of Colored Path-Con. Let $(\mathcal{V}, \mathcal{T})$ be a nice tree decomposition of G. By Assumption 2.1, we can assume that s and t are nonadjacent empty vertices. We add s and t to every bag in $\mathcal{T}$, and now we have $\{s,t\} \subseteq X_i$, for every bag $X_i \in \mathcal{T}$. For a bag X_i, we say that $v \in X_i$ is *useful* if $|\chi(v)| \leq k$. Let U_i be the set of all useful vertices in X_i and let $\overline{U_i} = X_i \setminus U_i$. We denote by V_i the set of vertices in the bags of the subtree of $\mathcal{T}$ rooted at X_i. For any two vertices $u, v \in X_i$, let $G^i_{uv} = G[(V_i \setminus X_i) \cup \{u, v\}]$. We extend the notion of a minimal set of k-valid u-v paths w.r.t. a vertex, developed in the previous section, to the set of vertices in a bag of $\mathcal{T}$.

▶ **Definition 4.1.** A set of k-valid u-v paths $\mathcal{P}_{uv}$ in G^i_{uv} is *minimal* w.r.t. X_i if it satisfies the following properties:

- **(i)** There does not exist two paths $P_1, P_2 \in \mathcal{P}_{uv}$ such that $\chi(P_1) \cap \chi(X_i) = \chi(P_2) \cap \chi(X_i)$;
- **(ii)** there does not exist two paths $P_1, P_2 \in \mathcal{P}_{uv}$ such that $\chi(P_1) \subseteq \chi(P_2)$; and
- **(iii)** for any $P \in \mathcal{P}_{uv}$ there does not exist a u-v path P' in G^i_{uv} such that $\chi(P') \subsetneq \chi(P)$.

The following lemma uses the upper bound on the cardinality of a minimal set of k-valid u-v paths w.r.t. a vertex, derived in Theorem 3.13 in the previous section, to obtain an upper bound on the cardinality of a minimal set of k-valid u-v paths w.r.t. a bag of $\mathcal{T}$:

▶ **Lemma 4.2.** *Let X_i be bag, $u, v \in X_i$, and $\mathcal{P}_{uv}$ a set of k-valid u-v paths in G^i_{uv} that is minimal w.r.t. X_i. Then the number of paths in $\mathcal{P}_{uv}$ is at most $h(k)^{|X_i|}$, where $h(k) = \mathcal{O}(c^{k^2} k^{2k^2+k})$, for some constant $c > 1$.*

▶ **Definition 4.3.** Let X_i be a bag in $\mathcal{T}$. A *pattern* π for X_i is a sequence $(v_1 = s, \sigma_1, v_2, \sigma_2, \ldots, \sigma_{r-1}, v_r = t)$, where $\sigma_i \in \{0,1\}$ and $v_i \in U_i$. For a bag X_i, and a pattern $(v_1 = s, \sigma_1, v_2, \sigma_2, \ldots, \sigma_{r-1}, v_r = t)$ for X_i, we say that a sequence of paths $\mathcal{S} = (P_1, \ldots, P_{r-1})$ *conforms* to (X_i, π) if:

- For each $j \in [r-1]$, $\sigma_j = 1$ implies that P_j is an induced path from v_j to v_{j+1} whose internal vertices are contained in $V_i \setminus X_i$ and P_j is empty otherwise; and
- $|\chi(\mathcal{S})| = |\bigcup_{j\in[r-1]} \chi(P_j)| \leq k$.

▶ **Definition 4.4.** Let X_i be a bag, π a pattern for X_i, and $\mathcal{S}_1, \mathcal{S}_2$ two sequences of paths that conform to (X_i, π). We write $\mathcal{S}_1 \preceq_i \mathcal{S}_2$ if $|\chi(\mathcal{S}_1) \cup (\chi(\mathcal{S}_2) \cap \chi(X_i))| \leq |\chi(\mathcal{S}_2)|$.

Using the relation $\preceq_i$ on the set of sequences that conform to (X_i, π), we can define the key notion of representative sets that makes the dynamic programming approach work:

▶ **Definition 4.5.** Let X_i be a bag and $\pi = (v_1, \sigma_1, v_2 \ldots, \sigma_{r-1}, v_r)$ a pattern for X_i. A set $\mathcal{R}_\pi$ of sequences that conform to (X_i, π) is a *representative* set for (X_i, π) if:

- **(i)** For every sequence $\mathcal{S}_1 \in \mathcal{R}_\pi$, and for every sequence $\mathcal{S}_2 \neq \mathcal{S}_1$ that conforms to (X_i, π), if $\mathcal{S}_1 \preceq_i \mathcal{S}_2$ then $\mathcal{S}_2 \notin \mathcal{R}_\pi$;
- **(ii)** for every sequence $\mathcal{S} \in \mathcal{R}_\pi$, and for every path $P \in \mathcal{S}$ between v_j and v_{j+1}, $j \in [r-1]$, there does not exist a v_j-v_{j+1} path P' in $G^i_{v_j v_{j+1}}$ such that $\chi(P') \subsetneq \chi(P)$; and
- **(iii)** for every sequence $\mathcal{S} \notin \mathcal{R}_\pi$ that conforms to (X_i, π) and satisfies that no two paths in $\mathcal{S}$ share a vertex that is not in X_i, there is a sequence $\mathcal{W} \in \mathcal{R}_\pi$ such that $\mathcal{W} \preceq_i \mathcal{S}$.

The following lemma uses the upper bound on the cardinality of a minimal set of k-valid u-v paths w.r.t. a bag X_i, derived in Lemma 4.2, to obtain an upper bound on the cardinality of a representative set w.r.t. a bag and a fixed pattern (X_i, π):

▶ **Lemma 4.6.** *Let X_i be bag, π a pattern for X_i, and $\mathcal{R}_\pi$ be a representative set for (X_i, π). Then the number of sequences in $\mathcal{R}_\pi$ is at most $h(k)^{|X_i|^2}$, where $h(k) = \mathcal{O}(c^{k^2} k^{2k^2+k})$, for some constant $c > 1$.*

For each bag X_i, we maintain a table Γ_i that contains, for each pattern for X_i, a representative set of sequences $\mathcal{R}_\pi$ for (X_i, π). The rest is a technical dynamic programming algorithm over $(\mathcal{V}, \mathcal{T})$ that computes the table Γ_i at a bag X_i for each bag type (introduce, forget, join) in the nice tree decomposition. We conclude with the following theorem:

▶ **Theorem 4.7.** *There is an algorithm that on input (G, C, χ, s, t, k) of* Colored Path-Con*, either outputs a k-valid s-t path in G or decides that no such path exists, in time $\mathcal{O}^\star(f(k)^{6\omega^2})$, where ω is the treewidth of G and $f(k) = \mathcal{O}(c^{k^2} k^{2k^2+k})$, for some constant $c > 1$. Therefore,* Colored Path-Con *parameterized by both k and the treewidth ω of the input graph is* FPT.

5 Extensions and Applications

In this section, we explain how to extend the FPT result for Colored Path-Con w.r.t. the parameterization by both k and the treewidth of the graph, to the parameterization by both k and the length ℓ of the sought path, and discuss important applications of this extended result. We formally define the problem w.r.t. the parameterization by k and ℓ:

Bounded-length Colored Path-Con

Given: A planar graph G; a set of colors C; $\chi : V \longrightarrow 2^C$; two designated vertices $s, t \in V(G)$; and $k, \ell \in \mathbb{N}$

Question: Does there exist a k-valid s-t path of length at most ℓ in G?

To extend Theorem 4.7, we repeatedly contract every edge uv incident to a vertex v whose distance to s is more than $\ell+1$; we assign the resulting vertex the color set $\chi(u) \cup \chi(v)$. (We do not delete such vertices in order to preserve the color-connectivity property.) Afterwards, the radius of G is at most $\ell+1$, and hence G has treewidth at most $3 \cdot (\ell+1)+1 = 3\ell+4$ [14]. Although the treewidth of G is bounded by a function of ℓ, we cannot use the FPT algorithm for Colored Path-Con, parameterized by k and the treewidth of G, to solve Bounded-length Colored Path-Con because a k-valid path returned by the algorithm for Colored Path-Con may have length more than ℓ. We can extend the FPT results for Colored Path-Con to Bounded-length Colored Path-Con to show the following:

▶ **Theorem 5.1.** Bounded-length Colored Path-Con *parameterized by both k and the length of the path is* FPT.

We now describe applications of Theorem 5.1. The first application is a direct consequence of this theorem.

▶ **Corollary 5.2.** *For any computable function h, the restriction of* Colored Path-Con *to instances in which the length of the path is at most $h(k)$ is* FPT *parameterized by k.*

We note that the above restriction of Colored Path-Con can be shown to be NP-hard.

Corollary 5.2 directly implies Korman *et al.*'s results [12], showing that Obstacle Removal is FPT for unit-disk obstacles and for similar-size fat-region obstacles with constant overlapping number. Using Bereg and Kirkpatrick's result [2], the length of a shortest k-valid path for unit-disk obstacles is at most $3k$ (see also Lemma 3 in Korman *et al.* [12]). By Corollary 2 in [12], the length of a shortest k-valid path for similar-size fat-region obstacles with constant overlapping number is linear in k. Corollary 5.2 generalizes these FPT results, which required quite some effort, and provides an explanation to why the problem is FPT for such restrictions, namely because the path length is upper bounded by a function of k.

The second application is related to an open question posed in [9, 11]. For an instance $I = (G, C, \chi, s, t, k)$ of Colored Path-Con, and a color $c \in C$, define the *intersection number* of c, denoted $\iota(c)$, to be the number of vertices in G on which c appears. Define the *intersection number* of G, $\iota(G)$, as $\max\{\iota(c) \mid c \in C\}$.

▶ **Corollary 5.3.** *For any computable function h,* Colored Path-Con *restricted to instances (G, C, χ, s, t, k) satisfying $\iota(G) \leq h(k)$ is* FPT *parameterized by k.*

Corollary 5.3 has applications pertaining to instances of Connected Obstacle Removal whose auxiliary graphs have intersection number bounded by a function of k. An interesting case that was studied is when the obstacles are convex polygons, each intersecting at most a constant number of other polygons. The complexity of this problem was posed as an open question in [9, 11], and remains unresolved. Corollary 5.3 implies that the problem is FPT, even for the more general setting in which the obstacles are arbitrary convex regions satisfying that the number of regions intersected by any region is a constant. (Note that convexity is important here.)

References

1 H. Alt, S. Cabello, P. Giannopoulos, and C. Knauer. Minimum cell connection in line segment arrangements. *International Journal of Computational Geometry and Applications*, 27(3):159–176, 2017.

2 S. Bereg and D. Kirkpatrick. Approximating barrier resilience in wireless sensor networks. In *Proceedings of ALGOSENSORS*, pages 29–40, 2009.

3 R. Carr, S. Doddi, G. Konjevod, and M. Marathe. On the red-blue set cover problem. In *Proceedings of SODA*, pages 345–353, 2000.

4 D. Chan and D. Kirkpatrick. Multi-path algorithms for minimum-colour path problems with applications to approximating barrier resilience. *Theoretical Computer Science*, 553:74–90, 2014.

5 R. Diestel. *Graph Theory, 4th Edition*. Springer, 2012.

6 R. Downey and M. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, Berlin, Heidelberg, 2013.

7 E. Eiben, J. Gemmell, I. Kanj, and A. Youngdahl. Improved results for minimum constraint removal. In *Proceedings of AAAI*. AAAI Press, 2018.

8 E. Eiben and I. Kanj. How to navigate through obstacles? *CoRR*, abs/1712.04043v1, 2017. `arXiv:1712.04043`.

9 L. Erickson and S. LaValle. A simple, but NP-hard, motion planning problem. In *Proceedings of AAAI*. AAAI Press, 2013.

10 A. Gorbenko and V. Popov. The discrete minimum constraint removal motion planning problem. In *Proceedings of the American Institute of Physics*, volume 1648. AIP Press, 2015.

11 K. Hauser. The minimum constraint removal problem with three robotics applications. *International Journal of Robotics Research*, 33(1):5–17, 2014.

12 M. Korman, M. Löffler, R. Silveira, and D. Strash. On the complexity of barrier resilience for fat regions and bounded ply. *Computational Geometry*, 72:34–51, 2018.

13 S. Kumar, T. Lai, and A. Arora. Barrier coverage with wireless sensors. In *Proceedings of MOBICOM*, pages 284–298. ACM, 2005.

14 N. Robertson and P. Seymour. Graph minors. III. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

15 K. Tseng and D. Kirkpatrick. On barrier resilience of sensor networks. In *Proceedings of ALGOSENSORS*, pages 130–144, 2012.

16 S. Yang. *Some Path Planning Algorithms in Computational Geometry and Air Traffic Management*. PhD thesis, University of New Yort at Stony Brook. Available at: https://dspace.sunyconnect.suny.edu/handle/1951/59927, 2012.

17 S. Yuan, S. Varma, and J. Jue. Minimum-color path problems for reliability in mesh networks. In *Proceedings of INFOCOM*, pages 2658–2669, 2005.

Faster Algorithms for Integer Programs with Block Structure

Friedrich Eisenbrand
EPFL, 1015 Lausanne, Switzerland
friedrich.eisenbrand@epfl.ch

Christoph Hunkenschröder
EPFL, 1015 Lausanne, Switzerland
christoph.hunkenschroder@epfl.ch

Kim-Manuel Klein
EPFL, 1015 Lausanne, Switzerland
kim-manuel.klein@epfl.ch

Abstract

We consider integer programming problems $\max\{c^T x : \mathcal{A}x = b, l \leq x \leq u, x \in \mathbb{Z}^{nt}\}$ where $\mathcal{A}$ has a (recursive) block-structure generalizing *n-fold* integer programs which recently received considerable attention in the literature. An n-fold IP is an integer program where $\mathcal{A}$ consists of n repetitions of submatrices $A \in \mathbb{Z}^{r \times t}$ on the top horizontal part and n repetitions of a matrix $B \in \mathbb{Z}^{s \times t}$ on the diagonal below the top part. Instead of allowing only two types of block matrices, one for the horizontal line and one for the diagonal, we generalize the n-fold setting to allow for arbitrary matrices in every block. We show that such an integer program can be solved in time $n^2 t^2 \varphi \cdot (r\, s\, \Delta)^{\mathcal{O}(rs^2+sr^2)}$ (ignoring logarithmic factors). Here Δ is an upper bound on the largest absolute value of an entry of $\mathcal{A}$ and φ is the largest binary encoding length of a coefficient of c. This improves upon the previously best algorithm of Hemmecke, Onn and Romanchuk that runs in time $n^3 t^3 \varphi \cdot \Delta^{\mathcal{O}(st(r+t))}$. In particular, our algorithm is not exponential in the number t of columns of A and B.

Our algorithm is based on a new upper bound on the ℓ_1-norm of an element of the *Graver basis* of an integer matrix and on a proximity bound between the LP and IP optimal solutions tailored for IPs with block structure. These new bounds rely on the *Steinitz Lemma*.

Furthermore, we extend our techniques to the recently introduced *tree-fold IPs*, where we again present a more efficient algorithm in a generalized setting.

2012 ACM Subject Classification Theory of computation → Integer programming

Keywords and phrases n-fold, Tree-fold, Integer Programming

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.49

Related Version A full version of the paper is available at `https://arxiv.org/abs/1802.06289`.

Funding This work was supported by the Swiss National Science Foundation (SNSF) within the project *Convexity, geometry of numbers, and the complexity of integer programming (Nr. 163071)*.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 49; pp. 49:1–49:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

An *integer program (IP)* is an optimization problem of the form

$$\max\{c^T x : Ax = b,\, l \leq x \leq u,\, x \in \mathbb{Z}^n\} \tag{1}$$

which is described by a *constraint matrix* $A \in \mathbb{Z}^{m\times n}$, an *objective function* vector $c \in \mathbb{Z}^n$ a *right-hand side* vector $b \in \mathbb{Z}^m$ and *lower* and *upper bounds* $l \leq x \leq u$. Integer programming is one of the most important paradigms in the field of algorithms as a breadth of combinatorial optimization problems have an IP-model, see, e.g. [17, 20]. Since integer programming is NP-hard, there is a strong interest in restricted versions of integer programs that can be solved in polynomial time, while still capturing interesting classes of combinatorial optimization problems. A famous example is the class of integer programs with *totally unimodular* constraint matrix, capturing flow, bipartite matching, and shortest path problems for example. This setting has been extended to *bimodular* integer programming recently [1].

Another such polynomial-time solvable restriction is n-fold integer programming [6]. Given two matrices $A \in \mathbb{Z}^{r\times t}$ and $B \in \mathbb{Z}^{s\times t}$ and a vector $b \in \mathbb{Z}^{r+ns}$ for some $r, s, t, n \in \mathbb{Z}_+$. An n-fold Integer Program (n-fold IP) is an integer program (1) with constraint matrix

$$\mathcal{A} = \begin{pmatrix} A & A & \dots & A \\ B & 0 & \dots & 0 \\ 0 & B & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & B \end{pmatrix} \tag{2}$$

Clearly, one can assume that $t \geq r$ and $t \geq s$ holds, as linearly dependent equations can be removed. Notice that the number of variables of an n-fold integer program is $t \cdot n$. The best known algorithm to solve an n-fold IP is due to Hemmecke, Onn and Romanchuk [10] with a running time of $\mathcal{O}(n^3t^3\varphi) \cdot \Delta^{\mathcal{O}(st(r+t))}$, where Δ is the absolute value of the largest entry in $\mathcal{A}$ and φ is the logarithm of the largest absolute value of a component of c. For fixed Δ, r, s and t, the running time depends only polynomially (cubic) on the number of variables and is therefore more efficient than applying algorithms for general IPs based on lattice-basis reduction [12, 16] or dynamic programming [7, 19].

The n-fold setting has gained strong momentum in the last years, especially in the fields of *parameterized complexity* and *approximation algorithms*. An algorithm is *fixed parameter tractable (fpt)* with respect to a parameter k derived from the input, if its running time is of the form $f(k) \cdot n^{O(1)}$ for some computable function f. The result of Hemmecke et al. [10] shows that integer programming is fixed parameter tractable with respect to Δ, s, r and t.

This opens the possibility to model combinatorial optimization problems with a fixed parameter as an n-fold integer program, see for instance [3, 13] and thereby obtain novel new results in the area of parameterized complixity. Very recently Jansen, Klein, Maack and Rau [11] used n-fold IPs to formulate an enhanced *configuration IP*, that is capable to track additional properties of configurations. With this enhanced IP they were able to develop approximation algorithms for several scheduling problems that involve setups. Not only for the scheduling problems, but also in the design of efficient algorithms for string and social choice problems, n-fold IPs have been successfully applied [14, 15].

A generalization of the classical n-fold IP, called *tree-fold* IP, was very recently introduced by Chen and Marx [2]. A matrix A is of tree-fold structure, if it is of recursive n-fold

structure, i.e. the matrices $B^{(i)}$ in IP (2) are of n'-fold structure themselves, and so on. Chen and Marx presented an algorithm to solve tree-fold IPs which runs in time $f(L) \cdot n^3 \varphi$, where φ is the encoding length and L involves parameters of the tree like the height of the tree and the number of variables and rows of the involved sub-matrices. They applied the tree-fold IP to a special case of the traveling salesman problem, where m clients have to visit every node of a weighted tree and the objective is to minimize the longest tour over all clients. Using the framework of tree-fold IPs, they obtained an fpt algorithm with a running time of $f(K) \cdot |V|^{\mathcal{O}(1)}$, where K is the longest tour of a client in the optimal solution and V is the set of vertices of the tree. However, the function f involves a term with a tower of K exponents.

1.1 Graver Bases and Augmentation Algorithms

Before we discuss our contributions, we have to review the core concepts of the algorithm of Hemmecke, Onn and Romanchuk [10] in a nutshell.

Suppose we are solving a general integer program (1) with constraint matrix $A \in \mathbb{Z}^{m \times n}$ and that we have a feasible solution z_0 at hand. Let $z^\star$ be an optimal solution. The vector $z^\star - z_0$ lies in the kernel of A, i.e., $A(z^\star - z_0) = 0$. An integer vector $y \in \ker(A)$ is called a *cycle* of A. Two vectors $u, v \in \mathbb{R}^n$ are said to be *sign compatible* if $u_i \cdot v_i \geq 0$ for each i. A cycle $y \in \ker(A)$ is *indecomposable* if it is not the sum of two sign-compatible and non-zero cycles of A. The set of indecomposable and integral elements from the kernel of A is called the *Graver basis* of A, [8], see also [18, 5].

A result of Cook, Fonlupt and Schrijver [4] implies that there exist $2n$ Graver-basis elements $g_1, \ldots, g_{2n} \in \ker(A)$ each sign compatible with $z^\star - z_0$ such that

$$z^\star - z_0 = \sum_{i=1}^{2n} \lambda_i g_i$$

holds for $\lambda_i \in \mathbb{N}_0$. For each i one has that $z_0 + \lambda_i g_i$ is a feasible integer solution of (1). Furthermore, there exists one i with $c^T(z^\star - z_0)/(2\,n) \leq \lambda_i c^T g_i$. Thus there exists an element g of the Graver basis of A and a positive integer $\lambda \in \mathbb{N}$ such that $z_0 + \lambda\, g$ is feasible and the gap to the optimum value has been reduced by a factor of $1 - 1/(2\,n)$.

Why should it be any simpler to find such an *augmenting vector* g as above? The crucial ingredient that is behind the power of this approach are *bounds* on the ℓ_1-norm of elements of the Graver basis of A. In some cases, these bounds are much more restrictive than the original lower and upper bounds $l \leq x \leq u$ and thus help in dynamic programming. In fact, each element g of the Graver basis of A has ℓ_1-norm bounded by $\|g\|_1 \leq \delta \cdot (n - m)$ where δ is the largest absolute value of a sub-determinant of A, see [18]. Applying the Hadamard bound, this means that

$$\|g\|_1 \leq m^{m/2} \Delta^m \cdot (n - m), \tag{3}$$

where Δ is a largest absolute value of an entry of A. Let us denote $m^{m/2}\Delta^m \cdot (n-m)$ by G_A. In order to find an augmenting solution which reduces the optimality gap by a factor of (roughly) $1 - 1/n$ one solves the following *augmentation integer program* with a *suitable* λ,

$$\max\{c^T y : Ay = 0,\, l - z_0 \leq \lambda \cdot y \leq u - z_0,\, \|y\|_1 \leq G_A, y \in \mathbb{Z}^n\}. \tag{4}$$

and replaces z_0 by $z_0 + \lambda \cdot y^\star$, where $y^\star$ is the optimal solution of (4). The number of augmenting steps can be bounded by $\mathcal{O}(n \log(c^T(z^\star - z_0)))$.

At first sight, it seems that one has not gained much with this approach, except that the right-hand side vector b has disappeared. In the case of n-fold integer programming however,

the ℓ_1-norm of an element of the Graver basis of $\mathcal{A}$ is bounded by a function in r, s, t and Δ and thus much smaller than the bound (3). This can be exploited in dynamic programming approaches.

Contributions of this paper. We present several elementary observations that, together, result in a much faster algorithm for integer programs with block structure including n-fold and tree-fold integer programs. We start with the following.

i) The ℓ_1-norm of an element of the Graver basis of a given matrix $A \in \mathbb{Z}^{m\times n}$ is bounded by $(2\,m\cdot\Delta+1)^m$, where Δ is an upper bound on the absolute value of each entry of A. This is shown with the Steinitz lemma and uses similar ideas as in [7]. Compared to the previous best bound (3), this new bound is independent on the number of columns n of A.

We then turn our attention to integer programming problems

$$\max\{c^T x : \mathcal{A}\,x = b,\, l \le x \le u,\, x \in \mathbb{Z}^{n\times t}\} \tag{5}$$

with constraint matrix of the form

$$\mathcal{A} = \begin{pmatrix} A^{(1)} & A^{(2)} & \dots & A^{(n)} \\ B^{(1)} & 0 & \dots & 0 \\ 0 & B^{(2)} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & B^{(n)} \end{pmatrix},$$

where $A^{(1)},\dots,A^{(n)} \in \mathbb{Z}^{r\times t}$ and $B^{(1)},\dots,B^{(n)} \in \mathbb{Z}^{s\times t}$ are arbitrary matrices. This is a more general setting than n-fold integer programming, since the matrices on the top line and on the diagonal respectively do not have to repeat. In this setting, we obtain the following results.

ii) The ℓ_1-norm of an element of the Graver basis of $\mathcal{A}$ is bounded by $\mathcal{O}(r\,s\,\Delta)^{(r+1)(s+1)}$ which is independent on the number of columns t of the $A^{(i)}$ and $B^{(i)}$.

iii) We next provide a special proximity bound for integer programs with block structure (5). Let $x^\star$ be an arbitrary optimal solution of the linear programming relaxation of (5). We show that there exists an optimal solution $z^\star$ of (5) with

$$\|x^\star - z^\star\|_1 \le n\,t\,(r\,s\,\Delta)^{\mathcal{O}(r\,s)}.$$

iv) We then exploit the bounds ii) and iii) in a new dynamic program to solve (5). Its running time is bounded by

$$n^2t^2\varphi\log^2 nt\cdot(rs\Delta)^{\mathcal{O}(r^2s+rs^2)} + \mathrm{LP}$$

where φ denotes the largest binary encoding length of c, and LP denotes the time needed to solve the LP relaxation of (5).

The main advantage of the running time of our algorithm is the improved dependency on the parameter t. In contrast, the previous best known algorithm by Hemmecke, Onn and Romanchuk [10] for classical n-fold IPs invovles a term $\Delta^{\mathcal{O}(st^2)}$ and therefore has an exponential dependency on t. Recall that we can assume that $t \ge r, s$ holds. The number of columns t can be very large. Even if we do not allow column-repetitions, t can be as large as Δ^{r+s} and in applications involving *configuration IPs* this is often the order of magnitude one is dealing with. Knop, Koutecky and Mnich [14] improved the dependency of t in a special

setting of n-fold to a factor $t^{\mathcal{O}(r)}$. In their setting, the matrix B on the diagonal consists of one line of ones only. Our running time is an improvement of their result also in this case.

Next, we generalize the notion of tree-fold IPs of [2] where we allow for arbitrary matrices at each node. This yields a rather natural description of a generalized tree-fold IP. We refer to Section 4 for the precise definition.

In this setting we obtain the following result.

v) We present an algorithm for generalized tree-fold IPs with a running time that is roughly doubly exponential in the height of the tree (for a precise running time we refer to Lemma 10). With this algorithm we improve upon the algorithm by Chen and Marx [2], which has a running time involving a term that has a tower of τ exponents, where τ is the height of the tree.

vi) Using the tree-fold IP formulation of [2], this implies an fpt algorithm for the the traveling salesman problem on trees with m clients with running time $2^{2^{poly(K)}} \cdot |V|^{O(1)}$, where K is the longest tour of an optimal solution over all clients.

Notation. We use the following notation throughout this paper. For positive numbers $n, r, s, t \in \mathbb{N}$ and index $i = 1, \ldots, n$, let $A^{(i)} \in \mathbb{Z}^{r \times t}$, $B^{(i)} \in \mathbb{Z}^{s \times t}$ with $\left|\left|A^{(i)}\right|\right|_\infty, \left|\left|B^{(i)}\right|\right|_\infty \leq \Delta$ for some constant Δ. Columns of matrices are denoted with a lower index, i.e. the j-th column of the matrix $A^{(i)}$ is denoted by $A^{(i)}_j$, and so on. With $\log x$, we denote the logarithm to the basis 2 of some number x.

We will often subdivide the set of entries in a vector $y \in \mathbb{R}^{nt}$ or a vector $\mathcal{A}y \in \mathbb{R}^{r+ns}$ into *bricks*. A vector $y \in \mathbb{R}^{nt}$ will consist of n bricks with t variables each, i.e.

$$y^T = \left((y^{(1)})^T, \quad (y^{(2)})^T, \quad \ldots, \quad (y^{(n)})^T\right)$$

with the brick $y^{(i)} \in \mathbb{R}^t$ corresponding to the block $B^{(i)}$. A vector $g = \mathcal{A}y \in \mathbb{R}^{r+ns}$ will consist of $n+1$ bricks,

$$(\mathcal{A}y)^T = \left((g^{(0)})^T, \quad (g^{(1)})^T, \quad \ldots, \quad (g^{(n)})^T\right),$$

where the first brick $g^{(0)} \in \mathbb{R}^r$ consists of the first r entries and corresponds to the block row $(A^{(1)}, \ldots, A^{(n)})$ of $\mathcal{A}$, and every other block $g^{(i)}$, $i \geq 1$, consists of s entries and corresponds to the block $B^{(i)}$. We will always use upper indices with brackets when referring to the bricks, and the indices will coincide with the index of the block $B^{(i)}$ they correspond to (except brick $g^{(0)}$). A simple but crucial observation we will use several times is the following. If y is a cycle of $\mathcal{A}$, then each brick $y^{(i)}$ is already a cycle of the matrix $B^{(i)}$.

2 The norm of a Graver-basis element

In this section, we provide the details of the contributions i) and ii). We will make use of the following lemma of Steinitz [9, 21]. Here $\|\cdot\|$ denotes an arbitrary norm.

▶ **Lemma 1** (Steinitz Lemma). *Let $v_1, \ldots, v_n \in \mathbb{R}^m$ be vectors with $||v_i|| \leq \Delta$ for $i = 1, \ldots, n$. If $\sum_{i=1}^n v_i = 0$, then there is a reordering $\pi \in S_n$ such that for each $k \in \{1, \ldots, n\}$ the partial sum $p_k := \sum_{i=1}^k v_{\pi(i)}$ satisfies $||p_k|| \leq m\Delta$ (for the same norm $\|\cdot\|$).*

▶ **Lemma 2.** *Let $A \in \mathbb{Z}^{m \times n}$ be an integer matrix, let Δ be an upper bound on the absolute value of each component of A, and let $y \in \mathbb{Z}^n$ be an element of the Graver basis of A. Then $\|y\|_1 \leq (2m\Delta + 1)^m$.*

Proof. We define a sequence of vectors $v_1, \dots, v_{\|y\|_1} \in \mathbb{Z}^m$ in the following manner. If $y_j \geq 0$, we add y_j copies of the j-th column of A to the sequence, if $y_j < 0$ we add $|y_j|$ copies of the negative of column j to the sequence.

Clearly, the v_i sum up to zero and their ℓ_∞-norm is bounded by Δ. Using Steinitz, there is a reordering $u_{\pi(1)}, \dots, u_{\pi(\|y\|_1)}$ of this sequence s.t. each partial sum $p_k := \sum_{j=1}^{k} u_{\pi(j)}$ is bounded by $m\Delta$ in the l_∞-norm. Clearly,

$$|\{x \in \mathbb{Z}^m : \|x\|_\infty \leq m\Delta\}| = (2m\Delta + 1)^m .$$

Thus, if $\|y\|_1 > (2m\Delta + 1)^m$, then two of these partial sums are the same and we have a sequence $u_{\pi(k)} + \dots + u_{\pi(k+\ell)} = 0$. But then we can decompose y into two vectors corresponding to this sequence and the remaining vectors $u_{\pi(i)}$. This shows the claim. ◀

We will now apply the Steinitz lemma to bound the ℓ_1-norm of an element of the Graver basis of

$$\mathcal{A} = \begin{pmatrix} A^{(1)} & A^{(2)} & \dots & A^{(n)} \\ B^{(1)} & 0 & \dots & 0 \\ 0 & B^{(2)} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & B^{(n)} \end{pmatrix},$$

where $A^{(1)}, \dots, A^{(n)} \in \mathbb{Z}^{r \times t}$ and $B^{(1)}, \dots, B^{(n)} \in \mathbb{Z}^{s \times t}$ are arbitrary matrices. Lemma 2 shows that the ℓ_1-norm of an element of the Graver basis of a matrix $B^{(i)}$ is bounded by $(2\,s\Delta + 1)^s =: L_B$.

▶ **Lemma 3.** *Let y be a Graver-basis element of $\mathcal{A}$, then*

$$||y||_1 \leq L_B\,(2r\Delta L_B + 1)^r =: L_{\mathcal{A}}.$$

Proof. Let g be a Graver basis element of $B^{(i)}$. Note that as $||g||_1 \leq L_B$ and $\left|\left|A^{(i)}\right|\right|_\infty \leq \Delta$, the infinity-norm of the vector $A^{(i)}g$ is bounded by

$$\left|\left|A^{(i)}g\right|\right|_\infty \leq \Delta L_B. \tag{6}$$

Now consider a Graver basis element $y \in \mathbb{Z}^{nt}$ of $\mathcal{A}$ and split it according to the matrices $B^{(i)}$ into bricks, i.e. $y^T = ((y^{(1)})^T, \dots, (y^{(n)})^T)$ with each $y^{(i)} \in \mathbb{Z}^t$ being a cycle of $B^{(i)}$. Hence, each $y^{(i)}$ can be decomposed into the sum of Graver basis elements $y_j^{(i)}$ of $B^{(i)}$, i.e. $y^{(i)} = y_1^{(i)} + \dots + y_{N_i}^{(i)}$. Thus, we have a decomposition

$$\begin{aligned} 0 &= (A^{(1)}, \dots, A^{(n)})y \\ &= A^{(1)}y^{(1)} + \dots + A^{(n)}y^{(n)} \\ &= A^{(1)}y_1^{(1)} + \dots + A^{(1)}y_{N_1}^{(1)} + \dots + A^{(n)}y_1^{(n)} + \dots + A^{(n)}y_{N_n}^{(n)} \\ &=: v_1 + \dots + v_N \in \mathbb{Z}^r \end{aligned}$$

for some $N = \sum_{i=1}^{n} N_i$ and $||v_i||_\infty \leq \Delta L_B$ for $i = 1, \dots, N$, using (6). Now we apply Steinitz to reorder the v_i s.t. each partial sum is bounded by $r\Delta L_B$ in the l_∞-norm. Again, if two partial sums were the same, we could decompose y, thus the number N of vectors

v_i is bounded by $(2r\Delta L_B + 1)^r$. Each v_i is of the form $A^{(j)}y_k^{(j)}$ for some j and k with $\|y_k^{(j)}\|_1 \leq L_B$, hence

$$\begin{aligned} ||y||_1 &\leq L_B \left(2r\Delta L_B + 1\right)^r \\ &= L_{\mathcal{A}}, \end{aligned}$$

finishing the proof. ◀

As $L_B \in \mathcal{O}(s\Delta)^s$ in our case, this shows $L_{\mathcal{A}} \in \mathcal{O}(rs\Delta)^{(r+1)(s+1)}$, as stated in point *ii*) in the previous chapter.

3 Solving the Generalized n-fold IP

Given a feasible solution x of the IP (5) we now follow the principle that we outlined in Section 1.1. There exists an element y of the Graver basis of $\mathcal{A}$ and a positive integer $\lambda \in \mathbb{N}$ such that $x + \lambda y$ is feasible and reduces the gap to the optimum value by a factor of $1 - 1/(2n)$. Suppose that we know λ. With our bound on $\|y\|_1 \leq L_A$ we will find an augmenting vector of at least this quality by solving (a relaxation of) the following *augmentation IP*:

$$\begin{aligned} \max\, & c^T y \\ & \mathcal{A}y = 0 \\ & ||y||_1 \leq L_{\mathcal{A}} \\ l - z \leq\, & \lambda y \leq u - z \\ & y \in \mathbb{Z}^{nt} \end{aligned} \tag{7}$$

The vector y we compute might violate the condition $||y||_1 \leq L_{\mathcal{A}}$, but we will have that $c^T y$ is at least as big as the optimum value of (7).

▶ **Lemma 4.** *Let λ be a fixed positive integer. In time $nt\,(rs\Delta)^{\mathcal{O}(r^2s+rs^2)}$, we can find an integral vector y with $Ay = 0$, $l - z \leq \lambda y \leq u - z$ and $c^T y \geq c^T y^\star$, where $y^\star$ is an optimum solution for (7).*

Proof. As λ is fixed, it will be convenient to rewrite the bounds on the variables as

$$\begin{aligned} & l^\star \leq y \leq u^\star \quad \text{with} \\ & l_i^\star = \max\left\{ \left\lceil \frac{l_i - z_i}{\lambda} \right\rceil, -L_{\mathcal{A}} \right\} \\ & u_i^\star = \min\left\{ \left\lfloor \frac{u_i - z_i}{\lambda} \right\rfloor, L_{\mathcal{A}} \right\}. \end{aligned} \tag{8}$$

In particular, $u^\star < \infty$. First observe that for each $y \in \mathbb{Z}^{nt}$ with $||y||_1 \leq L_{\mathcal{A}}$, one has

$$||\mathcal{A}y||_\infty \leq \Delta L_{\mathcal{A}}. \tag{9}$$

We can decompose $y = (y^{(1)}, \ldots, y^{(n)})$ into bricks according to the matrices $B^{(i)}$, and $B^{(k)}y^{(k)} = 0$ has to hold independently of the other variables. Let $U \subseteq \mathbb{Z}^{r+s}$ be the set of integer vectors of infinity norm at most $\Delta L_{\mathcal{A}}$. To find an optimal $y^\star$ for the augmentation IP (7) we construct the following acyclic digraph. There are two nodes 0_{start} and 0_{target}, together with nt copies of the set U, arranged in n blocks of t layers as

$$U_1^{(1)}, \ldots, U_t^{(1)}, U_1^{(2)}, \ldots, U_t^{(2)}, \ldots, U_1^{(n)} \ldots U_t^{(n)},$$

where the k-th block will correspond to the matrix

$$M^{(k)} := \begin{pmatrix} A^{(k)} \\ B^{(k)} \end{pmatrix}$$

(and thus to the brick $y^{(k)}$ of y). Writing $M^{(k)}_j$ for the j-th column of the matrix $M^{(k)}$, the arcs are given as follows. There is an arc from 0_{start} to $v \in U^{(1)}_1$ if there is an integer y_1 such that

$$v = y_1 M^{(1)}_1 \quad \text{and} \quad l^\star_1 \leq y_1 \leq u^\star_1$$

holds. The weight of this arc is $c_1 y_1$.

For two nodes $u \in U^k_{i-1}$ and $v \in U^k_i$ of two consecutive layers in the same block, we add an arc (u, v) if there is an integer $y_{(k-1)t+i}$ such that

$$v - u = y_{(k-1)t+i} M^{(k)}_i \quad \text{and} \quad l^\star_{(k-1)t+i} \leq y_{(k-1)t+i} \leq u^\star_{(k-1)t+i}$$

holds, i.e. if we can get from u to v by adding the i-th column of $\binom{A^{(k)}}{B^{(k)}}$ multiple times. The weight is $c_{(k-1)t+i} \cdot y_{(k-1)t+i}$. It remains to define the arcs between two blocks. If we fix a path through the whole block $U^{(k)}_1, \ldots, U^{(k)}_t$, this corresponds to fixing a brick $y^{(k)}$. Note that $M^{(k)} y^{(k)}$ has to be zero in the last s components, since continuing with this path in the next block will not change the entries of $\mathcal{A}y$ corresponding to $B^{(k)}$ any more. Thus, for placing an arc between two nodes $u \in U^k_t$ and $v \in U^{k+1}_1$ in two consecutive layers of different blocks, also the constraints $u_{r+1} = \cdots = u_{r+s} = 0$ have to be fulfilled.

Finally, we add arcs from $u \in U^{(n)}_t$ to 0_{target} if there exists an integer y_{nt} such that

$$-u = y_{nt} M^{(n)}_t \quad \text{and} \quad l^\star_{nt} \leq y_{nt} \leq u^\star_{nt}$$

holds. Again, the weight is $c_{nt} y_{nt}$.

Clearly, a longest $(0_{start} - 0_{target})$-path corresponds to an optimum solution of the augmentation IP (7), hence it is left to limit the time needed to find such a path.

The out-degree of each node is bounded by $u^\star_i - l^\star_i \leq 2L_{\mathcal{A}} + 1$ using (8). Therefore, the number of arcs is bounded by

$$\begin{aligned} nt \cdot |U| \cdot (2L_{\mathcal{A}} + 1) &= nt \left(2\Delta L_{\mathcal{A}} + 1\right)^{r+s} (2L_{\mathcal{A}} + 1) \\ &\leq nt \left(2\Delta L_{\mathcal{A}} + 1\right)^{r+s+1} \\ &\leq nt \left(2\Delta L_B \left(2r\Delta L_B + 1\right)^r + 1\right)^{r+s+1} \\ &= nt \cdot \mathcal{O}(\Delta r)^{r^2 s + r s^2 + o(r^2 s + r s^2)} \mathcal{O}(s)^{r^2 + rs + r}. \end{aligned}$$

We can find a shortest path by a Breadth-First Search in time linear in the number of edges. ◀

In the following lemma we consider the value $\Gamma := \max_i(u_i - l_i)$. In the case $u < \infty$, we can estimate $\Gamma \leq 2^\varphi$ and obtain a fixed running time in combination with Lemma 4. However, if there are variables present that are not bounded from above, we will combine this lemma with the proximity result of the next Section 3.1 which allows us to introduce artificial upper bounds $u' < \infty$.

▶ **Lemma 5.** *Consider the n-fold IP (5) with $u < \infty$. Let $\Gamma := \max_i(u_i - l_i)$. Given an initial feasible solution, we can find an optimum solution of the IP by solving the augmentation IP (7) for a constant $\lambda \in \mathbb{Z}_+$ at most*

$$\mathcal{O}\left(nt \log(\Gamma) \left(\log(nt\Gamma) + \varphi\right)\right)$$

times, where φ is the logarithm of the largest number occurring in the objective function c.

Proof (sketch). As previously discussed, for every feasible z there exists a pair (λ, y) s.t. $z + \lambda y$ is feasible and reducing the gap to the optimum value by $1 - 1/(2\,nt)$. This leads to roughly $nt \log(nt\,||c||_\infty \Gamma)$ iterations. If we only guess values for λ that are a power of 2, we only lose a constant factor but are able to limit the number of guesses. We refer to the full version of the paper for details. ◀

3.1 Proximity for n-fold IPs

If no explicit upper bounds are given (i.e. $u_i = \infty$ for some indices i), we cannot bound the number of necessary augmentation steps directly. To overcome this difficulty, we will present a proximity result in this section, stating that for an optimum rational solution $x^\star$, there exists an optimum integral solution $z^\star$ with $||x^\star - z^\star||_1 \leq ntL_{\mathcal{A}}$.

With this proximity result, we can first compute an optimum LP solution $x^\star$, and then introduce artificial box constraints $l(x^\star) \leq z \leq u(x^\star)$, depending on $x^\star$, knowing that at least one optimum IP solution lies within the introduced bounds.

▶ **Lemma 6.** *Let $x^\star$ be an optimum solution to the LP relaxation of (5). There exists an optimum integral solution $z^\star$ to (5) with*

$$||x^\star - z^\star||_1 \leq ntL_{\mathcal{A}} = nt(rs\Delta)^{\mathcal{O}(rs)}.$$

Proof. Let $x^\star$ be an optimum vertex solution of the LP relaxation of (5) and $z^\star$ be an optimum (integral) solution of (5) that minimizes the l_1-distance to $x^\star$.

We say a vector y *dominates* a cycle y' if they are sign-compatible and $|y'_i| \leq |y_i|$ for each i. The idea is to show that if the l_1-distance is too large, we can find a cycle dominated by $z^\star - x^\star$ and either add it to $x^\star$ or subtract it from $z^\star$ leading to a contradiction in both cases. However, as $z^\star - x^\star$ is fractional, we cannot decompose it directly but have to work around the fractionality.

To this end, denote with $\lfloor x^\star \rceil$ the vector $x^\star$ rounded towards $z^\star$ i.e. $\lfloor x^\star_i \rceil = \lfloor x^\star_i \rfloor$ if $z^\star_i \leq x^\star_i$ and $\lfloor x^\star_i \rceil = \lceil x^\star_i \rceil$ otherwise. Denote with $\{x^\star\}$ the fractional rest i.e. $\{x^\star\} = x^\star - \lfloor x^\star \rceil$. Consider the equation

$$\mathcal{A}\,(z^\star - x^\star) = \mathcal{A}\,(z^\star - \lfloor x^\star \rceil) - \mathcal{A}\{x^\star\} = 0.$$

Consider the integral vector $\mathcal{A}\{x^\star\}$. For each index i, we will obtain an integral vector w_i out of $\{x^\star\}_i \mathcal{A}_i$ by rounding the entries suitably such that

$$\mathcal{A}\{x^\star\} = \sum_{i=1}^{nt} (\{x^\star\}_i \mathcal{A}_i) = w_1 + \cdots + w_{nt}.$$

To be more formal, fix an index j and let $a_1, \ldots, a_{nt}$ denote the j-th entry of the vectors $\{x^\star\}_i \mathcal{A}_i$. Define $f := \left(\sum_{i=1}^{nt} a_i - \lfloor a_i \rfloor\right) \in \mathbb{Z}_+$ as the sum of the fractional parts. We round up f of the fractional entries a_i, and we round down all other fractional entries. If some a_i is integral already, it remains unchanged. After doing this for each component j, we obtain the vectors w_i as claimed. As $||\{x^\star\}||_\infty \leq 1$, each vector w_i is dominated by either $\mathcal{A}_i$ or $-\mathcal{A}_i$, in particular it inherits the zero entries.

Define the matrix

$$\mathcal{A}' := (w_1, \ldots, w_{nt})\,.$$

After permuting the columns, the matrix $(\mathcal{A}, -\mathcal{A}')$ has n-fold structure with parameters $r, s, 2t$. As Lemma 3 does not depend on t, the Graver basis elements of $(\mathcal{A}, -\mathcal{A}')$ are bounded

by $L_{\mathcal{A}}$ as well. We can now identify

$$\mathcal{A}(z^\star - x^\star) = (\mathcal{A}, -\mathcal{A}') \begin{pmatrix} z^\star - \lfloor x^\star \rceil \\ \mathbf{1}_{nt} \end{pmatrix} = 0,$$

and decompose the integral vector $\binom{z^\star - \lfloor x^\star \rceil}{\mathbf{1}_{nt}}$ into Graver basis elements of l_1-norm at most $L_{\mathcal{A}}$. But if

$$ntL_{\mathcal{A}} < ||z^\star - x^\star||_1 \leq \left|\left| \begin{pmatrix} z^\star - \lfloor x^\star \rceil \\ \mathbf{1}_{nt} \end{pmatrix} \right|\right|_1,$$

we obtain at least $nt + 1$ cycles. As $||\mathbf{1}_{nt}||_1 = nt$, this grants a cycle $\binom{\bar{y}}{\mathbf{0}_{nt}}$ and hence a cycle $\bar{y}$ of $\mathcal{A}$.

Case 1: $c^T\bar{y} \leq 0$: As $\bar{y}$ is dominated by $z^\star - \lfloor x^\star \rceil$, removing cycle $\bar{y}$ from the solution gives a new solution $\bar{z} = z^\star - \bar{y}$ with $c^T\bar{z} \geq c^T z^\star$, which is closer to the fractional solution $x^\star$. However, this contradicts the fact that $z^\star$ was chosen to be a solution with minimal distance $||x^\star - z^\star||_1$.

Case 2: $c^T\bar{y} > 0$: As we rounded $x^\star$ towards $z^\star$ and $\bar{y}$ is dominated by $z^\star - \lfloor x^\star \rceil$, we can add $\bar{y}$ to $x^\star$ and obtain a better solution, contradicting its optimality. ◀

We are now able to state our main theorem.

▶ **Theorem 7.** *The generalized n-fold IP (5) can be solved in time*

$$n^2t^2\varphi \log^2 nt \cdot (rs\Delta)^{\mathcal{O}(r^2s+rs^2)} + \boldsymbol{LP}$$

where φ denotes the logarithm of the largest number occurring in the input, and $\boldsymbol{LP}$ denotes the time needed to solve the LP relaxation of (5).

We refer to the full paper for a detailed analysis of the running time.

4 Tree-Fold IPs

Given matrices $A^{(i)} \in \mathbb{Z}^{m_i \times n}$ and vectors $b^{(i)} \in \mathbb{Z}^{m_i}$ for $i = 1, \ldots, N$ and $c, l, u \in \mathbb{Z}^n$ for some $n, N \in \mathbb{Z}_+$, $m_1, \ldots, m_N \in \mathbb{Z}^+$. We consider the following IP consisting of a system of (systems of) linear equations

$$\begin{aligned} \max\, & c^T x \\ & A^{(1)}x = b^{(1)} \\ & \qquad \vdots \\ & A^{(N)}x = b^{(N)} \\ & l \leq x \leq u \\ & x \in \mathbb{Z}^n. \end{aligned} \tag{10}$$

We call (10) a *tree-fold IP*, if for every matrix $A^{(i)}$ there is an index set $S(i)$ containing all indices of the non-zero columns of $A^{(i)}$, i.e.

$$S(i) \supseteq \left\{ j \mid A^{(i)}_j \neq 0 \right\},$$

such that the following two conditions hold. For all i, j, the sets $S(i), S(j)$ are either disjoint, or one of the sets is contained in the other. There is a matrix $A^{(i_0)}$ for which $S(i_0)$ contains all column indices.

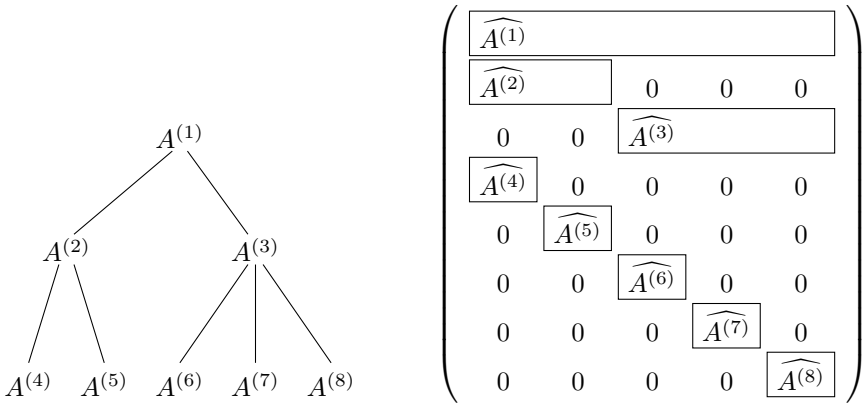


Figure 1 A matrix tree T and the induced tree-fold matrix $\mathcal{T}$, where $\widehat{A^{(i)}}$ denotes the part of $A^{(i)}$ that consists of the columns with index in $S(i)$. Note that if $A^{(1)}$ was not present, the set of columns of $\mathcal{T}$ could be bipartitioned into two sets orthogonal to each other.

Intuitively, the partial ordering induced by the sets $S(i)$ forms a tree T on the matrices $A^{(i)}$ (if the arcs stemming from transitivity are omitted). The root of this tree is the matrix $A^{(i_0)}$ with the largest set $S(i_0)$.

Analogously to our n-fold results, we will provide an upper bound on the l_1-norm of Graver basis elements of tree-fold matrices, together with a proximity result for optimum solutions. This will be sufficient to obtain an algorithm with a comparable running time.

Throughout this section, T will denote a tree as in Figure 1, we will denote the depth by τ and enumerate the layers starting at the deepest leaves (the leaves are not necessarily all in the same layer). The whole matrix induced by a tree-fold IP will be denoted by $\mathcal{T}$. This is, the IP (10) can be rewritten as

$$\begin{aligned} \max\, & c^T x \\ & \mathcal{T}x = b \\ & l \le x \le u \\ & x \in \mathbb{Z}^n \end{aligned} \tag{11}$$

▶ **Lemma 8.** *Let $\mathcal{T}$ be a tree-fold matrix where the corresponding matrix tree T has τ layers. Let the matrices of layer i have at most s_i rows and define $s = \prod_{i=1}^{\tau}(s_i+1)$ and $\Delta := ||\mathcal{T}||_\infty$. Then the Graver basis elements of $\mathcal{T}$ are bounded in their l_1-norm by*

$$L_\tau \le (3s\Delta)^{s-1}\,.$$

Proof (sketch). We enumerate the layers of T starting at the layer with the deepest leaves. We prove the claim by induction on the number τ of layers in the tree T. First observe that for $\tau = 1$, the claim follows by Lemma 2, as

$$L_1 \le (2s_1\Delta + 1)^{s_1} \le (3s\Delta)^{s-1}.$$

For the induction step, note that every child matrix $A^{(i)}$ of the root in T can be seen as the root matrix of a subtree T_i in T of depth $\tau - 1$ with at most $s_1, \dots, s_{\tau-1}$ rows in the corresponding layers. More formal, delete the root $A^{(1)}$ in T and let T_i be the connected component $A^{(i)}$ is in. Write

$$\widetilde{s} = \prod_{i=1}^{\tau-1}(s_i+1),$$

i.e. $s = \widetilde{s}(s_\tau + 1)$. By induction, we know that all Graver basis elements of the subtree-fold IPs $\mathcal{T}_i$ induced by T_i are bounded by

$$L_{\tau-1} \leq (3\widetilde{s}\Delta)^{\widetilde{s}-1} \leq (3s\Delta)^{\widetilde{s}-1}. \tag{12}$$

The rest of the induction step works similar to the proof of Lemma 3. We pick a cycle y of $\mathcal{T}$, decompose it into Graver basis elements for the subtree-fold matrices $\mathcal{T}_i$ and obtain a Steinitz sequence of vectors bounded by the induction hypothesis. However, due to space constraints, we omit the remaining part of this proof and refer to the full version of the paper. ◀

The following lemma states a proximity result for tree-fold IPs in the flavour of Lemma 6 for n-fold IPs. The proof uses that the bound in Lemma 8 only depends on the shape of the matrix but is independent of the number of columns in each block $A^{(i)}$, precisely as in Lemma 6. We refer to the full version of the paper for details on the proof.

▶ **Lemma 9.** *Let $\mathcal{T}$ be a matrix of tree-fold structure corresponding to the IP (10) and let $x^\star$ be an optimum solution to the LP relaxation of (10). There exists an optimum integral solution $z^\star$ to (10) with*

$$||x^\star - z^\star||_1 \leq nL_\tau.$$

We conclude with the following theorem that states the running time of our algorithm to solve a tree-fold IP. For a detailed analysis of the running time, we refer to the full version of the paper.

▶ **Theorem 10.** *Let $\mathcal{T}$ be of tree-fold structure with infinity-norm Δ and corresponding tree T. Let τ denote the number of layers of T and let the matrices of layer i have at most s_i rows.*

Define $s = \prod_{i=1}^{\tau}(s_i + 1)$ and $\sigma = \sum_{i=1}^{\tau} s_i$. Let n denote the number of columns of $\mathcal{T}$ and $l, u \in (\mathbb{Z} \cup \{\infty\})^n$. We can solve the IP (11),

$$\begin{aligned} \max\, & c^T x \\ & \mathcal{T}x = b \\ & l \leq x \leq u \\ & x \in \mathbb{Z}^n \end{aligned}$$

in time

$$n^2\varphi \log^2 n (s\Delta)^{\mathcal{O}(\sigma s)} + \boldsymbol{LP}$$

where φ denotes the logarithm of the largest number occurring in the input, and $\boldsymbol{LP}$ denotes the time needed to solve the LP relaxation of (11).

References

1 Stephan Artmann, Robert Weismantel, and Rico Zenklusen. A strongly polynomial algorithm for bimodular integer linear programming. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1206–1219. ACM, 2017.

2 Lin Chen and Dániel Marx. Covering a tree with rooted subtrees–parameterized and approximation algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2801–2820. SIAM, 2018.

3 Lin Chen, Dániel Marx, Deshi Ye, and Guochuan Zhang. Parameterized and approximation results for scheduling with a low rank processing time matrix. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 22:1–22:14, 2017.

4 William Cook, Jean Fonlupt, and Alexander Schrijver. An integer analogue of caratheodory's theorem. *Journal of Combinatorial Theory, Series B*, 40(1):63–70, 1986.

5 Jesús A De Loera, Raymond Hemmecke, and Matthias Köppe. *Algebraic and geometric ideas in the theory of discrete optimization*. SIAM, 2012.

6 Jesús A De Loera, Raymond Hemmecke, Shmuel Onn, and Robert Weismantel. N-fold integer programming. *Discrete Optimization*, 5(2):231–241, 2008.

7 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the Steinitz lemma. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 808–816. SIAM, 2018.

8 Jack E Graver. On the foundations of linear and integer linear programming i. *Mathematical Programming*, 9(1):207–226, 1975.

9 Victor S Grinberg and Sergey V Sevast'yanov. Value of the Steinitz constant. *Functional Analysis and Its Applications*, 14(2):125–126, 1980.

10 Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. N-fold integer programming in cubic time. *Mathematical Programming*, pages 1–17, 2013.

11 Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. Empowering the configuration-ip - new PTAS results for scheduling with setups times. *CoRR*, abs/1801.06460, 2018. URL: `http://arxiv.org/abs/1801.06460`.

12 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

13 Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *Journal of Scheduling*, pages 1–11, 2017.

14 Dušan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial n-fold Integer Programming and Applications. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 54:1–54:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

15 Dušan Knop, Martin Koutecký, and Matthias Mnich. Voting and bribing in single-exponential time. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 46:1–46:14, 2017.

16 Hendrik W Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

17 George L Nemhauser and Laurence A Wolsey. Integer and combinatorial optimization. interscience series in discrete mathematics and optimization. *ed: John Wiley & Sons*, 1988.

18 Shmuel Onn. Nonlinear discrete optimization. *Zurich Lectures in Advanced Mathematics, European Mathematical Society*, 2010.

19 Christos H Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, 1981.

20 Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

21 Ernst Steinitz. Bedingt konvergente reihen und konvexe systeme. *Journal für die reine und angewandte Mathematik*, 143:128–176, 1913.

On the Probe Complexity of Local Computation Algorithms

Uriel Feige[1]
Weizmann Institute of Science, Rehovot, Israel
uriel.feige@weizmann.ac.il

Boaz Patt-Shamir[2]
Tel Aviv University, Tel Aviv, Israel
boaz@tau.ac.il

Shai Vardi[3]
California Institute of Technology, Pasadena, CA, USA
svardi@caltech.edu

Abstract

In the Local Computation Algorithms (LCA) model, the algorithm is asked to compute a part of the output by reading as little as possible from the input. For example, an LCA for coloring a graph is given a vertex name (as a "query"), and it should output the color assigned to that vertex after inquiring about some part of the graph topology using "probes"; all outputs must be consistent with the same coloring. LCAs are useful when the input is huge, and the output as a whole is not needed simultaneously. Most previous work on LCAs was limited to bounded-degree graphs, which seems inevitable because probes are of the form "what vertex is at the other end of edge i of vertex v?". In this work we study LCAs for unbounded-degree graphs. In particular, such LCAs are expected to probe the graph a number of times that is significantly smaller than the maximum, average, or even minimum degree. We show that there are problems that have very efficient LCAs on any graph - specifically, we show that there is an LCA for the weak coloring problem (where a coloring is legal if every vertex has a neighbor with a different color) that uses $\log^* n + O(1)$ probes to reply to any query. As another way of dealing with large degrees, we propose a more powerful type of probe which we call a *strong* probe: given a vertex name, it returns a list of its neighbors. Lower bounds for strong probes are stronger than ones in the edge probe model (which we call *weak* probes). Our main result in this model is that roughly $\Omega(\sqrt{n})$ strong probes are required to compute a maximal matching.

Our findings include interesting separations between closely related problems. For weak probes, we show that while weak 3-coloring can be done with probe complexity $\log^* n + O(1)$, weak 2-coloring has probe complexity $\Omega(\log n / \log\log n)$. For strong probes, our negative result for *maximal* matching is complemented by an LCA for $(1-\epsilon)$-approximate *maximum* matching on regular graphs that uses $O(1)$ strong probes, for any constant $\epsilon > 0$.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Local computation algorithms, sublinear algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.50

[1] Supported in part by the Israel Science Foundation (grant No. 1388/16). Work partly done in Microsoft Research, Herzeliya, Israel.
[2] Supported in part by the Israel Science Foundation (grant No. 1444/14).
[3] Supported in part by the I-CORE in Algorithms Postdoctoral Fellowship, the Linde Foundation and NSF grants CNS-1254169 and CNS-1518941. Part of the research was carried out when Shai was a postdoctoral researcher at the Weizmann Institute of Science.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 50; pp. 50:1–50:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version A full version of the paper can be found at [9], https://arxiv.org/abs/1703.07734.

Acknowledgements We thank Noga Alon for an enlightening discussion and the anonymous reviewers for their useful comments.

1 Introduction

In classical algorithmic models, an algorithm is given an input and is required to compute an output. When dealing with truly massive data, such as the Internet, just reading the entire input may be impractical. The model of *local computation algorithms* (LCAs), as studied by Rubinfeld et al. [33], proposes the following approach. An algorithm in the LCA model is required to produce only a part of the output specified by a "query," and is expected to access only a small part of the input (without any pre-processing) using some simple "probes." For example, an LCA for maximal independent set (MIS) is given a vertex ID as a query, and is expected to return a "yes/no" answer, indicating whether the queried vertex is or is not in the MIS; all replies to queries must be consistent with the same MIS. To do that, the LCA can use probes of the form "which vertex is the ith neighbor of vertex v?", where v and i are the probe arguments. One of the main goals in LCA design is to minimize the number of probes required to produce an answer to a query.

In this paper we consider LCAs for graph problems. Almost exclusively, known LCAs for graph problems are efficient only for graphs of bounded degrees (with the notable exception of [20], which gives LCAs with polylog probe complexity for graphs with polylog degree). This may appear inevitable, because edge probes don't allow sublinear solutions to even learn a complete neighborhood of a linear-degree vertex. Our goal in this paper is to understand what *can* be done efficiently for graphs with unbounded degrees. We give two types of answers. First, we point to problems that admit efficient solutions for any graph. In particular, we give efficient LCAs for weak coloring [29], where the goal is to color vertices so that each vertex has a neighbor of a different color. Weak coloring has played a key role in the study of distributed algorithms e.g., [29, 10], in part due to its applications to resource allocations in distributed settings [25]. Hence it is natural that we study it in the context of LCAs. Moreover, this study has turned out to be worthwhile, because the results were unexpected: in our model there is a separation between weak 2-coloring and weak 3-coloring that was not observed in other models.

As another way of dealing with large degrees, we propose a more powerful probe model, where probing a vertex returns a list of all its neighbors. We call such probes "strong," as opposed to the "weak" edge probes. Lower bounds in this model consider the number of connections the LCA needs to make in order to probe the graph, even if communication along these channels is unbounded, and are stronger than lower bounds in the weak probe model. Strong probes can be thought of as an intermediate model that lies between the weak probe model and the distributed $\mathcal{LOCAL}$ model, and helps clarify the sources of differences between these two extremes. Our main negative result in this model is that approximately $\Omega(\sqrt{n})$ strong probes are required to compute a maximal matching.

Our results include the following. Let *WP* and *SP* stand for weak and strong probes, respectively and let n denote the number of nodes in the underlying graph. We give tight upper and lower bounds for weak 3-coloring. Our algorithm is deterministic and uses weak probes, whereas our lower bound holds also for randomized LCAs that may use strong probes.

▶ **Theorem 1.** *There exists a deterministic WP LCA for weak* 3*-coloring that uses* $\log^* n + O(1)$ *probes.*

▶ **Theorem 2.** *Every (randomized or deterministic) SP LCA for weak* 3*-coloring of the cycle graph requires* $\Omega(\log^* n)$ *probes.*

We describe a deterministic WP LCA for weak 2-coloring, and give a matching lower bound for graphs with maximal degree $d = O\left(\frac{\log n}{\log\log n}\right)$.

▶ **Theorem 3.** *There exists a deterministic WP LCA for weak* 2*-coloring that uses* $\log^* n + 2d_v + O(1)$ *weak probes, where* d_v *is the degree of the queried vertex.*

▶ **Theorem 4.** *Any deterministic WP LCA for weak* 2*-coloring* d*-regular graphs with* $d = O\left(\frac{\log n}{\log\log n}\right)$ *requires at least* $d/2$ *probes.*

We design a randomized LCA for weak 2-coloring, whose probe complexity is independent of the maximal degree and show how it can be implemented in both the strong and weak probe models.

▶ **Theorem 5.** *There exists a randomized WP LCA for weak* 2*-coloring that uses* $\Theta\left(\frac{\log^2 n}{\log\log n}\right)$ *probes, and a randomized SP LCA for weak* 2*-coloring that uses* $\Theta\left(\frac{\log n}{\log\log n}\right)$ *probes.*

We give a lower bound for vertex cover in the strong probe model. Specifically, we show that for high degree graphs, many strong probes are necessary to approximate a minimal vertex cover to any interesting precision.

▶ **Theorem 6.** *For any* $\epsilon < \frac{1}{2}$*, any randomized SP LCA that computes a vertex cover whose size is a* $(\frac{1}{2}n^{1-2\epsilon})$*-approximation to the size of the minimal vertex cover requires at least* ϵn^{ϵ} *probes.*

A corollary of Theorem 6 is the following.

▶ **Corollary 7.** *Any SP LCA for maximal matching on arbitrary graphs requires* $n^{1/2-o(1)}$ *probes.*

We describe an LCA that finds a matching that is a $(1-\epsilon)$-approximation to the maximum matching, for regular graphs, using a constant number of probes.

▶ **Theorem 8.** *There exists an SP LCA that finds a* $(1-\epsilon)$*-approximate maximum matching in expectation on* d*-regular graphs that uses* $\epsilon^{-O(\epsilon^{-2})}$ *probes per query.*

Finally, we show that for graphs with sufficiently high girth and degree, polynomially (in ϵ^{-1}) many strong probes suffice.

▶ **Theorem 9.** *There exists an SP LCA that finds a* $(1-\epsilon)$*-approximate maximum matching in expectation on* d*-regular graphs of girth* g*, with* $d \geq \epsilon^{-1}$ *and* $g \geq \epsilon^{-3}$*, that uses* $O(\epsilon^{-7})$ *probes per query.*

1.1 Overview of Our Techniques

Our main result in this abridged version is the proof of Theorem 6. We construct a family of bipartite graphs in which a large subset of vertices have "almost" the same view at distance 2. Exactly one of these vertices, v_0, needs to be added to the vertex cover; however, there are many vertices for which a small number of strong probes does not suffice in order to

verify that they are not v_0, and hence they must add themselves to the vertex cover. In our construction we use vertex naming schemes based on low degree polynomials to ensure that certain vertices do not share many neighbors. This result gives a separation between the SP model (and hence also the WP model) and the $\mathcal{LOCAL}$ model, as it is possible to compute a maximal matching in the $\mathcal{LOCAL}$ model in $O(\log n)$ rounds w.h.p. [24].

In the full version of the paper [9] we discuss deterministic and randomized methods of sampling a "parent" for each vertex. No matter how the parents are chosen, the resulting graph is a disjoint set of directed subgraphs, each one containing exactly one cycle. The main technical content of here is the analysis of the diameter of these subgraphs, depending on the choice of parent selection scheme.

We then address the problem of weak 3-coloring. Our LCA (the proof of Theorem 1) is based on the following approach. Given an arbitrary graph, the directed subgraphs formed by the parent relation (as above) span all vertices, and each of their connected components has at least two vertices. Hence any weak coloring of every component separately induces a weak coloring of the whole graph. Each component has one cycle, but it turns out that the 3-coloring algorithm for rooted trees of Goldberg, Plotkin and Shannon [13] can be adapted in order to legally 3-color (and hence also weakly 3-color) such a component. When implemented as an LCA, the upper bound of $\log^* n + O(1)$ on the number of probes follows from a similar upper bound on the number of rounds of the (modified) algorithm of [13]. To prove a nearly matching lower bound (Theorem 2) we use a reduction to the lower bound of Naor [28] (extending [21]) for distributed algorithms that legally 6-color a cycle. Adapting lower bounds from the distributed setting to the LCA setting also involves an argument of Göös et al. [15].

We show how to augment the previous algorithm in order to reduce the number of colors to 2, thus proving Theorem 3. It takes only one more probe to transform the weak 3-coloring to a weak 2-coloring that is legal for all vertices except for those vertices that do not serve as parents (which we refer to as *leaves*). The final step involves changing the colors of (some) leaves. In order to determine whether a vertex v is a leaf, we probe all of its neighbors, making the probe complexity linear in the degree of the queried vertex.

One natural method of proving lower bounds for LCAs is by reduction to the distributed $\mathcal{LOCAL}$ model, as was done in [15] (and as we do in the proof of Theorem 2). The relationship between LCAs and distributed algorithms has been studied before (e.g., [6, 31, 32, 33]) – given a distributed algorithm to a problem that takes t rounds, one immediately obtains an LCA that uses $O\left(d^t\right)$ probes (where d is the maximal degree), by probing all nodes within distance t. The inverse reduction doesn't work, as an LCA may probe remote (disconnected) nodes. Consider, for example, the following artificial problem: each vertex has to color itself blue if the node with ID 1 has an odd number of neighbors, and red otherwise. An LCA for this problem needs a single probe, while a distributed algorithm requires time proportional to the graph's diameter. Göös et al. [15] show that for many natural problems, probing remote vertices does not help. But even if we consider only probes to neighbors of discovered vertices, the best lower bound we can hope for using such a reduction is the distributed *time* lower bound: a lower bound of t rounds in the distributed model implies a lower bound of t probes in the LCA model (recall that an upper bound of t rounds in the distributed model implies an upper bound of $O\left(d^t\right)$ probes!). This suggests that we may need new tools to obtain stronger lower bounds. In the proof of Theorem 4, we iteratively construct families of d-regular graphs, where graphs in family F_i (for $i \leq d/2$) contain roughly d^i vertices, and we show that weak 2-coloring of graphs in family F_i requires at least i deterministic weak probes. Every graph G in family F_i is composed of d disjoint copies of graphs $H_1, \ldots, H_d$ from family

F_{i-1} and two auxiliary vertices a_i and b_i. From each graph H_j a single edge is removed, and instead one of its endpoints is connected to a_i and the other to b_i. Intuitively, it is reasonable to expect that a_i cannot decide on a color before it knows the color of at least one of its neighbors. Likewise, it is reasonable to expect that each neighbor, being a member of a graph in F_{i-1}, requires (by an induction hypothesis) $i-1$ probes. The combination of these two non-rigorous claims would imply that a_i requires at least i probes (one to determine the name of one of its neighbors, and $i-1$ probes so as to determine the color of that neighbor). Turning this informal intuition into a rigorous proof is nontrivial, and this is the main content of our proof of Theorem 4.

We design a randomized LCA for weak 2-coloring. A key aspect in our randomized LCA is that each vertex first chooses a random temporary color. This induces a weak 2-coloring for most vertices of the graph: each vertex whose temporary color differs from the temporary color of a neighbor can keep its color. Extending this weak 2-coloring to the remaining vertices is done by associating a parent with each vertex, with the intended goal that the color of the vertex will differ from the color of the parent (determining the color of the parent uses an inductive process). This aspect has several different implementations, leading to different probe complexities. Namely, for an arbitrary parent choice, the number of strong probes is $\Theta(\log n)$. If we implement the arbitrary choice using weak probes, the probe complexity is $\Theta(\log^2 n)$. For a more clever randomized choice of parent, we get that the strong probe complexity is $\Theta\left(\frac{\log n}{\log\log n}\right)$, and the weak probe complexity is $\Theta\left(\frac{\log^2 n}{\log\log n}\right)$. All these bounds on probe complexity hold with high probability.

We note that our results do not prove a separation between the complexities of deterministic and randomized WP LCAs for weak 2-coloring (although we conjecture that there is one), as our lower bound that is linear in the degree is proved only for regular graphs of degree at most $O\left(\frac{\log n}{\log\log n}\right)$, and our upper bound for randomized WP LCAs for weak 2-coloring is $O\left(\frac{\log^2 n}{\log\log n}\right)$.

Randomized LCAs generally use a pseudo-random generator in order to limit the number of bits that they use, while ensuring consistency (e.g., [1, 20, 32]). In order to explore the theoretical limitations of the probe complexity of LCAs, we assume that our randomized LCAs have unbounded access to random bits. Nevertheless, we show that one can implement the randomized WP LCA for weak 2-coloring (the one using the arbitrary parent choice scheme) using a pseudo-random generator with a seed of length $O(\log n)$.

We give an LCA for approximate maximum matching in regular graphs. We first describe an LCA for $(1-\epsilon)$ matching on graphs of degree bounded by d, that uses at most $\left(d+\frac{1}{\epsilon}\right)^{O\left(\epsilon^{-2}\right)}$ probes per query. (Alternatively, if we wish to have a better dependency on ϵ and are willing to have a dependency on n, then an LCA of Even, Medina and Ron [6] has probe complexity $d^{O\left(\frac{1}{\epsilon}\right)}+O\left(\frac{1}{\epsilon^2}\right)\log^* n$.) Our LCA $\mathcal{A}$ is a simple variation on a randomized LCA of Yoshida, Yamamoto and Ito [40]: whereas [40] does not limit the number of probes used by their LCA but instead analyze and provide upper bounds on the expected number of probes used by their LCA (expectation taken both other choice of random edge and randomness of the LCA), we run essentially the same LCA, but with a strict upper bound on the number of probes. This upper bound is a factor of $\frac{d}{\epsilon}$ larger than the expectation. Markov's inequality implies that this gives a $(1-\epsilon)$-approximation to the maximum matching in expectation.

To obtain an LCA for a d-regular graph G, if $d<\frac{1}{\epsilon^2}$ we use LCA $\mathcal{A}$. If $d>\frac{1}{\epsilon^2}$, we sparsify the graph: for some universal constant c (independent of ϵ), each edge remains in the graph with probability $\frac{c}{d\epsilon}$, and then all vertices that still have degree higher than $\frac{2c}{\epsilon}$ are removed from the graph. This results in a graph G' of degree bounded by $\frac{2c}{\epsilon}$. Every matching in G'

is a matching in G. Moreover, we show that the expected size of a maximum matching in G' (expectation taken over choice of random sparsification) is at least $(1 - \frac{\epsilon}{2})$ times the size of the maximum matching in G. Hence it suffices to find a $(1 - \frac{\epsilon}{2})$-approximate maximum matching in the bounded degree graph G', and it will serve as a $(1-\epsilon)$-approximate maximum matching in G. A sufficiently large matching in G' can be found by $\mathcal{A}$, using $\left(\frac{1}{\epsilon}\right)^{O\left(\frac{1}{\epsilon^2}\right)}$ probes to G' per query. To implement $\mathcal{A}$ on G', but using only probes to G, we show that each strong probe to G' can be simulated by $O(1/\epsilon)$ strong probes to G.

We show that we can find a $(1 - \epsilon)$-approximation to the maximum matching for regular graphs with sufficiently high degree and girth using polynomially (in $1/\epsilon$) many probes. Gamarnik and Goldberg [11] show that the randomized greedy algorithm finds a $(1 - \epsilon)$ approximation to the maximum matching on regular graphs with sufficiently high degree and girth. Similarly to the previous result, if the vertex degrees are sufficiently small (say, below $\frac{1}{\epsilon^3}$), we can use an LCA of [40] (an implementation of the randomized greedy algorithm that uses in expectation $O(d)$ probes in graphs of maximum degree d), while placing a strict upper bound on the number of probes (this upper bound is a factor of $\frac{1}{\epsilon^{O(1)}}$ larger than the expectation). If the degrees are large, our approach is once again to sparsify the graph G prior to using the LCA of [40] (modified to have a strict upper bound on the number of probes). Unfortunately, the resulting graph G' is only nearly regular but not actually regular. This requires us to extend the result of [11] from regular graphs to nearly regular graphs. We do so without relying on the proofs of [11], by the following approach. We add imaginary edges to G', making it regular (while maintaining high girth). The LCA now runs on a regular graph, and hence the bounds of [11] apply. The new problem that arises is that the matching that is output by the LCA might contain imaginary edges. However, we prove that the expected fraction of imaginary edges in that matching is similar to their fraction within the input graph (our proof uses both the high girth assumption and the fact that the randomized greedy algorithm is local in nature). This, combined with the fact that the fraction of imaginary edges in the input graph was small (because G' is nearly regular), implies that the imaginary edges can be discarded from the solution without significantly affecting the approximation ratio.

1.2 Related Work

Measures

There are several criteria by which one can measure the performance of LCAs. Rubinfeld et al. [33] focus on the time complexity of LCAs: how long it takes to reply to a query; Alon et al. [1] emphasize the space complexity, in particular, the length of the random seed used (randomized LCAs need a global random seed to ensure consistency). Mansour, Patt-Shamir and Vardi [26] introduce a unified model, that takes into account all four complexity measures: *probe complexity*, *time complexity* (per query) and space complexity, divided into *enduring memory* (in all known LCAs, this includes only the random seed) and *transient space* (the computational space used per query). They show that it is possible to obtain LCAs for which all of these are independent of n for certain problems, such as a $(1 - \epsilon)$ approximation to a maximal acyclic subgraph, using $d^{O(1/\epsilon)}$ probes, where d is the maximal degree of the graph. LCAs that do not use any enduring memory are called *stateless* [6]. Indeed, the deterministic algorithms in this paper are stateless. Another property that is considered desirable in LCAs is *query-obliviousness* [33]: the property that the replies to different queries do not depend on the order in which the queries are given. Again, the LCAs of this paper are all query-oblivious.

LCAs and high-degree graphs

As mentioned above, most known results assume bounded degrees. For example, Mansour et al. [27] describe LCAs for maximal matching and other problems that use polylogarithmic (in the size of the graph; exponential in the degree) time and space when the degrees of the graph are bounded by a constant. Even, Medina and Ron [6], focusing on probe complexity, give deterministic LCAs for MIS, maximal matching and $(d+1)$-coloring for graphs of degree bounded by a constant d, which use $d^{O(d^2)}\log^* n$ probes. Fraigniaud, Heinrich and Kosowski [10] investigate local conflict coloring, a general distributed labeling problem, and use their results to improve the probe complexity of $(d+1)$-vertex coloring to approximately $d^{O(\sqrt{d})}\log^* n$ probes.

Some papers allow for slightly super-constant degrees: Levi, Rubinfeld and Yodpinyanee [20] give LCAs for MIS and maximal matching for graphs of degree $2^{O(\sqrt{\log\log n})}$, using an improvement of Ghaffari [12]. Reingold and Vardi [32] give LCAs for MIS, maximal matching and other problems that apply to graphs that are sampled from some distribution. This limitation allows them to address graphs with higher maximal degree, as long as the average degree is $O(\log\log n)$, and the tail of the distribution is sufficiently light. If we restrict ourselves to LCAs that use polylogarithmic time and space, the approximate maximum matching LCA of [20] accommodates graphs of polylogarithmic degree. Levi, Ron and Rubinfeld [19] describe an LCA that constructs spanners using a number of probes polynomial in d.

Lower bounds

There are few explicit impossibility results LCAs. Göös et al. [15] show that any LCA for MIS requires $\Omega(\log^* n)$ probes, by showing that probing vertices that have not yet been discovered is not useful. This implies that, on a ring, the number of probes that an LCA needs to make is "roughly the same" as the number of rounds required by a distributed LOCAL algorithm, implying that the lower bounds of Linial [21] and Naor [28] hold for LCAs as well. Levi, Ron and Rubinfeld [19] show that an LCA that determines whether an edge belongs to a sparse spanning subgraph requires $\Omega(\sqrt{n})$ probes. Feige, Mansour and Schapire [8], adapting a lower bound from the property testing literature [14], show that approximating the minimum vertex cover in bounded degree bipartite graphs within a ratio of $1+\epsilon$ (for some explicit fixed $\epsilon > 0$) cannot be done with $o(\sqrt{n})$ probes.

Weak coloring

Weak coloring was introduced by Naor and Stockmeyer [29]. They give a LOCAL distributed algorithm that requires $\log^* d + O(1)$ rounds for weak 2-coloring graphs of maximal degree d, assuming all degrees are odd. In contrast, they show that there is no constant time LOCAL algorithm for weak c-coloring all graphs with vertices with even degrees, for constant c. Our deterministic weak 2-coloring LCA (Theorem 3) uses $\log^* n + O(d_v)$ probes, but when cast within the LOCAL model it takes $\log^* n + O(1)$ rounds (independently of d_v).

Additional LCA background

LCAs are not restricted to graphs. Well known examples include *locally-decodable codes* (LDCs) (e.g., [18, 39]) and local reconstruction (e.g., [17, 34]). LDCs are error-correcting codes that allow a single bit of the original message to be decoded with high probability by querying a small number of bits of a codeword. Local reconstruction involves recovering the

value of a function f for a particular input x given oracle access to a closely related function g. LCAs have recently been applied to solving convex problems in a distributed fashion [22]. Traditional methods for solving distributed optimization, such as iterative descent methods (e.g., [23]) and consensus methods (e.g., [4]), require global communication, and any edge failure or lag in the system immediately affects the entire solution, by delaying computation or causing it not to be computed at all; furthermore, if the network changes in a small way, the entire solution needs to be recomputed. If an LCA is used, most of the system remains unaffected by local changes or failures. Hence LCAs can be used to make systems more robust to edge failures, lag, and dynamic changes. LCAs have also been used in the context of mechanism design [16], machine learning [8], and designing distributed algorithms [7]. There are other situations when LCAs may be useful - say we wish to perform some computation on each of the vertices of an MIS of some huge graph. LCAs allow us to be able to begin work on some vertices before the entire MIS is computed, and guarantee that the local replies to the queries will be consistent with the same global solution, that will be available at some point in the future.

LCAs can also be used as subroutines in approximation algorithms e.g., [5, 31, 30, 40]. The goal of such algorithms is to output an approximation to the size of the solution to some combinatorial problem (such as Vertex Cover, Maximum Matching, Minimum Spanning Tree), in time sublinear in the input size. In particular, if one has an LCA whose running time is t that solves some problem, one can obtain an approximation to the size of the solution (with some constant probability) by executing this LCA on a sufficiently large (but constant) number of vertices k chosen uniformly at random, to obtain an approximation algorithm whose running time is kt (see e.g., [30, 36] for more details).

The concept of LCAs is related to but should not be confused with local algorithms as in [2, 3, 35]. The difference is that these local algorithms do not require the output for different probes to satisfy a global consistency property, but rather to satisfy some local criteria. For example, the goal might be for each vertex to output a small dense subgraph that contains it, without requiring two different vertices to agree on whether they share the same dense subgraph or not.

1.3 Paper Outline

Due to space restriction, we include a single result with complete proof in this version. See [9] for the full version of this paper.

2 Preliminaries

We denote the set of integers $\{1, 2, \ldots, n\}$ by $[n]$. All logarithms are base 2. Our input is a simple undirected graph $G = (V, E)$, $|V| = n$, in which every vertex has an ID and all IDs are distinct. For simplicity, we assume that the IDs are taken from the set $[n]$. The neighborhood of a vertex v, denoted $N(v)$, is the set of vertices that share an edge with v: $N(v) = \{u : (u, v) \in E\}$. The *degree* of a vertex v, is $|N(v)|$. The *girth* of G, denoted girth(G) is the length of the shortest cycle in G.

LCAs

Our definition of LCAs is slightly different from previous definitions in that it focuses on probe complexity. We do this so as not to introduce unnecessary notation. See [26, 36] for definitions that also take into account other complexity measures.

▶ **Definition 10** (Probe)**.** We assume that the input graph is represented as a two dimensional n by $d+1$ array, where d is the maximum degree. Rows are labeled from 1 to n by the vertex names. For any v, the cell $(v,0)$ specifies the degree d_v of v, the cell (v,j) for $1 \leq j \leq d_v$ specifies the name of the neighbor connected to v's jth port. Cells (v,j) for $d_v < j \leq d$ contain 0. We define strong and weak probes as follows.

- A *strong probe* (SP) specifies the ID of a vertex v; the reply to the probe is the entire row corresponding to v (namely, the list of all neighbors of v).
- A *weak probe* (WP) specifies a single cell (v,j) and receives its content (namely, only the j^{th} neighbor of v).

We note that knowing that u is v's i^{th} neighbor does not give us information regarding which one of u's ports v is connected to. This property is crucial for the proof of Theorem 4.

▶ **Definition 11** (Local computation algorithm)**.** A deterministic $p(n)$*-probe local computation algorithm* $\mathcal{A}$ for a computational problem is an algorithm that receives an input of size n. Given a query x, $\mathcal{A}$ makes at most $p(n)$ probes to the input in order to reply. $\mathcal{A}$ must be *consistent*; that is, the algorithm's replies to all of the possible queries combine to a single feasible solution to the problem.

A randomized $(p(n), s(n), \delta(n))$*-local computation algorithm* $\mathcal{A}$ differs from a deterministic one in the following aspects. Before receiving its input, it is allowed to write $s(n)$ random bits (referred to as the random *seed*) to memory.[4] Thereafter, it must behave like a deterministic LCA, except that when answering queries it may also read and use the random seed. For any input G, $|G| = n$, the probability (over the choice of random seed) that there exists a query in G for which $\mathcal{A}$ uses more than $p(n)$ probes is at most $\delta(n)$, which is called $\mathcal{A}$'s failure probability.

When LCA $\mathcal{A}$ is given input graph $G = (V, E)$ and queried on vertex $v \in V$, we denote this by $\mathcal{A}(G, v)$. An LCA $\mathcal{A}$ is said to *require* k probes on a graph $G = (V, E)$, if there is at least one query for which $\mathcal{A}$ uses k probes. We say that an LCA $\mathcal{A}$ requires k probes for a family F of graphs, if for some graph $G \in F$, $\mathcal{A}$ requires k probes.

▶ **Definition 12** (Approximation algorithm)**.** Given a maximization problem over graphs and a real number $0 \leq \alpha \leq 1$, a (possibly randomized) α-approximation algorithm $\mathcal{A}$ is guaranteed, for any input graph G, to output a feasible solution whose expected value is at least an α fraction of the value of an optimal solution (in expectation over the random bits used by $\mathcal{A}$).[5]

3 Lower Bound for Vertex Cover

In this section we prove Theorem 6:

▶ **Theorem 6.** *For any $\epsilon < \frac{1}{2}$, any randomized SP LCA that computes a vertex cover whose size is a $(\frac{1}{2}n^{1-2\epsilon})$-approximation to the size of the minimal vertex cover requires at least ϵn^{ϵ} probes.*

In order to prove Theorem 6 , we use the minimax theorem of Yao [38], by showing a lower bound on the expected number of probes required by a deterministic LCA when the input is selected from a certain distribution. To this end we construct, for infinitely many values of n, a family of graphs, parametrized by a constant $k \geq 3$.

[4] In this work, except where explicitly mentioned, we allow the random seed to be unbounded.
[5] The definition of approximation algorithms to minimization problems is analogous, with $\alpha \geq 1$.

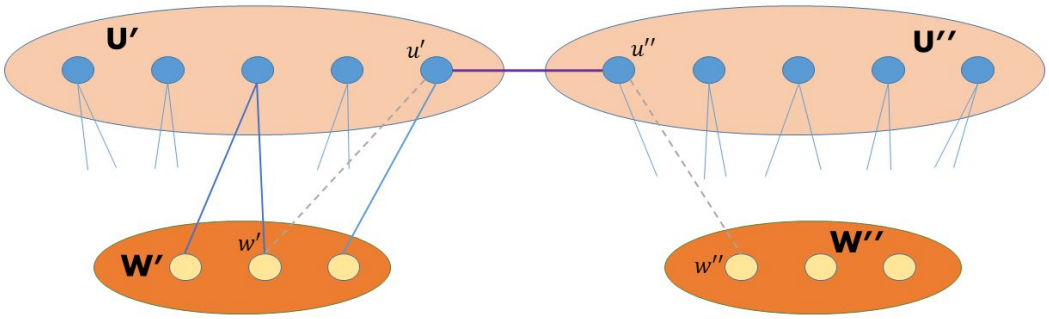


Figure 1 The graph fusion$(\mathbb{G}, (u', w'), (u'', w''))$. The dashed edges $e' = (u', w')$ and $e'' = (u'', v'')$ have been removed and the edge (u', u'') has been added.

Let p be a prime number, and $\mathbb{Z}(p)$ its associated field. Let $G^* = (U^* \cup W^*, E)$ be a bipartite graph, where $|U^*| = p^k$, $|W^*| = p^2$. Label each vertex in U^* by a k-tuple $(a_0, a_1, \ldots, a_{k-1})$, $a_i \in \mathbb{Z}(p), i \in \{0, 1, \ldots, k-1\}$ and the vertices in W^* by a pair (b_0, b_1), $b_i \in \mathbb{Z}(p), i \in \{0, 1\}$. Associate each vertex $u_j = (a_0, a_1, \ldots, a_{k-1}) \in U^*$ with the polynomial $f_j(x) = a_0 + a_1 x + \cdots + a_{k-1}x^{k-1}$. Connect $(b_0, b_1) \in W^*$ to u_j iff $b_1 = f_j(b_0)$.

▶ **Lemma 13.** *For every two different vertices $u_i, u_j \in U^*$, $N(u_i) \cap N(u_j) \leq k - 1$.*

Proof. Consider f_i and f_j, the polynomials associated with u_i and u_j respectively. Let $g(x) = f_i(x) - f_j(x)$. As g is not the zero polynomial, it has at most $k-1$ roots in $\mathbb{Z}(p)$. ◀

Let $\mathbb{G}$ to be a graph consisting of two identical copies of G^*. We denote these two copies by $G' = (U' \cup W', E')$ and $G'' = (U'' \cup W'', E'')$. Let $U = U' \cup U''$; let $W = W'' \cup W''$; let $n = |U \cup W| = 2(p^k + p^2)$.

We define the following operation on $\mathbb{G}$ (see Figure 1). Let e' and e'' be edges such that $e' = (u', w') : u' \in U', w' \in W'$ and $e'' = (u'', w'') : u'' \in U'', w'' \in W''$. Remove e' and e'' from $\mathbb{G}$, and add an edge $e = (u', u'')$. We call this operation fusion$(\mathbb{G}, e', e'')$, and call u and u' the *fusion vertices*. Note that there are p^{k+1} possible choices for e' and p^{k+1} possible choices for e''.

Given a graph $G =$ fusion$(\mathbb{G}, (u', w'), (u'', w''))$, the optimal size of the vertex cover of G is at most $2p^2 + 1$, as W and a fusion vertex constitute a vertex cover.

Note that a vertex can locally detect whether it is in U or in W just by looking at its own degree. However, to detect whether it is one of the fusion vertices, it needs to determine the degrees of its neighbors, which is impossible to do with number of probes significantly smaller than its degree.[6]

We now describe the graph family we use. Let $\mathbb{G} = (U \cup W, E)$ be as above. Let Π be the set of all possible namings of $U \cup W$ by the ID set $[n]$. Let $T = E' \times E''$. Given a naming $\pi \in \Pi$ and a pair of edges $\tau = (e', e'') \in T$, the graph $\mathbb{G}(\tau, \pi)$ is defined as follows:

1. The topology of $\mathbb{G}(\tau, \pi)$ is given by fusion$(\mathbb{G}, e', e'')$.
2. The vertices of $\mathbb{G}(\tau, \pi)$ are named according to π.

The family of graphs we consider is $\mathcal{G}(\Pi, T) = \{\mathbb{G}(\tau, \pi) \mid \tau \in T, \pi \in \Pi\}$. We now analyze the behavior of a given deterministic LCA A with proble complexity less than p/k on a

[6] Compare this with a distributed algorithm in the LOCAL model, for which one round suffices to determine this.

graph chosen uniformly at random from $\mathcal{G}(\Pi, T)$. We first make the following simplification. Suppose that A running on some $\mathbb{G}(\tau, \pi)$ is given v as a query. If A probes w'_τ or w''_τ, it knows that v is neither u'_τ nor u''_τ and hence A need not include v in the VC. For simplicity, we also assume that if A probes u'_τ or u''_τ, it knows not to add v to the VC. Note that this only strengthens A, hence we can make this assumption without loss of generality.

We use the following definition.

▶ **Definition 14** (View)**.** Let $\mathcal{A}$ be a deterministic SP LCA. We denote by $\textsc{View}(\mathcal{A}, G, v)$ the subgraph that $\mathcal{A}$ discovers when queried on vertex v in graph G, i.e., the set of all probed vertices and their neighbors.

▶ **Lemma 15.** *Let $\mathcal{A}$ be an SP LCA with probe complexity less than p/k. Let $G \in \mathcal{G}(\Pi, T)$. Assume that A is queried on G with vertex v. Then there is some vertex $w \in N(v)$ such that w has no neighbors except v in $\textsc{View}(\mathcal{A}, G, v)$. (That is, neither w nor any of its neighbors (except v) is probed in $\mathcal{A}(G, v)$.)*

Proof. $\mathcal{A}$ probes v and, say, a vertices from U and b vertices from W, for some $a, b \in \mathbb{N}$ such that $a + b < p/k$. From Lemma 13, A sees at most $a(k-1)$ vertices from $N(v)$ as a result of probing vertices in $U \setminus \{v\}$. Furthermore, A sees at most b vertices from $N(v)$ as a result of probing vertices in W. As $a(k-1) + b < p = |N(v)|$, at least one vertex in $N(v)$ is only seen once, while probing v. ◀

Consider an input graph $\mathbb{G}(\tau, \pi)$, where $\tau = ((u'_\tau, w'_\tau), (u''_\tau, w''_\tau))$. We use the following notation.

- A_v denotes the event that $\mathcal{A}$ is given v as a query. Note that A_v is independent of π and τ.
- $X_{\tau,\pi,i}$ denotes the event that none of $u'_\tau, w'_\tau, u''_\tau, w''_\tau$ is probed when A is queried on $i \in [n]$.

▶ **Lemma 16.** *Fix π and τ. Let v be a vertex in $U' \setminus \{u'_\tau\}$. If A_v and $X_{\tau,\pi,\pi(v)}$ hold, there exist $\pi_1 \in \Pi, \tau_1 \in T$ such that $\textsc{View}(\mathcal{A}, \mathbb{G}(\pi_1, \tau_1), u'_{\tau_1}) = \textsc{View}(\mathcal{A}, \mathbb{G}(\pi, \tau), v)$.*

Proof. By Lemma 15, there is some vertex $w \in N(v)$ that has no neighbors other than v in $\textsc{View}(\mathcal{A}, G(\pi, \tau), v)$. Let π_1 be identical to π except that v and u'_τ are interchanged. Set $\tau_1 = ((v, w), e''_\tau)$. The lemma follows. ◀

A symmetrical argument holds for $v \in U'' \setminus \{u''_\tau\}$.

▶ **Lemma 17.** *Fix $\mathbb{G}$, and let $i \in [n]$ be the ID of the vertex given to a deterministic SP LCA $\mathcal{A}$ as a query. If the probe complexity of $\mathcal{A}$ is less than p/k, then $\Pr[X_{\tau,\pi,i}] \geq 1 - \frac{1}{kp}$, where the probability is over the choice of π and τ.*

Proof. Fix π. If A probes a vertices in U and b vertices in W, it will hit one of $u'_\tau, w'_\tau, u''_\tau, w''_\tau$ with probability at most $\frac{a}{p^k} + \frac{b}{p^2}$, over the choice of τ. Since $a, b \geq 0$ and $a + b \leq p/k$, the probability is maximized for $a = 0, b = p/k$. As this bound holds for any π, the result follows. ◀

Lemma 16 and Lemma 17 imply that when a deterministic SP LCA is queried on a vertex $u \in U$ from a random graph in $\mathcal{G}(T, \Pi)$, it cannot discern in less than p/k probes whether u is a fusion vertex with probability greater than $\frac{1}{kp}$. Hence the LCA must add vertex u to the VC, because at least one fusion vertex *must* be in the VC. Therefore, the size of the VC that A computes is at least $p^k - O(1)$, whereas the optimal VC has size at most $p^2 + 1$:

▶ **Theorem 18.** *There does not exist a deterministic SP LCA A that computes a VC that is an $(\frac{1}{2}n^{1-2\epsilon})$-approximation to the optimal VC and uses fewer than ϵn^{ϵ} probes with probability greater than $\frac{1}{kp}$ on a graph chosen uniformly at random from $\mathcal{G}(\Pi, T)$.*

Proof. Let $\epsilon = 1/k$. Recall that $n = \Theta(p^k)$. We have shown that if the number of probes is less than $p/k = \Theta(n^{\epsilon} \cdot \epsilon)$, then the approximation ratio is at least $p^k/p^2 - o(1) = n^{1-2/k} - o(1)$. ◀

Applying Yao's principle [38] to Theorem 18 completes the proof of Theorem 6.

▶ **Corollary 19.** *Any SP LCA for maximal matching on arbitrary graphs requires $\Omega(n^{1/2-o(1)})$ probes.*

Proof. It is well known that, given any maximal matching, taking both end vertices of every edge gives a 2-approximation to the VC (e.g., [37]). Therefore, an LCA for maximal matching would immediately give a 2-approximation to the minimal vertex cover. Setting $2 = \Theta(n^{1-2\epsilon})$ in Theorem 6 gives $\epsilon = 1/2$. The result follows. ◀

References

1. Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *Proc. 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1132–1139, 2012.
2. Reid Andersen. A local algorithm for finding dense subgraphs. *ACM Trans. Algorithms*, 6(4), 2010.
3. Reid Andersen, Shayan Oveis Gharan, Yuval Peres, and Luca Trevisan. Almost optimal local graph clustering using evolving sets. *J. ACM*, 63(2):15, 2016.
4. Vincent D Blondel, Julien M Hendrickx, Alex Olshevsky, and John N Tsitsiklis. Convergence in multiagent coordination, consensus, and flocking. In *Proceedings of IEEE Conference on Decision and Control*, pages 2996–3000. IEEE, 2005.
5. Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM J. Comput.*, 34(6):1370–1379, 2005.
6. Guy Even, Moti Medina, and Dana Ron. Best of two local models: Local centralized and local distributed algorithms. *CoRR*, abs/1402.3796, 2014. URL: `http://arxiv.org/abs/1402.3796`, `arXiv:1402.3796`.
7. Guy Even, Moti Medina, and Dana Ron. Distributed maximum matching in bounded degree graphs. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking, ICDCN*, pages 18:1–18:10, 2015.
8. Uriel Feige, Yishay Mansour, and Robert E. Schapire. Learning and inference in the presence of corrupted inputs. In *Proceedings of The 28th Conference on Learning Theory, COLT*, pages 637–657, 2015.
9. Uriel Feige, Boaz Patt-Shamir, and Shai Vardi. On the probe complexity of local computation algorithms. *CoRR*, abs/1703.07734, 2017. `arXiv:1703.07734`.
10. Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS*, pages 625–634, 2016.
11. David Gamarnik and David A. Goldberg. Randomized greedy algorithms for independent sets and matchings in regular graphs: Exact results and finite girth corrections. *Combinatorics, Probability and Computing*, 19:61–85, 1 2010. `doi:10.1017/S0963548309990186`.
12. Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 270–277, 2016.
13. Andrew V. Goldberg, Serge A. Plotkin, and Gregory E. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM J. Discret. Math.*, 1(4):434–446, 1988.

14 Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

15 Mika Göös, Juho Hirvonen, Reut Levi, Moti Medina, and Jukka Suomela. Non-local probes do not help with many graph problems. *Distributed Computing - 30th International Symposium, DISC*, pages 201–214, 2016.

16 Avinatan Hassidim, Yishay Mansour, and Shai Vardi. Local computation mechanism design. *ACM Trans. Economics and Comput.*, 4(4):21:1–21:24, 2016. doi:10.1145/2956584.

17 M. Jha and S. Raskhodnikova. Testing and reconstruction of Lipschitz functions with applications to data privacy. In *Proc. 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2011.

18 J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proc. 32nd Annual ACM Symposium on the Theory of Computing (STOC)*, pages 80–86, 2000.

19 Reut Levi, Dana Ron, and Ronitt Rubinfeld. Local algorithms for sparse spanning graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 826–842, 2014.

20 Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Brief announcement: Local computation algorithms for graphs of non-constant degrees. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA*, pages 59–61, 2015.

21 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1), 1992.

22 Palma London, Niangjun Chen, Shai Vardi, and Adam Wierman. Distributed optimization via local computation algorithms. http://users.cms.caltech.edu/~plondon/loco.pdf, 2017.

23 Steven H Low, Fernando Paganini, and John C Doyle. Internet congestion control. *IEEE control systems*, 22(1):28–43, 2002.

24 Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986.

25 Nancy A. Lynch. Upper bounds for static resource allocation in a distributed system. *J. Comput. Syst. Sci.*, 23(2):254–278, 1981.

26 Yishay Mansour, Boaz Patt-Shamir, and Shai Vardi. Constant-time local computation algorithms. *Theory of Computing Systems*, pages 1–19, 2017.

27 Yishay Mansour, Aviad Rubinstein, Shai Vardi, and Ning Xie. Converting online algorithms to local computation algorithms. In *Proc. 39th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 653–664, 2012.

28 Moni Naor. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM J. Discrete Math.*, 4(3):409–412, 1991.

29 Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995.

30 Huy N. Nguyen and Krzystof Onak. Constant-time approximation algorithms via local improvements. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 327–336, 2008.

31 M. Parnas and D. Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1–3), 2007.

32 Omer Reingold and Shai Vardi. New techniques and tighter bounds for local computation algorithms. *J. Comput. Syst. Sci.*, 82(7):1180–1200, 2016.

33 Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Proc. 2nd Symposium on Innovations in Computer Science (ICS)*, pages 223–238, 2011.

34 Michael E. Saks and C. Seshadhri. Local monotonicity reconstruction. *SIAM J. Comput.*, 39(7):2897–2926, 2010.

35 Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM J. Comput.*, 42(1):1–26, 2013.

36 Shai Vardi. *Designing Local Computation Algorithms and Mechanisms*. PhD thesis, Tel Aviv University, Tel Aviv, Israel, 2015.

37 Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001.

38 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, FOCS '77, pages 222–227, 1977.

39 Sergey Yekhanin. Locally decodable codes. *Foundations and Trends in Theoretical Computer Science*, 6(3):139–255, 2012.

40 Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. Improved constant-time approximation algorithms for maximum matchings and other optimization problems. *SIAM J. Comput.*, 41(4):1074–1093, 2012.

Fully-Dynamic Bin Packing with Little Repacking

Björn Feldkord
Paderborn University, Paderborn, Germany

Matthias Feldotto
Paderborn University, Paderborn, Germany

Anupam Gupta[1]
Carnegie Mellon University, Pittsburgh, USA

Guru Guruganesh[2]
Carnegie Mellon University, Pittsburgh, USA

Amit Kumar[3]
IIT Delhi, New Delhi, India

Sören Riechers
Paderborn University, Paderborn, Germany

David Wajc[4]
Carnegie Mellon University, Pittsburgh, USA

"To improve is to change; to be perfect is to change often."
– Winston Churchill.

Abstract

We study the classic BIN PACKING problem in a fully-dynamic setting, where new items can arrive and old items may depart. We want algorithms with low asymptotic competitive ratio *while repacking items sparingly* between updates. Formally, each item i has a *movement cost* $c_i \geq 0$, and we want to use $\alpha \cdot OPT$ bins and incur a movement cost $\gamma \cdot c_i$, either in the worst case, or in an amortized sense, for α, γ as small as possible. We call γ the *recourse* of the algorithm. This is motivated by cloud storage applications, where fully-dynamic BIN PACKING models the problem of data backup to minimize the number of disks used, as well as communication incurred in moving file backups between disks. Since the set of files changes over time, we could recompute a solution periodically from scratch, but this would give a high number of disk rewrites, incurring a high energy cost and possible wear and tear of the disks. In this work, we present optimal tradeoffs between number of bins used and number of items repacked, as well as natural extensions of the latter measure.

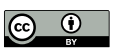
2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Bin Packing, Fully Dynamic, Recourse, Tradeoffs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.51

Related Version This paper is a merged version of papers [15], https://arxiv.org/abs/1711.02078, and [11], https://arxiv.org/abs/1711.01231.

[1] Work done in part while the authors were at the Simons Institute for the Theory of Computing.
[2] Work done in part while the authors were at the Simons Institute for the Theory of Computing.
[3] Work done in part while the authors were at the Simons Institute for the Theory of Computing.
[4] Work done in part while the authors were at the Simons Institute for the Theory of Computing.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 51; pp. 51:1–51:24

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements The work of Björn Feldkord, Matthias Feldotto and Sören Riechers is supported in part by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901). The work of Anupam Gupta and Guru Guruganesh is supported in part by NSF awards CCF-1536002, CCF-1540541, and CCF-1617790. Amit Kumar and Anupam Gupta are part of the Joint Indo-US Virtual Center for Computation and Uncertainty. The work of David Wajc is supported in part by NSF grants CCF-1527110, CCF-1618280 and NSF CAREER award CCF-1750808.

1 Introduction

Consider the problem of data backup on the cloud, where multiple users' files are stored on disks (for simplicity, of equal size). This is modeled by the BIN PACKING problem, where the items are files and the bins are disks, and we want to pack the items in a minimum number of bins. However, for the backup application, files are created and deleted over time. The storage provider wants to use a small number of disks, and also keep the communication costs incurred by file transfers to a minimum. Specifically, we want bounded "recourse", i.e., items should be moved sparingly while attempting to minimize the number of bins used in the packing. These objectives are at odds with each other, and the natural question is to give optimal tradeoffs between them.

Formally, an instance $\mathcal{I}$ of the BIN PACKING problem consists of a list of n items of sizes $s_1, s_2, \ldots, s_n \in [0, 1]$. A bin packing algorithm seeks to pack the items of $\mathcal{I}$ into a small number of unit-sized bins; let $OPT(\mathcal{I})$ be the minimum number of unit-sized bins needed to pack all of $\mathcal{I}$'s items. This NP-hard problem has been studied since the 1950s, with hundreds of papers addressing its many variations; see e.g., [18, Chapter 2] and [7] for surveys. Much of this work (e.g. [27, 39, 29, 30, 31, 37, 36, 33, 16, 2]) starting with [35] studies the *online* setting, where items arrive sequentially and are packed into bins immediately and irrevocably. While the offline problem is approximable to within an *additive* term of $\mathcal{O}(\log OPT)$ [17], in the online setting there is a 1.540-*multiplicative gap* between the algorithm and OPT in the worst case, even as $OPT \to \infty$ [2]. Given the wide applicability of the online problem, researchers have studied the problem where a small amount of repacking is allowed upon item additions and deletions. Our work focuses on the bounded recourse setting, and we give *optimal tradeoffs* between the number of bins used and amount of repacking required in the fully-dynamic setting for several settings.

A *fully-dynamic* BIN PACKING *algorithm* $\mathcal{A}$, given a sequence of item insertions and deletions, maintains at every time t a feasible solution to the BIN PACKING instance $\mathcal{I}_t$ given by items inserted and not yet deleted until time t. Every item i has a size s_i and a *movement cost* c_i, which $\mathcal{A}$ pays every time $\mathcal{A}$ moves item i between bins. Different approaches have been developed to measure the amount of repacking, in the context of data backup these measures corresponds to the communication or energy cost. For example, the movement cost c_i may be proportional to the size of the file (which is the communication cost if the files are all large), or may be a constant (if the files are small, and the overhead is entirely in setting up the connections), or may depend on the file in more complicated ways.

Formally, the fully-dynamic BIN PACKING problem is as follows.

▶ **Definition 1.1.** *A fully-dynamic algorithm $\mathcal{A}$ has (i) an* asymptotic competitive ratio α, *(ii)* additive term β *and (iii)* recourse γ, *if at each time t it packs the instance $\mathcal{I}_t$ in at most $\alpha \cdot OPT(\mathcal{I}_t) + \beta$ bins, while paying at most $\gamma \cdot \sum_{i=1}^{t} c_i$ movement cost until time t. If at each time t algorithm $\mathcal{A}$ incurs at most $\gamma \cdot c_t$ movement cost, for c_t the cost of the item updated at time t, we say $\mathcal{A}$ has* worst-case *recourse γ, otherwise we say it has* amortized *recourse γ.*

Any algorithm must pay $\frac{1}{2}\sum_{i=1}^{t} c_i$ movement cost just to insert the items, so an algorithm with γ recourse spends at most 2γ times more movement cost than the absolute minimum. The goal is to design algorithms which simultaneously have low asymptotic competitive ratio (a.c.r), additive term, and recourse. There is a natural tension between the a.c.r and recourse, so we want to find the optimal a.c.r-to-recourse trade-offs.

1.1 Related Work

Based on the classical BIN PACKING Problem [14], different online and dynamic variants have been developed and investigated. Due to space constraints we focus on the scenarios which share the most important properties with our model and only mention the best results for them. In the *Online Bin Packing Problem* [35], the items are unknown to the algorithm at the beginning and appear one after another. Balogh et al. [2] show a lower bound of 1.54037 for this problem and Seiden [33] presents an approximation algorithm with a competitive ratio of 1.58889. In the *Dynamic Bin Packing* setting [28], additionally to arrivals as in the Online Bin Packing Problem departures of items are also allowed. Here, a lower bound of 8/3 by Wong et al. [38] and an upper bound of 2.897 by Coffman et al. [28] exist.

We now turn our attention to the models which allow repacking of items. In the *Relaxed Online Bin Packing Problem*, first studied by Gambosi et al. [12, 13], online arrivals and no departures occur, but repacking of items is allowed. Repacking means that items can be assigned to another bin in the course of the execution, while in a setting without repacking decisions are irrevocable. Gambosi et al. gave a 4/3-a.c.r algorithm for the insertion-only setting which moves each item $\mathcal{O}(1)$ times, which in terms of recourse translates into $\mathcal{O}(1)$ amortized recourse for general movement costs. Following Sanders et al. [32], who studied makespan minimization with recourse, Epstein et al. [9] re-introduced the dynamic BIN PACKING problem with the movement costs c_i equaling the size s_i (the *size cost* setting, where worst-case recourse is also called *migration factor*). Epstein et al. gave a solution with a.c.r $(1+\varepsilon)$ and bounded (exponential in ϵ^{-1}) recourse for the insertion-only setting. Jansen and Klein [23] improved the recourse to $poly(\varepsilon^{-1})$ for insertion-only algorithms. The best known lower bound for an algorithm which uses only constant recourse in the *unit cost* model (i.e. $c_i = 1$ for all items) is originally given for our model, but it also applies to this setting with 1.3871 by Balogh et al. [4]. From the positive perspective Balogh et al. [5] give an approximation algorithm based on the Harmonic Fit Algorithm [29] for which they achieve a competitive ratio of $3/2$ and $\mathcal{O}(\varepsilon^{-1})$ movements per update. Since our algorithms are also applicable to this setting we improve this result to also close the gap between the lower and upper bound for this problem.

Our setting is the most powerful model among the presented ones, the *Fully-Dynamic Bin Packing* [21], in which we allow arrivals and departures of items as well as repacking. Ivković and Lloyd [21, 22] introduced the model of Fully-Dynamic Bin Packing and developed an algorithm called *Mostly Myopic Packing (MMP)* which achieves a $\frac{5}{4}$-competitive ratio. Their algorithm is based on an offline algorithm by Johnson [25, 26] and utilizes a technique whereby the packing of an item is done with a total disregard for already packed items of a smaller size. In contrast to our work, they use the concept of bundling smaller elements in their analysis, i.e. a group of items smaller than ε may be moved as a single item for unit costs. They can show that the number of single items or bundles of very small elements that need to be repacked is bounded by a constant. Additionally, Ivković [19] also gives a slightly simpler version of this algorithm, called *Myopic Packing (MP)*. It uses similar ideas but ignores one step of MMP that results in a much easier analysis and a competitive ratio of $\frac{4}{3}$. Berndt et al. [6] consider exactly the same setting, also allowing the bundling of very

small elements in the analysis. Their algorithm achieves a $1+\varepsilon$ approximation ratio and a bound of $\mathcal{O}\left(1/\varepsilon^4 \log\left(1/\varepsilon\right)\right)$ for the recourse in both the size costs and unit costs model.

In addition to the positive algorithmic results, researchers also explored lower bounds for this setting. All results assume that no bundling is allowed in the unit cost model (otherwise there can only be the trivial lower bound of one [6]), hence allowing only a constant number of shifted items per time step. Ivković and Lloyd [20] show a lower bound of $\frac{4}{3}$. Their construction uses the inability of an algorithm to react to insertions and deletions of items with size slightly larger than $\frac{1}{2}$ when the remaining items may be of arbitrarily small size. Balogh et al. [3, 4] improve this bound to roughly 1.3871. They extend the technique of the previous lower bound by constructing multiple lists of large items whose sizes are chosen through the construction of a linear program. Their results are the inspiration for some of the parameter choices in this work. For size costs, Berndt et al. [6] showed that any $(1+\varepsilon)$ a.c.r algorithm must have *worst-case recourse* $\Omega(\varepsilon^{-1})$. While these give nearly-tight results for "size costs" $c_i = s_i$, the unit cost ($c_i = 1$) and general cost cases were not so well understood prior to this work.

1.2 Our Results

We give (almost) tight characterizations for the recourse-to-asymptotic competitive ratio trade-off for fully-dynamic BIN PACKING under (a) unit movement costs, (b) general movement costs and (c) size movement costs. Our results are summarized in the following theorems. (See Tables 1 and 2 for a tabular listing of our results contrasted with the best previous results.) Note that an amortized recourse bound is a weaker (resp. stronger) upper (resp. lower) bound. In the context of sensitivity analysis (see [32]), our bounds provide a tight characterization of the *average* change between *approximately*-optimal solutions of slightly modified instances.

Unit Costs. Consider the most natural movement costs, *unit costs*, where $c_i = 1$ for all items i. Here we give tight upper and lower bounds. Let $\alpha = 1 - \frac{1}{W_{-1}(-2/e^3)+1} \approx 1.3871$ (here W_{-1} is the lower real branch of the Lambert W-function [8]). Balogh et al. [4] showed that α is a lower bound on the a.c.r with constant recourse. We present an alternative and simpler proof of this lower bound, also giving tighter bounds: doing better than α requires either *polynomial* additive term or recourse. Moreover, we give two matching algorithms proving α is tight for this problem: The first one uses directly the insights from the lower bound while the second one drives deeper into the structural insights of the current problem instance and reaches a slightly better result.

▶ **Theorem 1.2** (Unit Costs Tight Bounds). *For any $\varepsilon > 0$, there exists fully-dynamic* BIN PACKING *algorithms with amortized competitive ratio $(\alpha+\varepsilon)$ with worst case recourse $\mathcal{O}(\varepsilon^{-2})$ under unit movement costs and additive term $\mathcal{O}(\varepsilon^{-1})$. Conversely, any algorithm with a.c.r $(\alpha-\varepsilon)$ has additive term and amortized recourse whose product is $\Omega(\varepsilon^4 \cdot n)$ under unit movement costs.*

General Costs. Next, we consider the most general problem, with arbitrary movement costs. Theorem 1.2 showed that in the unit cost model, we can get a better a.c.r than for online BIN PACKING without repacking, whose optimal a.c.r is at least 1.540 ([2]). Alas, the fully-dynamic BIN PACKING problem with the general costs is not easier than the arrival-only online problem (with no repacking).

Table 1 Fully-dynamic bin packing with limited recourse: Positive results (Big-$\mathcal{O}$ notation dropped for notational simplicity)

Costs	A.C.R	Additive	Recourse	W.C.	Notes	Reference
Unit	$1.5+\varepsilon$	ϵ^{-1}	ϵ^{-1}	✓	insertions only	Balogh et al. [5]
	$\alpha+\varepsilon$	ϵ^{-1}	ε^{-2}	✓	$\alpha \approx 1.387$	**Theorem 2.11**
General	1.589	1	1	✗		**Theorem 3.5**
	1.333	1	1	✗	insertions only	Gambosi et al. [13]
Size	$1+\varepsilon$	1	$\varepsilon^{-O(\epsilon^{-2})}$	✓	insertions only	Epstein & Levin [10]
	$1+\varepsilon$	ϵ^{-2}	ϵ^{-4}	✓	insertions only	Jansen & Klein [23]
	$1+\varepsilon$	$poly(\epsilon^{-1})$	$\epsilon^{-3}\log(\varepsilon^{-1})$	✓	insertions only	Berndt et al. [6]
	$1+\varepsilon$	$poly(\epsilon^{-1})$	$\epsilon^{-4}\log(\varepsilon^{-1})$	✓		Berndt et al. [6]
	$1+\varepsilon$	ϵ^{-1}	ϵ^{-2}	✗		**Fact 4.1**

Table 2 Fully-dynamic bin packing with limited recourse: Negative results.

Costs	A.C.R	Additive	Recourse	W.C.	Notes	Reference
Unit	1.333	$o(n)$	1	✓		Ivković & Lloyd [20]
	$\alpha-\varepsilon$	$o(n)$	1	✓	$\alpha \approx 1.387$	Balogh et al. [4]
	$\alpha-\varepsilon$	$o(\epsilon^2 \cdot n^{\delta})$	$\Omega(\varepsilon^2 \cdot n^{1-\delta})$	✗	for all $\delta \in (0, 1/2]$	**Theorem 2.1**
General	1.540	$o(n)$	∞	✗	as hard as online	**Theorem 3.1**
Size	$1+\varepsilon$	$o(n)$	$\Omega(\epsilon^{-1})$	✓		Berndt et al. [6]
	$1+\varepsilon$	$o(n)$	$\Omega(\epsilon^{-1})$	✗		**Theorem 4.2**

▶ **Theorem 1.3** (Fully Dynamic as Hard as Online). *Any fully-dynamic* BIN PACKING *algorithm with bounded recourse under general movement costs has a.c.r at least as high as that of any online* BIN PACKING *algorithm. Given current bounds ([2]), this is at least* 1.540.

Given this result, is it conceivable that the fully-dynamic model is *harder* than the arrival-only online model, even allowing for recourse? We show this is likely not the case, as we can almost match the a.c.r of the current-best algorithm for online BIN PACKING.

▶ **Theorem 1.4** (Fully Dynamic Nearly as Easy as Online). *Any algorithm in the Super Harmonic family of algorithms can be implemented in the fully-dynamic setting with constant recourse under general movement costs. This implies an algorithm with a.c.r* 1.589 *using [33].*

The current best online BIN PACKING algorithm [1] is not from the Super Harmonic family but is closely related to it. It has an a.c.r of 1.578, so our results for fully-dynamic BIN PACKING are within a hair's width of the best bounds known for online BIN PACKING. It remains an open question as to whether our techniques can be extended to achieve the improved a.c.r bounds while maintaining constant recourse. While we are not able to give a black-box reduction from fully-dynamic algorithms to online algorithms, we conjecture that such a black-box reduction exists and that these problems' a.c.r are equal.

Size Costs. Finally, we give an extension of the already strong results known for the size cost model (where $c_i = s_i$ for every item i). We show that the lower bound known in the worst-case recourse model extends to the amortized model as well, for which it is easily shown to be tight.

▶ **Theorem 1.5** (Size Costs Tight Bounds). *For any $\varepsilon > 0$, there exists a $(1+\varepsilon)$-a.c.r algorithm with $\mathcal{O}(\varepsilon^{-2})$ additive term and $\mathcal{O}(\varepsilon^{-1})$ amortized recourse under size costs. Conversely, for infinitely many $\varepsilon > 0$, any $(1+\varepsilon)$-a.c.r algorithm with $o(n)$ additive term requires $\Omega(\varepsilon^{-1})$ amortized recourse under size costs.*

The hardness result above was previously only known for *worst-case* recourse [6]; this previous lower bound consists of a hard instance which effectively disallowed any recourse, while lower bounds against amortized recourse can of course not do so.

1.3 Techniques and Approaches

Unit Costs. For unit costs, our lower bound is based on a natural instance consisting of small items, to which large items of various sizes (greater than $\frac{1}{2}$) are repeatedly added/removed [4]. The near-optimal solutions oscillate between either having most bins containing both large and small items, or most bins containing only small items. Since small items can be moved rarely, this effectively makes them static, giving us hard instances. We can optimize for the lower bound arising thus via a *gap-revealing* LP, similar to [4]. However, rather than bound this LP precisely, we exhibit a near-optimal dual solution showing a lower bound of $\alpha - \varepsilon$.

For the first upper bound, the same LP now changes from a gap-revealing LP to a *factor-revealing* LP. The LP solution shows how the small items should be packed in order to "prepare" for arrival of large items, to ensure $(\alpha + \epsilon)$-competitiveness. An important building block of our algorithms is the ability to deal with (sub)instances made up solely of small items. In particular, we give fully-dynamic BIN PACKING algorithms with a.c.r $(1+\varepsilon)$ and only $\mathcal{O}(\varepsilon^{-2})$ *worst-case* recourse if all items have size $\mathcal{O}(\varepsilon)$. The ideas we develop are versatile, and are used, e.g., to handle the small items for general costs – we discuss them in more detail below. We then extend these ideas to allow for (approximately) packing items according to a "target curve", which in our case is the solution to the above LP. At this point the LP makes a re-appearance, allowing us to analyze a simple packing of large items "on top" of the curve (which we can either maintain dynamically using the algorithm of Berndt et al. [6] or recompute periodically in linear time); in particular, using this LP we are able to prove this packing of large items on top of small items has an optimal a.c.r of $\alpha + \mathcal{O}(\varepsilon)$. This implies that lazily repacking the large items near-optimally and packing them on top of the small items easily yields $\mathcal{O}(\varepsilon^{-2})$ amortized recourse. Relying on the fully-dynamic $poly(\varepsilon^{-1})$ migration factor AFPTAS algorithms of [6] with some additional ideas allows us to leverage this LP to obtain worst-case recourse bounds.

In the second upper bound, we take a different, more recourse-efficient method to pack the large items, which results in improved worst-case recourse. Specifically, we utilize an algorithm called Myopic Packing (MP) by Ivković [19]. This algorithm has a competitive ratio of $4/3$ (it is below the lower bound for our model since it uses bundling, i.e., it allows the repacking of a group of small items as one) and modifies only a constant number of bins per time step. Applying this algorithm to only items of a size of at least $\varepsilon/15$ restricts the amount of repacking to $\mathcal{O}(1/\varepsilon)$ per time step. We develop a new view of this algorithm to derive structural properties of the solution which are needed for the combination of major and minor items. Small items are handled similarly to the previous algorithm, however, we provide a fixed solution to the LP from before in order to use it in the analysis of the combined solution. Finally, the bins with small items are merged with the bins with large items by utilizing a greedy-like approach, where small chunks of reserved space and a big cumulative size of major items is prioritized in order to guarantee an efficient utilization of the reserved space. The combination has two main challenges: Firstly, we ensure that

this greedy process only has to modify $\mathcal{O}(1/\varepsilon)$ bins per time step. Secondly, we guarantee a space-efficient combination resulting in an overall good solution quality. The analysis carefully utilizes the structural insights about the solution for major items to estimate the solution quality within the bins that did not get merged.

General Costs. To transform any online lower bound to a lower bound for general costs even with recourse, we give the same sequence of items but with movement costs that drop sharply. This prohibits worst-case recourse, but since we allow amortized recourse, the large costs c_i for early items can still be used to pay for movement of the later items (with smaller c_i values). Hence, if the online algorithm moves an element e to some bin j, we delete all the elements after e and proceed with the lower bound based on assigning $e \mapsto j$. Now a careful analysis shows that for some sub-sequence of items the dynamic algorithm effectively performs no changes, making it a recourse-less online algorithm – yielding the claimed bound.

For the upper bound, we show how to emulate any algorithm in the *Super Harmonic* class of algorithms [33] even in the fully-dynamic case. We observe that the analysis for SH essentially relies on maintaining a stable matching in an appropriate compatibility graph. Now our algorithm for large items uses a subroutine similar to the Gale-Shapley algorithm. Our simulation also requires a solution for packing similarly-sized items. The idea here is to sort items by their cost (after rounding costs to powers of two), such that any insertion or deletion of items of this size can be "fixed" by moving at most one item of each smaller cost, yielding $\mathcal{O}(1)$ worst-case recourse.

The final ingredient of our algorithm is packing of small items into $(1-\epsilon)$-full bins. Unlike the online algorithm which can just use FirstFit, we have to extend the ideas of Berndt et al. [6] for the size-cost case. Namely we group bins into buckets of $\Theta(1/\epsilon)$ bins, such that all but one bin in a bucket are $1-\mathcal{O}(\epsilon)$ full, guaranteeing these bins are $1-\mathcal{O}(\epsilon)$ full on average. While for the size-cost case, bucketing bins and sorting the small items by size readily yields an $\mathcal{O}(\epsilon^{-1})$ recourse bound, this is more intricate for general case where size and cost are not commensurate. Indeed, we also maintain the small items in sorted order according to their size-to-movement-cost ratio (i.e., their *Smith ratio*), and only move items to/from a bin once it has $\Omega(\epsilon)$'s worth of volume removed/inserted (keeping track of erased, or "ghost" items). This gives an amortized $\mathcal{O}(\epsilon^{-2})$ recourse cost for the small items.

Size Costs. The technical challenge for size costs is in proving the lower bound with *amortized* recourse that matches the easy upper bound. Our proof uses item sizes which are roughly the reciprocals of the Sylvester sequence [34]. While this sequence has been used often in the context of bin packing algorithms, our sharper lower bound explicitly relies on its divisibility properties (which have not been used in similar contexts, to the best of our knowledge). We show that for these sizes, any algorithm $\mathcal{A}$ with a.c.r $(1+\epsilon)$ must, on an instance containing N items of each size, have $\Omega(N)$ many bins with one item of each size. In contrast, on an instance containing N items of each size *except the smallest size* ε, algorithm $\mathcal{A}$ must have some (smaller) $\mathcal{O}(N)$ many bins with more than one distinct size in them. Repeatedly adding and removing the N items of size ε, the algorithm must suffer a high amortized recourse.

We give a more in-depth description of our algorithms and lower bounds for unit, general and size costs, highlighting some salient points of their proofs in §2,§3 and §4, respectively. Due to space constraints we defer most proofs to the full versions of this paper ([15, 11]).

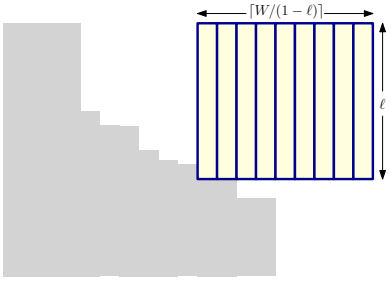


Figure 1 A packing of instance $\mathcal{I}_\ell$. The ℓ-sized items (in yellow) are packed "on top" of the small items (in grey).

2 Unit Movement Costs

We consider the natural unit movement costs model. First, we show that an a.c.r better than $\alpha = 1 - \frac{1}{W_{-1}(-2/e^3)+1} \approx 1.387$ implies either polynomial additive term or recourse. Next, we give tight $(\alpha+\epsilon)$-a.c.r algorithms, with both additive term and recourse polynomial in ϵ^{-1}. Our key idea is to use an LP that acts both as a *gap-revealing LP* for a lower bound, and also as a *factor-revealing LP* (as well as an inspiration) for our algorithm.

2.1 Impossibility results

Alternating between two instances, where both have $2n-4$ items of size $1/n$ and one instance also has 4 items of size $1/2 + 1/2n$, we get that any $\left(\frac{3}{2} - \varepsilon\right)$-competitive online BIN PACKING algorithm must have $\Omega(n)$ recourse. However, this only rules out algorithms with a *zero* additive term. Balogh et al. [4] showed that any $\mathcal{O}(1)$-recourse algorithm (under unit movement costs) with $o(n)$ additive term must have a.c.r at least $\alpha \approx 1.387$. We strengthen both this impossibility result by showing the need for a large additive term or recourse to achieve any a.c.r below α. Specifically, arbitrarily small polynomial additive terms imply near-linear recourse. Our proof is shorter and simpler than in [4]. As an added benefit, the LP we use to bound the competitive ratio will inspire our algorithm in the next section.

▶ **Theorem 2.1** (Unit Costs: Lower Bound). *For any $\varepsilon > 0$ and $\frac{1}{2} > \delta > 0$, for any algorithm $\mathcal{A}$ with a.c.r $(\alpha - \varepsilon)$ with additive term $o(\varepsilon \cdot n^\delta)$, there exists an dynamic bin packing input with n items on which $\mathcal{A}$ uses recourse at least $\Omega(\varepsilon^2 \cdot n^{1-\delta})$ under unit movement cost.*

The Instances: The set of instances is the a natural one, and was also considered by [4]. Let $1/2 > \delta > 0$, $c = 1/\delta - 1$, and let $W \geq 1/\varepsilon$ be a large integer. Our hard instances consist of *small* items of size $1/W^c$, and *large* items of size ℓ for all sizes $\ell \in \mathcal{S}_\varepsilon \triangleq \{\ell = 1/2 + i \cdot \varepsilon \mid i \in \mathbb{N}_{>0}, \ell \leq 1/\alpha\}$. Specifically, input $\mathcal{I}_s$ consists of $\lfloor W^{c+1} \rfloor$ small items, and for each $\ell \in \mathcal{S}_\varepsilon$, the input $\mathcal{I}_\ell$ consists of $\lfloor W^{c+1} \rfloor$ small items followed by $\lfloor \frac{W}{1-\ell} \rfloor$ size-ℓ items. The optimal bin packings for $\mathcal{I}_s$ and $\mathcal{I}_\ell$ require precisely $OPT(\mathcal{I}_s) = W$ and $OPT(\mathcal{I}_\ell) = \lceil \frac{W}{1-\ell} \rceil$ bins respectively. Consider any fully-dynamic bin packing algorithm $\mathcal{A}$ with limited recourse and bounded additive term. When faced with input $\mathcal{I}_s$, algorithm $\mathcal{A}$ needs to distribute the small items in the available bins almost uniformly. And if this input is extended to $\mathcal{I}_\ell$ for some $\ell \in \mathcal{S}_\varepsilon$, algorithm $\mathcal{A}$ needs to move many small items to accommodate these large items (or else use many new bins). Since $\mathcal{A}$ does not know the value of t beforehand, it cannot "prepare" simultaneously for all large sizes $\ell \in \mathcal{S}_\varepsilon$.

As a warm-up we show that the linear program (LP_ε) below gives a lower bound on the *absolute* competitive ratio α_ε of any algorithm $\mathcal{A}$ with no recourse. Indeed, instantiate the variables as follows. On input $\mathcal{I}_s$, let N_x be the number of bins in which $\mathcal{A}$ keeps free space in the range $[x, x+\varepsilon)$ for each $x \in \{0\} \cup \mathcal{S}_\varepsilon$. Hence the total volume packed is at most $N_0 + \sum_{x\in\mathcal{S}_\varepsilon}(1-x)\cdot N_x$. This must be at least $Vol(\mathcal{I}_s) \geq W - 1/W^c$, implying constraint ($\mathrm{Vol}_\varepsilon$). Moreover, as $OPT(\mathcal{I}_s) = W$, the α_ε-competitiveness of $\mathcal{A}$ implies constraint ($\mathrm{small}_\varepsilon$). Now if instance $\mathcal{I}_s$ is extended to $\mathcal{I}_\ell$, since $\mathcal{A}$ moves no items, these ℓ-sized items are placed either in bins counted by N_x for $x \geq \ell$ or in new bins. Since $\mathcal{A}$ is α_ε-competitive and $OPT(\mathcal{I}_\ell) \leq \lceil \frac{W}{1-\ell} \rceil$ we get constraint (CR_ε). Hence the optimal value of (LP_ε) is a valid lower bound on the competitive ratio α_ε.

$$
\begin{aligned}
&\text{minimize } \alpha_\varepsilon && && && (\mathrm{LP}_\varepsilon)\\
&\text{s.t. } N_0 + \textstyle\sum_{x\in\mathcal{S}_\varepsilon}(1-x)\cdot N_x && \geq W - 1/W^c && && (\mathrm{Vol}_\varepsilon)\\
&\quad N_0 + \textstyle\sum_{x\in\mathcal{S}_\varepsilon} N_x && \leq \alpha_\varepsilon \cdot W && && (\mathrm{small}_\varepsilon)\\
&\quad N_0 + \textstyle\sum_{x\in\mathcal{S}_\varepsilon, x\leq \ell-\varepsilon} N_x + \left\lceil \frac{W}{1-\ell} \right\rceil && \leq \alpha_\varepsilon \cdot \left\lceil \frac{W}{1-\ell} \right\rceil && \forall \ell \in \mathcal{S}_\varepsilon && (\mathrm{CR}_\varepsilon)\\
&\quad N_x \geq 0
\end{aligned}
$$

The claimed lower bound on the a.c.r of recourse-less algorithms follows from Lemma 2.2.

▶ **Lemma 2.2.** *The optimal value $\alpha^\star_\varepsilon$ of (LP_ε) satisfies $\alpha^\star_\varepsilon \in [\alpha - \mathcal{O}(\varepsilon), \alpha + \mathcal{O}(\varepsilon)]$.*

To extend the above argument to the fully-dynamic case, we observe that any solution to (LP_ε) defined by packing of $\mathcal{I}_s$ as above must satisfy some constraint (CR_ε) for some $\ell \in \mathcal{S}_\varepsilon$ with equality, implying a competitive ratio of at least α_ϵ. Now, to beat this bound, a fully-dynamic algorithm must move at least ϵ volume of small items from bins which originally had less than $x - \epsilon$ free space. As small items have size $1/W^c = 1/n^\delta$, this implies that $\Omega(\epsilon n^\delta)$ small items must be moved for every bin affected by such movement. This argument yields Lemma 2.3, which together with Lemma 2.2 implies Theorem 2.1.

▶ **Lemma 2.3.** *For all $\varepsilon > 0$ and $\frac{1}{2} > \delta > 0$, if $\alpha^\star_\varepsilon$ is the optimal value of (LP_ε), then any fully-dynamic* BIN PACKING *algorithm $\mathcal{A}$ with a.c.r $\alpha^\star_\varepsilon - \varepsilon$ and additive term $o(\varepsilon^2 \cdot n^\delta)$ has recourse $\Omega(\varepsilon^2 \cdot n^{1-\delta})$ under unit movement costs.*

2.2 Matching Algorithmic Results

As mentioned earlier, LP_ε also guides our algorithm. For the rest of this section, items smaller than ε are called *small*, and the rest are *large*. Items of size $s_i > 1/2$ are *huge*.

To begin, imagine a total of W volume of small items come first, followed by large items. Inspired by the LP analysis above, we pack the small items such that an N_ℓ/W fraction of bins have ℓ free space for all $\ell \in \{0\} \cup \mathcal{S}_\varepsilon$, where the N_ℓ values are near-optimal for (LP_ε). We call the space profile used by the small items the "curve"; see Figure 1. In the LP analysis above, we showed that this packing can be extended to a packing with a.c.r $\alpha + \mathcal{O}(\varepsilon)$ if $W/(1-\ell)$ items of size ℓ are added. But what if large items of *different* sizes are inserted and deleted? In §2.2.1 we show that this approach retains its $(\alpha + \mathcal{O}(\epsilon))$-a.c.r in this case too, and outline a linear-time algorithm to obtain such a packing.

The next challenge is that the small items may also be inserted and deleted. In §2.2.2 we show how to dynamically pack the small items with bounded recourse, so that the number of bins with any amount of free space $f \in \mathcal{S}_\varepsilon$ induce a near-optimal solution to LP_ε.

Finally, in §2.2.3 we combine the two ideas together to obtain our fully-dynamic algorithm.

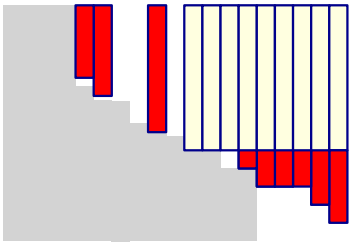


Figure 2 A packing of instance $\mathcal{I}'$. Large items are packed "on top" of the small items (in grey). Parts of large items not in the instance $\mathcal{I}_\ell^k$ are indicated in red.

2.2.1 LP_ε as a Factor-Revealing LP

In this section we show how we use the linear program LP_ε to analyze and guide the following algorithm: pack the small items according to a near-optimal solution to LP_ε, and pack the large items near-optimally "on top" of the small items.

To analyze this approach, we first show it yields a good packing if all large items are huge and have some common size $\ell > 1/2$. Consider an instance $\mathcal{I}_s$ consisting solely of small items with total volume W, packed using N_x bins with gaps in the range $[x, x+\varepsilon)$ for all $x \in \{0\} \cup \mathcal{S}_\varepsilon$, where $\{\alpha_\varepsilon, N_0, N_x : x \in \mathcal{S}_\varepsilon\}$ form a feasible solution for (LP_ε). We say such a packing is α_ε-*feasible*. By the LP's definition, any α_ε-feasible packing of small items can be extended to an α_ε-competitive packing for any extension $\mathcal{I}_\ell$ of $\mathcal{I}_s$ with $\ell \in \mathcal{S}_\varepsilon$, by packing the size ℓ items in the bins counted by N_x for $x \geq \ell$ before using new bins. In fact, this solution satisfies a similar property for any extension $\mathcal{I}_\ell^k$ obtained from $\mathcal{I}_s$ by adding *any* number k of items all of size ℓ. (Note that $\mathcal{I}_\ell$ is the special case of $\mathcal{I}_\ell^k$ with $k = \lfloor W/(1-\ell) \rfloor$.)

▶ **Lemma 2.4** (Huge Items of Same Size). *Any α_ε-feasible packing of small items of $\mathcal{I}_s$ induces an α_ε-competitive packing for all extensions $\mathcal{I}_\ell^k$ of $\mathcal{I}_s$ with $\ell > 1/2$ and $k \in \mathbb{N}$.*

Now, to pack the large items of the instance $\mathcal{I}$, we first create a similar new instance $\mathcal{I}'$ whose large items are all huge items. To do so, we first need the following observation.

▶ **Observation 2.5.** *For any input $\mathcal{I}$ made up of solely large items and function $f(\cdot)$, a packing of $\mathcal{I}$ using at most $(1+\varepsilon)\cdot OPT(\mathcal{I}) + f(\varepsilon^{-1})$ bins has all but at most $2\varepsilon \cdot OPT(\mathcal{I}) + 2f(\varepsilon^{-1}) + 3$ of its bins containing either no large items or being more than half-filled by large items.*

Now, consider a packing $\mathcal{P}$ of the large items of $\mathcal{I}$ using at most $(1+\varepsilon)\cdot OPT(\mathcal{I}) + f(\varepsilon^{-1})$ bins. By Observation 2.5, at most $2\varepsilon \cdot OPT(\mathcal{I}) + \mathcal{O}(f(\varepsilon^{-1}))$ bins in $\mathcal{P}$ are at most half full. We use these bins when packing $\mathcal{I}$. For each of the remaining bins of $\mathcal{P}$ we "glue" all large items occupying the same bin into a single item, yielding a new instance $\mathcal{I}'$ with only huge items, with any packing of $\mathcal{I}'$ "on top" of the small items of $\mathcal{I}$ trivially inducing a similar packing of $\mathcal{I}$ with the same number of bins. We pack the huge items of $\mathcal{I}'$ on top of the curve greedily, repeatedly packing the smallest such item in the fullest bin which can accommodate it. See §2.3.3 for a description of an implementation of this idea with limited recourse. Now, if we imagine we remove and decrease the size of some of these huge items, this results in a new (easier) instance of the form $\mathcal{I}_\ell^k$ for some k and $\ell > 1/2$, packed in no more bins than $\mathcal{I}'$ (see Figure 2). By Lemma 2.4, our packing of $\mathcal{I}'$ (and of $\mathcal{I}$) uses at most $\alpha_\varepsilon \cdot OPT(\mathcal{I}_\ell^k) \leq \alpha_\varepsilon \cdot OPT(\mathcal{I}') \leq (\alpha_\varepsilon + \mathcal{O}(\epsilon)) \cdot OPT(\mathcal{I}) + \mathcal{O}(f(\varepsilon^{-1}))$ bins. Performing the same while only near-optimally packing large items of size exceeding $1/4$, and packing the large items of size in the range $(\varepsilon, 1/4]$ so that we only open a new bin if all bins are at least $3/4$ full (in which case we would obtain a $4/3 < \alpha$ a.c.r) allows us to obtain even better recourse bounds. These ideas underlie the following theorem (a more complete proof of which appears in [15]).

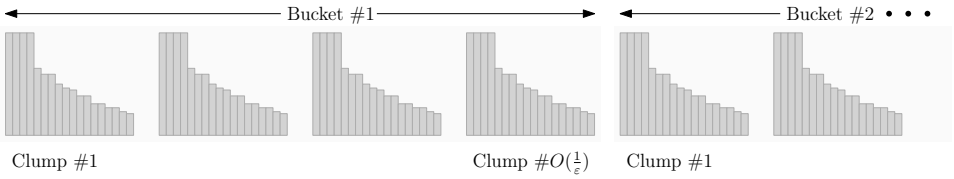


Figure 3 Buckets have $\mathcal{O}(\varepsilon^{-1})$ clumps, clumps have T bins.

▶ **Theorem 2.6.** *An α_ε-feasible packing of the small items of an instance $\mathcal{I}$ can be extended into a packing of $\mathcal{I}$ using at most $(\alpha_\epsilon + \mathcal{O}(\varepsilon)) \cdot OPT(\mathcal{I}) + \mathcal{O}(\varepsilon^{-2})$ bins in linear time. Moreover, given a fully-dynamic $(1+\varepsilon) \cdot OPT(\mathcal{I}) + f(\varepsilon^{-1})$ packing of items of size greater than $1/4$ of $\mathcal{I}_t$ and an α_ε-feasible packing of its small items, one can maintain a packing using $(\alpha_\epsilon + \mathcal{O}(\varepsilon)) \cdot OPT(\mathcal{I}_t) + \mathcal{O}(\max\{\varepsilon^{-2}, f(\varepsilon^{-1}))\}$ bins with $\mathcal{O}(\varepsilon^{-1})$ worst-case recourse per item move in the near-optimal dynamic packing of the items of size exceeding $1/4$.*

It remains to address the issue of maintaining an α_ε-feasible packing of small items dynamically using limited recourse.

2.2.2 Dealing With Small Items: "Fitting a Curve"

We now consider the problem of packing ε-small items according to an approximately-optimal solution of $(\mathrm{LP}_\varepsilon)$. We abstract the problem thus.

▶ **Definition 2.7** (Bin curve-fitting). *Given a list of bin sizes $0 \le b_0 \le b_1 \le \ldots, b_K \le 1$ and relative frequencies $f_0, f_1, f_2, \ldots, f_K$, such that $f_x \ge 0$ and $\sum_{x=0}^{K} f_x = 1$, an algorithm for the* bin curve-fitting problem *must pack a set of m of items with sizes $s_1, \ldots, s_m \le 1$ into a minimal number of bins N such that for every $x \in [0, K]$ the number of bins of size b_x that are used by this packing lie in $\{\lfloor N \cdot f_x \rfloor, \lceil N \cdot f_x \rceil\}$.*

If we have $K = 0$ with $b_0 = 1$ and $f_0 = 1$, we get standard BIN PACKING. We want to solve the problem only for (most of the) small items, in the fully-dynamic setting. We consider the special case with relative frequencies f_x being multiples of $1/T$, for $T \in \mathbb{Z}$; e.g., $T = \mathcal{O}(\varepsilon^{-1})$. Our approach is inspired by the algorithm of [23], and maintains bins in increasing sorted order of item sizes. The number of bins is always an integer product of T. Consecutive bins are aggregated into *clumps* of exactly T bins each, and clumps aggregated into $\Theta(\varepsilon^{-1})$ *buckets* each. Formally, each clump has T bins, with $f_x \cdot T \in \mathbb{N}$ bins of size b_x for $x = 0, \ldots, K$. The bins in a clump are ordered according to their capacity b_x, so each clump looks like its target curve. Each bucket except the last consists of some $s \in [1/\varepsilon, 3/\varepsilon]$ consecutive clumps (the last bucket may have fewer than $1/\epsilon$ clumps). See Figure 3. For each bucket, all bins except those in the last clump are full to within an additive ε.

Inserting an item adds it to the correct bin according to its size. If the bin size becomes larger than the target size for the bin, the largest item overflows into the next bin, and so on. Clearly this maintains the invariant that we are within an additive ε of the target size. We perform $\mathcal{O}(T/\epsilon)$ moves in the same bucket; if we overflow from the last bin in the last clump of the bucket, we add a new clump of T new bins to this bucket, etc. If a bucket contains too many clumps, it splits into two buckets, at no movement cost. An analogous (reverse) process happens for deletes. Loosely, the process maintains that on average the bins are full to within $\mathcal{O}(\varepsilon)$ of the target fullness – one loss of ε because each bin may have ε space, and another because an $\mathcal{O}(\varepsilon)$ fraction of bins have no guarantees whatsoever.

We now relate this process to the value of LP_ε. We first show that setting $T = \mathcal{O}(\varepsilon^{-1})$ and restricting to frequencies which are multiples of ε does not hurt us. Indeed, for us, $b_0 = 1$,

and $b_x = (1-x)$ for $x \in \mathcal{S}_\varepsilon$. Since ($\mathrm{LP}_\varepsilon$) depends on the total volume W of small items, and f_x may change if W changes, it is convenient to work with a normalized version normalized by W, which is essentially (LP_ε) with $W = 1$. Now $n_x = N_x/W$ can be interpreted as just being proportional to number of bins of size b_x, and we can define $f_x = n_x / \sum_x n_x$ to the fraction of bins of size b_x in our solution. However, we also need each f_x to be an integer multiple of $1/T$ for some integer $T = \mathcal{O}(\varepsilon^{-1})$. We achieve this by slightly modifying the LP solution, obtaining the following.

▶ **Lemma 2.8** (Multiples of ε). *For any optimal solution $\{n_x\}$ to (LP_ε) (with $W = 1$) with objective value α_ε, we can construct in linear time a solution $\{\tilde{n}_x\} \subseteq \varepsilon \cdot \mathbb{N}$ with objective value $\alpha_\varepsilon + \mathcal{O}(\varepsilon)$.*

Using a near-optimal solution to (LP_ε) as in the above lemma, we obtain a bin curve-fitting instance with $T = \mathcal{O}(\varepsilon^{-1})$. Noting that if we ignore the last bucket's $\mathcal{O}(T/\varepsilon^{-1}) = \mathcal{O}(\varepsilon^{-2})$ bins in our solution, we obtain an $(\alpha^\star_\varepsilon + \mathcal{O}(\varepsilon))$-feasible packing (with additive term $\mathcal{O}(\varepsilon^{-2})$) of the remaining items by our algorithm above, using $\mathcal{O}(\varepsilon^{-2})$ worst-case recourse. We obtain the following.

▶ **Lemma 2.9** (Small Items Follow the LP). *Let $\varepsilon \leq 1/6$. Using $\mathcal{O}(\varepsilon^{-2})$ worst-case recourse we can maintain packing of small items such that the content of all but $\mathcal{O}(\varepsilon^{-2})$ designated bins in this packing form an $(\alpha^\star_\varepsilon + \mathcal{O}(\varepsilon))$-feasible packing.*

2.2.3 Our Algorithm

From Lemma 2.8 and Lemma 2.9, we can maintain an $(\alpha^\star_\varepsilon + \mathcal{O}(\varepsilon))$-feasible packing of the small items (that is, a packing of inducing a solution to (LP_ε) with objective value $(\alpha^\star_\varepsilon + \mathcal{O}(\varepsilon))$), all while using $\mathcal{O}(\varepsilon^{-2})$ worst-case recourse. From Theorem 2.6, using a $(1+\varepsilon)$-a.c.r packing of the large items one can extend such a packing of the small items into an $(\alpha^\star_\varepsilon + \mathcal{O}(\varepsilon))$-approximate packing for $\mathcal{I}_t$, where $\alpha^\star_\varepsilon \leq \alpha + \mathcal{O}(\varepsilon)$, by Lemma 2.2. It remains to address the recourse incurred by extending this packing to also pack the large items.

Amortized Recourse. Here we periodically recompute in linear time the extension guaranteed by Theorem 2.6. Dividing the time into epochs and lazily addressing updates to large items (doing nothing on deletion and opening new bins on insertion) for $\varepsilon \cdot N$ steps, where N is the number of bins we use at the epoch's start, guarantees a $(\alpha + \mathcal{O}(\varepsilon))$-a.c.r throughout the epoch. Finally, as the number of large items at the end of an epoch of length $\epsilon \cdot N$ is at most $\mathcal{O}(N/\epsilon)$, repacking them incurs $\mathcal{O}(N/\varepsilon)/(\epsilon \cdot N) = \mathcal{O}(\epsilon^{-2})$ amortized recourse.

Worst Case Recourse. To obtain worst-case recourse we rely on the fully-dynamic $(1+\varepsilon)$-a.c.r algorithm of Berndt al. [6] to maintain a $(1+\varepsilon)$-approximate packing of the sub-instance made of items of size exceeding $1/4$ using $\tilde{O}(\varepsilon^{-3})$ size cost recourse, and so at most $\tilde{O}(\varepsilon^{-3})$ item moves (as these items all have size $\Theta(1)$). By Theorem 2.6, our $(\alpha + \mathcal{O}(\varepsilon))$-feasible solution for the small items of $\mathcal{I}_t$ can be extended dynamically to an $(\alpha + (\mathcal{O}(\varepsilon)))$-a.c.r packing of all of $\mathcal{I}_t$, using $\mathcal{O}(\varepsilon^{-1})$ worst-case recourse per item change in the dynamic packing of the items of size exceeding $1/4$ of $\mathcal{I}_t$, yielding an $\tilde{O}(\varepsilon^{-4})$ worst-case recourse bound overall.

From the above discussions we obtain this section's main result – tight algorithms for unit costs with polynomial additive term and recourse.

▶ **Theorem 2.10** (Unit Costs: Upper Bound). *There is a polytime fully-dynamic* BIN PACKING *algorithm which achieves $\alpha + \mathcal{O}(\varepsilon)$ a.c.r, additive term $\mathcal{O}(\varepsilon^{-2})$ and $\mathcal{O}(\varepsilon^{-2})$ amortized recourse, or additive term poly(ε^{-1}) and $\tilde{O}(\varepsilon^{-4})$ worst-case recourse, under unit movement costs.*

2.3 An Algorithm with Improved Worst-Case Recourse

In this section, we present a second algorithm for the Unit Costs model, which uses only $\mathcal{O}(\varepsilon^{-2})$ worst-case recourse. The previous algorithm mainly used the lower bound as borders, and then applied two different algorithms for small and large items, combining the two in a greedy fashion. A near-optimal packing of the large items "on top" of the small items yields the claimed competitive ratio. In this section, we show that a simple fully-dynamic $4/3$-competitive algorithm of Ivković [19] which has recourse only $\mathcal{O}(\varepsilon^{-1})$ for the large items (and only modifies a constant number of bins) combined in the same way yields the same $\alpha + \varepsilon$ a.c.r, but with better worst-case recourse.

For the remainder of the section, we refer to items of size at most $\delta = \varepsilon/15$ as *minor* items and items larger than this size as *major* items. In §2.3.1 we outline our packing of the minor items, which differs slightly from that of §2.2.2 (in particular it only incurs an $\mathcal{O}(\varepsilon^{-1})$, rather than $\mathcal{O}(\varepsilon^{-2})$, additive term). In §2.3.2 we outline Ivković's algorithm and some of its structural properties we rely on for our analysis. Finally, in §2.3.3 we prove that the combination of these solutions in a greedy manner as discussed in §2.2.1 yields an a.c.r of $\alpha + \mathcal{O}(\varepsilon)$. The full proofs of this section are deferred to [11].

The result of this section is summarized in the following theorem:

▶ **Theorem 2.11** (Unit Cost: Improved Upper Bound). *For each $\varepsilon \in (0, 1)$, there exists an algorithm for the Fully-Dynamic* BIN PACKING *Problem with a.c.r. $(1 + \varepsilon) \cdot \alpha$, additive term $\mathcal{O}(\varepsilon^{-1})$ and $\mathcal{O}(\varepsilon^{-2})$ worst-case recourse, under unit movement costs.*

2.3.1 Prospective Packing of Minor Items

For our improved recourse bound, our packing of small items here is essentially identical to that of 2.2.2, so we only highlight the differences here. As before, we pack the small items in bins which are sorted by item sizes, with bins partitioned into buckets of $\mathcal{O}(\varepsilon^{-1})$ clumps of size $\mathcal{O}(\varepsilon^{-1})$, with a prescribed height for each bin in the clump, and one "buffer" clump in each bucket which is potentially empty (with none of its bins exceeding its target filling height). Now, unlike the solution of 2.2.2, the number of bins of each target height will be the same in each clump, which will imply that the fraction of bins of type at most some j (i.e., filling height at most some w_j) is at most some $\mathcal{O}(\varepsilon)$ times the value assigned to it by the LP solution. This is sufficient to obtain the required competitive ratio for instances comprised solely of small items. The advantage of each clump having the same number of bins of each type however has the added benefit that it implies that once a particular clump is "full", (has all its bin of height at least its target height minus $\varepsilon/15$), we can begin packing major items "on top" of the obtained "curve". In particular this implies an additive term of $\mathcal{O}(\varepsilon^{-1})$, rather than $\mathcal{O}(\varepsilon^{-2})$. As before, the worst-case recourse is at most $\mathcal{O}(\varepsilon^{-2})$.

Choice of Parameters

What remains open in the description of the algorithm is the concrete assignment of bin types and the choice of the parameters k and T. Our choice of filling heights is inspired by the parameters from the lower bound by Balogh et al. [4] (or equivalently, of §2.1), but in order to get the desired upper bound instead, each filling height is essentially replaced by the next smaller filling height from the lower bound. Let $\alpha := 1 - 1/(W_{-1}(-2/e^3)+1) \approx 1.3871$ be the value of the lower bound.

For each bin clump of size T, we need to take care of the correct fraction of bins of a certain type j, which we implicitly determine by parameters z_j (for notational convenience, we also write $z'_j := \sum_{i=1}^{j} z_i$). For a minor item workload of W, we aim to create roughly

$z'_k \cdot W$ bins, where we choose $z'_k \coloneqq (1 + \varepsilon/4)\alpha$, hence achieving the desired tight bound for minor items. Note that the frequencies of each bin type as defined in 2.7 can be obtained by $f_i = z_i/z'_k$. The filling height corresponding to bin type k is defined as $w_k \coloneqq y_k \coloneqq (z'_k - 1)/z'_k$. Bins of this type have the largest remaining space, which is $1/z'_k$. Intuitively, the reason is that we do not need to reserve space for major items of size at least $1/z'_k$ as packing these items in exclusive bins still results in an approximation ratio of z'_k.

For the other bin types $j < k$, we now choose the parameters y_j according to the geometric series $y_j = \frac{1}{2}(2 \cdot y_k)^{\frac{j-1}{k-1}}$ (see also [4] for the background on why this is a good choice). The filling heights w_j of the different bin types depend almost directly on these parameters: We set $w_j \coloneqq y_j \ \forall j > 1$ and $w_1 \coloneqq 1 = y_1 + \frac{1}{2}$. This perceived inconsistency is due to the shift of the other w_j (w.r.t. the lower bound) as explained above.

The remaining values for z_j are set such that $z'_j \coloneqq y_k/(y_j(1-y_k))$ holds for all j. The values for z'_j are a result of optimizing the number of bins of type $\leq j$ against a class of bad instances where many items of size $1 - y_j + \varepsilon'$ (for some tiny $\varepsilon' > 0$) are inserted. These items can not be packed into the same bins with such types, however fit into bins of type $> j$. Note that the choice of these parameters results in $1/4 < 1 - 1/z'_k = y_k < \ldots < y_1 = 1/2$ and $3/4 < 2 \cdot (z'_k - 1) = z'_1 < \ldots < z'_k = (1 + \varepsilon/4) \cdot \alpha$.

Based on these parameters, a clump of T bins is organized as follows: We choose the size of a bin clump to be $T = \lceil 4z'_k/\varepsilon\alpha \rceil = \lceil 4/\epsilon \rceil + 1$ and the total number of bins of type $\leq j$ to be $\lceil z'_j/z'_k \cdot T \rceil$. Hence, the number of bins of type 1 is $\lceil z_1/z'_k \cdot T \rceil$, whereas for bin types $j > 1$, it is determined by $\lceil z'_j/z'_k \cdot T \rceil - \lceil z'_{j-1}/z'_k \cdot T \rceil$. Note that the rounding implies that the number of bins of some types may be zero.

Finally, we choose the number of bin types to be $k = \lceil 3/\varepsilon \rceil + 1$.

Analysis

The main goal of the upcoming analysis is to bound the number of bins used for a given payload W of minor items. In the following we state properties of the structure of the algorithm's solution.

In order to count the number of bins in our solution, we need the following technical lemma, which essentially resembles the volume constraint Vol_ε of LP_ε. Remember that $\delta = \varepsilon/15$ denotes the maximum size of a minor item.

▶ **Lemma 2.12.** *A set of full bins which consists of at least $z_j \cdot W$ bins of type j for each bin type j contains a workload of at least W of minor items; i.e., $\sum_{j=1}^{k} z_j W \cdot (w_j - \delta) \geq W$.*

▶ **Lemma 2.13.** *For all $1 \leq j \leq k$, the total number of bins of any type $i \leq j$ is at most $(z'_j + \frac{3}{4}\varepsilon\alpha)W$.*

This bound is mainly used for the analysis in §2.3.3, showing that a packing with the parameters as defined above indeed fits the target curve as desired. However, it also directly implies the approximation ratio for instances where only minor items are present (constraint small_ε in LP_ε).

▶ **Corollary 2.14.** *For an instance with only minor items, the algorithm achieves an a.c.r. of $z'_k + \frac{3}{4}\varepsilon\alpha = (1 + \varepsilon)\alpha$.*

▶ **Lemma 2.15.** *The recourse (regarding minor items) during an insertion or deletion of a minor item is bounded by $\mathcal{O}(\varepsilon^{-2})$.*

2.3.2 Dealing with Major Items

For major items, i.e. items with a size larger than $\delta = \varepsilon/15$, we use an algorithm called *Myopic Packing* (MP) by Ivković [19] which is a simplified version of the *MMP* algorithm by Ivković and Lloyd [21]. The algorithm is essentially a fully-dynamic variant of Johnson's First Fit Grouping Algorithm [24, 25]. For the sake of completeness, and since [19] is not freely available on the Internet and the algorithm is an essential component of our algorithm, we give a detailed description of this algorithm below.

We divide the major items in four sub-groups, depending on their size: A B (big) item has a size in $\left(\frac{1}{2}, 1\right]$, an L (large) item in $\left(\frac{1}{3}, \frac{1}{2}\right]$, an S (small) item in $\left(\frac{1}{4}, \frac{1}{3}\right]$ and an O (other) item in $\left(\frac{\varepsilon}{15}, \frac{1}{4}\right]$. Ignoring additional O items for now, the following bin types of interest can occur: BL, BS, B, LLS, LL, LSS and SSS. The name of the bin type represents the items of type B, L and S contained in that bin. The additional bin types LS, L, SS, S can only occur at most two times in a packing in total, so they induce an additive constant of at most 2 and thus can be ignored for the analysis. Each different type $\tau \in \mathcal{T} := \{BL, BS, B, LLS, LL, LSS, SSS\}$ is given a priority such that we have a total ordering $<$ on $\mathcal{T}$ which is $BL > BS > B > LLS > LL > LSS > SSS$. Out of the listed bin types above, the MP algorithm utilizes all but bins of type LSS.

The general idea of the algorithm is to insert items in a First Fit manner with disregard of items of a smaller type. This is realized by a recursive procedure which takes items out of a dedicated auxiliary storage and inserts them into the regular packing. When an item is inserted into a bin, items of lower types are removed from this bin and added to the auxiliary storage. It is important that the algorithm maintains a structural property called thoroughness, which reflects the structure of the packing as done by the First Fit Grouping Algorithm by Johnson. This property is formally defined in Lemma 2.16.

In more detail, the algorithm does the following: For an insertion of an item a, it is simply added to the auxiliary storage and a procedure to clear the storage is called. For a deletion of a, all items in the same bin as a are removed from the regular packing and added to the auxiliary storage. Then the procedure to clear the storage is called. At the end of an insert or delete operation, only a constant number of items remains in the auxiliary storage and is packed into at most 2 bins. The procedure to clear the auxiliary storage works as follows:

1. Every B item from the auxiliary storage is inserted into the regular packing. The thoroughness property is maintained by successively searching for fitting L and then S items in bins of a lower type to pair with the new B item. The remaining items from the bins from which the L or S item was removed are moved to the auxiliary storage.
2. L and S items from the auxiliary storage are paired with B bins from the regular packing whenever possible. Bins of the same or higher type are not changed in the process, i.e. an L item can only be paired with a B item of a BS or B bin. Other items from the bin in which these items are inserted are moved to the auxiliary storage.
3. As long as there are at least two L items in the auxiliary storage, they are inserted in a new bin and potentially paired with an S item either from the auxiliary storage or from an existing SSS bin to form an LLS bin. If no fitting S item exists, an LL bin is created. If an S item is taken from a regular bin, the remaining items are moved to the auxiliary storage.
4. As long as there are at least three S items in the auxiliary storage, new bins of type SSS are formed with these items and inserted into the regular packing. At the end of this step, the auxiliary storage contains at most one L item and two S items which can be packed into at most two bins.
5. All remaining O items in the auxiliary storage are moved to the regular packing in a first fit manner. This implies that bins that contain any other item type than O are prioritized over bins that exclusively contain O items.

Properties of the Algorithm

For our analysis of the packing, we mainly use the thoroughness property which is ensured by the algorithm and proven in [19]:

▶ **Lemma 2.16** ([19]). *A bin of type $\tau \in \mathcal{T}$ is* thorough *if there do not exist two bins B_1 and B_2 with lower types $\tau_1, \tau_2 \in \mathcal{T}$, $\tau_1, \tau_2 < \tau$ such that items from B_1 and B_2 can be used to form a bin of type τ. In the solution of the MP algorithm, bins of type BL, BS and LLS are thorough.*

We denote by ALG_τ and OPT_τ (e.g. ALG_{BL} and OPT_{BL}) the number of bins of type $\tau \in \mathcal{T}$ in ALG and OPT, respectively. The same notation is adapted for multiple types of bins (e.g. $\text{ALG}_{BL,LLS} = \text{ALG}_{BL} + \text{ALG}_{LLS}$). Using this notation, we get:
$\text{ALG} \leq \text{ALG}_{BL} + \text{ALG}_{BS} + \text{ALG}_B + \text{ALG}_{LLS} + \text{ALG}_{LL} + \text{ALG}_{SSS} + 2$, and
$\text{OPT} \geq \text{OPT}_{BL} + \text{OPT}_{BS} + \text{OPT}_B + \text{OPT}_{LLS} + \text{OPT}_{LL} + \text{OPT}_{SSS} + \text{OPT}_{LSS}$.

Note that the only reason these are not equalities is that the number of bins of type LS, L, SS or S is between 0 and 2.

We first argue that we may assume that no O items are part of the input: Consider the case that there is a bin containing only O items in ALG. Then every bin in ALG except one is filled with items of a cumulative size of at least $3/4$. Together with Lemma 2.13 we then directly get an approximation factor of $(1+\varepsilon)\alpha$ even if we would not combine our solutions for minor and major items at all. Regarding the case that there is no such O bin, since MP packs items in a myopic manner, it would produce the same solution if the O items were not part of the input. Hence we may compare the solution to an instance of the optimal solution, which does not need to consider any O items in its instance.

In order to ease the analysis of the algorithm, we introduce assumptions to the optimal solution and show that these do not increase the value of the optimal solution.

▶ **Lemma 2.17.** *Let* ALG *be the packing of the MP algorithm of a set of major items. For an optimal solution* OPT *of a packing of the same items, the following properties can be assumed without increasing the number of used bins in* OPT*:*

1. OPT *does not pack a BS bin containing a B item that is part of a BL bin in* ALG*.*
2. OPT *does not pack a B bin containing a B item that is part of a BL bin in* ALG*.*

We use the thoroughness of the MP algorithm (cf. Lemma 2.16) to show the following four statements that compare the solutions of ALG and OPT with each other. They will later be used in the analysis of the combination of our two approaches.

▶ **Lemma 2.18.** *Let* ALG *be the packing of the MP algorithm and* OPT *an optimal solution:*

1. $\text{ALG}_{BL} + \text{ALG}_{BS} + \text{ALG}_B = \text{OPT}_{BL} + \text{OPT}_{BS} + \text{OPT}_B$
2. $\text{ALG}_{BL} + \text{OPT}_{LSS} \geq 2\,(\text{ALG}_{LLS,LL} - \text{OPT}_{LLS,LL})$
3. $\text{ALG}_{BS} + \text{ALG}_{LL} + 2\text{OPT}_{LSS} \geq 3\,(\text{ALG}_{SSS} - \text{OPT}_{SSS})$
4. $\text{ALG}_{BS} + \text{OPT}_{LLS} + 2\text{OPT}_{LSS} \geq 3\,(\text{ALG}_{SSS} - \text{OPT}_{SSS})$

We finally determine the recourse that occurs during the insertion or deletion of a major item. Note that while considering this, we need to also account for possible O items.

▶ **Lemma 2.19.** *The recourse of the MP algorithm (for major items) is bounded by $\mathcal{O}(\varepsilon^{-1})$.*

2.3.3 Combining Major and Minor Items

The two presented approaches for the packing of minor and major items treat these items independently. These independent solutions now have to be combined into one to reach a good approximation guarantee. Assume we obtain these solutions as two sets of bins, where the bins $(B_1^{\min}, \dots, B_m^{\min})$ are the result of the algorithm for minor items and the bins $\left(B_1^{\text{maj}}, \dots, B_n^{\text{maj}}\right)$ are the result of the algorithm for major items. Note that although we temporarily ignored items of a size in $(\varepsilon/15, 1/4]$ in the analysis for the major items, the algorithm that combines the two solutions of course does not ignore the potential O items.

We first describe the structure of the packing we want to achieve and then show how to maintain that structure over time. The goal is to create pairs of bins with one bin from each solution while not modifying too many pairs in each time step. For ease of description, we still refer to two bins $B_i^{\min}$ and B_j^{maj} as two different bins even though their contents may be packed into the same bin. In such a case, we say that $B_i^{\min}$ is paired with B_j^{maj}.

We want to maintain a greedy-style combination of the two lists of bins, which can be described by the following combination process: The list of bins with minor items is (partially) sorted by their type $(1, \dots, k)$, i.e. bins potentially filled with more minor items appear earlier (the ordering here is different compared to the ordering in §2.3.1 when the minor items are actually packed). The list of bins with major items is sorted by their filling height in decreasing order (regardless of their type). The process iterates over the k bin types in the solution of minor items starting with type 2 (since there is no reserved space in bins of type 1) in increasing order. For each bin $B_i^{\min}$ of type j, we iterate over the bins with major items starting with the bins that have the largest filling height. We pair $B_i^{\min}$ with a bin with major items B_ℓ^{maj} that has a filling height of at most $1 - w_j$ and for which ℓ is minimal, i.e., the first bin with major items whose items fit into the reserved space of the respective (minor item) bin type.

Note that this process incorporates all bins of the minor solution that contain at least one item, including the ones that are part of a buffer clump. We do not use bins that contain no minor items at all (even if they are already present in the minor algorithm as part of a buffer clump). Such a greedy-style packing can be maintained while only modifying $\mathcal{O}(k)$ pairings per changed bin in either one of the two solutions. A major reason for this is that for the bins with minor items, only their type is of interest. The pairings of bins need to be changed if one of the following happens:

A change in the solution of major items: For each insertion or deletion of a major item, the solution of major items is modified independently first. The above described greedy process is then used to determine which bins with major items need to be matched with which types of bins of minor items. The combination is then modified to fit the new solution by switching out the major bins where needed, starting with those which are paired with bins of minor items of type 2.

A change in the solution of minor items: As for the major items, the solution of minor items is first modified independently upon insertion or deletion of a minor item. The modification may add or remove at most one bin of minor items. In this case, the above greedy approach is used to recalculate which bins with major items need to be matched with which types of bins of minor items. The solution is modified accordingly, starting with the matching with bins with minor items of type 2.

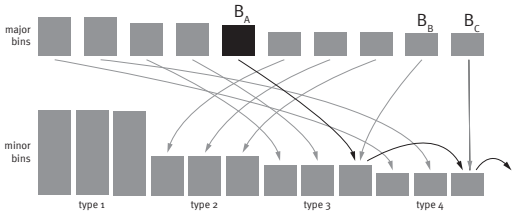


Figure 4 The gray bins represent current bins of the two solutions, the arrows indicate the current combination. The bin B_A is now inserted into the solution for the major items. B_A only fits into minor bins of type 3 or higher. The black arrows indicate the switching process. Bin B_A displaces B_B to a minor bin of type 4. Since there is no bin left for B_C, it is not combined with any minor bin after the changes.

Analysis

Due to the described greedy approach for changes and Lemma 2.15 and 2.19, we can bound the total recourse.

▶ **Lemma 2.20.** *The recourse is bounded by $\mathcal{O}(\varepsilon^{-2})$ for each insertion or deletion.*

▶ **Lemma 2.21.** *Let B^{min} be a bin with minor items of type j. If B^{min} is not combined with a bin of major items, all non-combined bins with major items have a filling height of at least $1 - w_j$.*

It remains to show that our algorithm achieves the desired approximation quality.

▶ **Lemma 2.22.** *Let* Alg *be the number of bins our algorithm uses for an arbitrary input sequence and* Opt *the number of bins used by an optimal solution. Then this yields* $\text{Alg} \leq (1+\varepsilon)\cdot\alpha\cdot\text{Opt}$.

Proof. We reuse the notation of Sections 2.3.1 and 2.3.2. For the bins with major items of Alg, we introduce a collection of bins called *L-S-quartet*, consisting of three LL bins and one SSS bin. The number of such collections is denoted by $Q^{\text{Alg}} := \left\lfloor \min\{\text{Alg}_{SSS}, \frac{1}{3}\text{Alg}_{LL}\} \right\rfloor$. We split the number of LL and SSS bins in Alg in the number of bins that can be put in such quartets, denoted by $\text{Alg}^{Q}_{LL} := 3Q^{\text{Alg}}$ and $\text{Alg}^{Q}_{SSS} := Q^{\text{Alg}}$, and the respective number of bins that cannot be put in such a collection, denoted by Alg^{-Q}_{LL} and Alg^{-Q}_{SSS}. Note that it therefore holds $\text{Alg}_{LL} = \text{Alg}^{Q}_{LL} + \text{Alg}^{-Q}_{LL}$ and $\text{Alg}_{SSS} = \text{Alg}^{Q}_{SSS} + \text{Alg}^{-Q}_{SSS}$. Furthermore, due to the definition of Q^{Alg}, it holds $\text{Alg}^{-Q}_{LL} \leq 2$ or $\text{Alg}^{-Q}_{SSS} = 0$.

Note that we assume that the solution of Alg for the major items does not contain bins with only O items, otherwise the approximation ratio would follow directly as argued in the previous section.

We estimate the optimal solution by adding up all the items which need to be packed. As before, we denote by W the cumulative size of all minor items. We estimate the cumulative size of major items by considering the different types of bins with major items $\tau \in \mathcal{T}$ and their minimal filling height $F(\tau)$ resulting from the minimum size of the respective items (e.g., for bins of type BL, we have $F(BL) = 1/2 + 1/3 = 5/6$). By incorporating the fact that all bins in quartets have an average filling height of $3/4$ we get

$$\text{Opt} \geq W + \sum_{\tau\in\mathcal{T}} F(\tau)\cdot\text{Alg}^{-Q}_{\tau} + \frac{3}{4}\text{Alg}^{Q}_{LL} + \frac{3}{4}\text{Alg}^{Q}_{SSS} \text{ as well as} \tag{1}$$

$$\text{Opt} \geq W + \sum_{\tau\in\mathcal{T}} F(\tau)\cdot\text{Alg}^{-Q}_{\tau} + 3Q^{\text{Alg}}. \tag{2}$$

From Lemma 2.21 we get the following: Suppose j is the maximum index such that a bin of type j is not combined with a bin of major items (pick $j = 1$ if such a bin does not exist). Then all remaining bins with major items must have a size of at least $1 - w_j$. Note that if $j = k$, then $\text{ALG} \leq (z'_k + \frac{3}{4}\varepsilon\alpha)W + z'_k \cdot W^{maj}$ directly follows, where W^{maj} is the workload of major items, and hence the approximation ratio. We assume $j < k$ in the following. We introduce $\text{ALG}^{\geq(1-w_j);-Q}_{BL,BS,B,LLS,LL,SSS}$ as the number of bins (with major items) of the given types used by our algorithm, limited to bins with a filling height of at least $1 - w_j$ and excluding all quartets. Hence we have

$$\begin{aligned}\text{ALG} &\leq (z'_j + \frac{3}{4}\varepsilon\alpha)W + \text{ALG}^{\geq(1-w_j);-Q}_{BL,BS,B,LLS,LL,SSS} + \text{ALG}^{Q}_{LL,SSS} \\ &\leq (z'_j + \frac{3}{4}\varepsilon\alpha)\text{OPT} + \sum_{\tau\in\mathcal{T}}(1 - z'_j \cdot F(\tau)) \cdot \text{ALG}^{\geq(1-w_j);-Q}_{\tau} + (1 - \frac{3}{4}z'_j)\text{ALG}^{Q}_{LL,SSS} \qquad (3)\end{aligned}$$

from (1) as well as

$$\begin{aligned}\text{ALG} &\leq (z'_j + \frac{3}{4}\varepsilon\alpha)W + \text{ALG}^{\geq(1-w_j);-Q}_{BL,BS,B,LLS,LL,SSS} + 4Q^{\text{ALG}} \\ &\leq (z'_j + \frac{3}{4}\varepsilon\alpha)\text{OPT} + \sum_{\tau\in\mathcal{T}}(1 - z'_j \cdot F(\tau)) \cdot \text{ALG}^{\geq(1-w_j);-Q}_{\tau} + (4 - 3z'_j)Q^{\text{ALG}}. \qquad (4)\end{aligned}$$

from (2). We use one of these estimations depending on Q^{ALG}.

Let $\hat{z}_j := z'_k - z'_j$. For the two cases with $\text{ALG}^{-Q}_{LL} \leq 2$ or $\text{ALG}^{-Q}_{SSS} = 0$ we show the following lemmas:

▶ **Lemma 2.23.** *Let* $\text{ALG}^{-Q}_{LL} \leq 2$. *Then it holds that*

$$\text{OPT} \geq \frac{1}{\hat{z}_j} \cdot \left(\sum_{\tau\in\mathcal{T}}(1 - z'_j \cdot F(\tau)) \cdot \text{ALG}^{\geq(1-w_j);-Q}_{\tau} + (1 - \frac{3}{4}z'_j)\text{ALG}^{Q}_{LL,SSS}\right).$$

▶ **Lemma 2.24.** *Let* $\text{ALG}^{-Q}_{SSS} = 0$. *Then it holds that*

$$\text{OPT} \geq \frac{1}{\hat{z}_j} \cdot \left(\sum_{\tau\in\mathcal{T}}(1 - z'_j \cdot F(\tau)) \cdot \text{ALG}^{\geq(1-w_j);-Q}_{\tau} + (4 - 3z'_j)Q^{\text{ALG}}\right).$$

Hence, from (3) together with Lemma 2.23 or (4) together with Lemma 2.24, we conclude

$$\text{ALG} \leq (z'_j + \frac{3}{4}\varepsilon\alpha)\text{OPT} + \hat{z}_j \cdot \text{OPT} \leq (z'_k + \frac{3}{4}\varepsilon\alpha)\text{OPT} = (1 + \varepsilon)\alpha\text{OPT}.$$

◀

Finally, Theorem 2.11 now directly follows from our analysis: Lemma 2.22 gives the approximation ratio of $(1 + \varepsilon) \cdot \alpha$ and Lemma 2.20 bounds the worst-case recourse to $\mathcal{O}(\varepsilon^{-2})$.

3 General Movement Costs

We now consider the case of general movement costs, and show a close connection with the (arrival-only) online problem. We first show that the fully-dynamic problem under general movement costs cannot achieve a better a.c.r than the online problem. Next, we match the a.c.r of any SUPER-HARMONIC algorithm for the online problem in the fully-dynamic setting.

3.1 Matching the Lower Bounds for Online Algorithms

Formally, an *adversary process* $\mathcal{B}$ for the online BIN PACKING problem is an adaptive process that, depending on the state of the system (i.e., the current set of configurations used to pack the current set of items) either adds a new item to the system, or stops the request sequence. We say that an adversary process shows a lower bound of c for online algorithms for BIN PACKING if for any online algorithm $\mathcal{A}$, this process starting from the empty system always eventually reaches a state where the a.c.r is at least c.

▶ **Theorem 3.1.** *Let $\beta \geq 2$. Any adversary process $\mathcal{B}$ showing a lower bound of c for the a.c.r of any online* BIN PACKING *algorithm can be converted into a fully-dynamic* BIN PACKING *instance with general movement costs such that any fully-dynamic* BIN PACKING *algorithm with amortized recourse at most β must have a.c.r at least c.*

Such a claim is simple for *worst-case* recourse. Indeed, given a recourse bound β, set the movement cost of the i-th item to be $(\beta + \varepsilon)^{n-i}$ for $\varepsilon > 0$. When the i-th item arrives we cannot repack *any* previous item because their movement costs are larger by a factor of $> \beta$. So this is a regular online algorithm. The argument fails, however, if we allow amortization.

To construct our lower bound instance, we start with an adversary process and create a dynamic instance as follows. Each subsequent arriving item has exponentially decreasing (by a factor β) movement cost. When the next item arrives, our algorithm could move certain existing items. These items would have much higher movement cost than the arriving item, and so, this process cannot happen too often. Whenever this happens, we *reset* the state of the process to an earlier time and remove all jobs arriving in the intervening period. This will ensure that the algorithm always behaves like an online algorithm which has not moved any of the existing items. Since it cannot move jobs too often, the set of existing items which have not been moved by the algorithm grow over time. This idea allows us to show: implies:

▶ **Corollary 3.2.** *No fully-dynamic* BIN PACKING *algorithm with bounded recourse under general movement costs and $o(n)$ additive term is better than* 1.540*-asymptotically competitive.*

3.2 (Nearly) Matching the Upper Bounds for Online Algorithms

We outline some of the key ingredients in obtaining an algorithm with competitive ratio nearly matching our upper bound of the previous section. The first is an algorithm to pack similarly-sized items.

▶ **Lemma 3.3** (Near-Uniform Sizes)**.** *There exists a fully-dynamic* BIN PACKING *algorithm with constant worst-case recourse which given items of sizes $s_i \in [1/k, 1/(k-1))$ for some integer $k \geq 1$, packs them into bins of which all but one contain $k-1$ items and are hence at least $1 - 1/k$ full. (If all items have size $1/k$, the algorithm packs k items in all bins but one.)*

Lemma 3.3 readily yields a 2-a.c.r algorithm with constant recourse (see [15]). We now discuss how to obtain 1.69-a.c.r based on this lemma and the HARMONIC algorithm [29].

The Harmonic Algorithm. The idea of packing items of nearly equal size together is commonly used in online BIN PACKING algorithms. For example, the HARMONIC algorithm [29] packs large items (of size $\geq \epsilon$) as in Lemma 3.3, while packing small items (of size $\leq \varepsilon$) into dedicated bins which are at least $1 - \varepsilon$ full on average, using e.g., FIRSTFIT. This algorithm uses $(1.69 + \mathcal{O}(\varepsilon)) \cdot OPT + \mathcal{O}(\varepsilon^{-1})$ bins [29]. Unfortunately, due to item removals, FIRSTFIT won't suffice to pack small items into nearly-full bins.

To pack small items in a dynamic setting we extend our ideas for the unit cost case, partitioning the bins into *buckets* of $\Theta(1/\epsilon)$ many bins, such that all but one bin in a bucket are $1-\mathcal{O}(\epsilon)$ full, and hence the bins are $1-\mathcal{O}(\epsilon)$ full on average. Since the size and cost are not commensurate, we maintain the small items in sorted order according to their *Smith ratio* (c_i/s_i). However, insertion of a small item can create a large cascade of movements throughout the bucket. We only move items to/from a bin once it has $\Omega(\epsilon)$'s worth of volume removed/inserted (keeping track of erased, or "ghost" items). A potential function argument allows us to show amortized $\mathcal{O}(\epsilon^{-2})$ recourse cost for this approach, implying the following lemma, and by [29], a $(1.69+\mathcal{O}(\varepsilon))$-a.c.r algorithm with $\mathcal{O}(\epsilon^{-2})$ recourse.

▶ **Lemma 3.4.** *For all $\varepsilon \leq \frac{1}{6}$ there exists an asymptotically $(1+\mathcal{O}(\varepsilon))$-competitive* BIN PACKING *algorithm with $\mathcal{O}(\varepsilon^{-2})$ amortized recourse if all items have size at most ε.*

Seiden's Super-Harmonic Algorithms. We now discuss our remaining ideas to match the bounds of any Super-Harmonic algorithm [33] in the fully-dynamic setting. A *Super-harmonic* (SH) algorithm partitions the unit interval $[0,1]$ into $K+1$ intervals $[0,\varepsilon], (t_0=\varepsilon, t_1](t_1,t_2], \ldots, (t_{K-1}, t_K=1]$. Small items (of size $\leq \varepsilon$) are packed into dedicated bins which are $1-\varepsilon$ full. A large item has type i if its size is in the range $(t_{i-1}, t_i]$. The algorithm also colors items blue or red. Each bin contains items of at most two distinct item types i and j. If a bin contains only one item type, all its items are colored the same. If a bin contains two item types $i \neq j$, all type i items are colored blue and type j ones are colored red (or vice versa). The SH algorithm is defined by four sequences $(\alpha_i)_{i=1}^K, (\beta_i)_{i=1}^K, (\gamma_i)_{i=1}^K$, and a bipartite *compatibility graph* $\mathcal{G}=(V,E)$. A bin with blue (resp., red) type i items contains at most β_i (resp., γ_i) items of type i, and is *open* if it contains less than this many type i items. The compatibility graph $\mathcal{G}=(V,E)$ has vertex set $V=\{b_i \mid i\in[K]\}\cup\{r_j \mid j\in[K]\}$, with an edge $(b_i,r_j)\in E$ indicating blue items of type i and red items of type j are *compatible* and may share a bin. In addition, an SH algorithm must satisfy the following invariants.

- **(P1)** The number of open bins is $\mathcal{O}(1)$.
- **(P2)** If n_i is the number of type-i items, the number of red type-i items is $\lfloor \alpha_i \cdot n_i \rfloor$.
- **(P3)** If $(b_i,r_j)\in E$ (blue type i items and red type j items are compatible), there is no pair of bins with one containing only blue type i items and one containing only red type j items.

Appropriate choice of $(t_i)_{i=1}^{K+1}, (\alpha_i)_{i=1}^K, (\beta_i)_{i=1}^K, (\gamma_i)_{i=1}^K$ and $\mathcal{G}$ allows one to bound the a.c.r of any SH algorithm. (E.g., Seiden gives an SH algorithm with a.c.r 1.589 [33].)

Simulating SH algorithms. In a sense, SH algorithms ignore the exact size of large items, so we can take all items of some type and color. This extends Lemma 3.3 to pack at most β_i or γ_i of them per bin to satisfy Properties 1 and 2. The challenge is in maintaining Property 3: consider a bin with β_i blue type i items and γ_j type-j items, and suppose the type i items are all removed. Suppose there exists an open bin with items of type $i' \neq i$ compatible with j. If the movement costs of both type j and type i' items are significantly higher than the cost of the type i items, we cannot afford to place these groups together, violating Property 3. To avoid such a problem, we use ideas from *stable matchings*. We think of groups of β_i blue type-i items and γ_j red type-j items as nodes in a bipartite graph, with an edge between these nodes if $\mathcal{G}$ contains the edge (b_i,r_j). We maintain a stable matching under updates, with priorities being the value of the costliest item in a group of same-type items packed in the same bin. The stability of this matching implies Property 3; we maintain this stable matching using (a variant of) the Gale-Shapley algorithm. Finally, relying on our solution for packing small items as in §3.2, we can pack the small items in bins which are $1-\varepsilon$ full on average. Combined with the SH bound of Seiden [33], we obtain the following:

▶ **Theorem 3.5.** *There exists a fully-dynamic* BIN PACKING *algorithm with a.c.r* 1.589 *and constant additive term using constant recourse under general movement costs.*

4 Size Movement Costs (Migration Factor)

In this section, we settle the optimal recourse to a.c.r tradeoff for size movement cost (referred to as *migration factor* in the literature); that is, $c_i = s_i$ for each item i. For worst-case recourse in this model (studied in [9, 23, 6]), $poly(\epsilon^{-1})$ upper and lower bounds are known for $(1+\epsilon)$-a.c.r. algorithms [6], though the optimal tradeoff remains elusive. We show that for amortized recourse the optimal tradeoff is $\Theta(\epsilon^{-1})$ recourse for $(1+\epsilon)$-a.c.r.

▶ **Fact 4.1.** *For all $\epsilon \leq 1/2$, there exists an algorithm requiring $(1+\mathcal{O}(\varepsilon))\cdot OPT(\mathcal{I}_t)+\mathcal{O}(\varepsilon^{-2})$ bins at all times t while using only $\mathcal{O}(\varepsilon^{-1})$ amortized migration factor.*

This upper bound is trivial – it suffices to repack according to an AFPTAS whenever the volume changes by a multiplicative $(1+\epsilon)$ factor. The challenge here is in showing this algorithm's a.c.r to recourse tradeoff is *tight*. We do so by constructing an instance where addition or removal of small items of size $\approx \varepsilon$ causes *every* near-optimal solution to be far from every near-optimal solution after addition/removal.

▶ **Theorem 4.2.** *For infinitely many $\epsilon > 0$, any fully-dynamic* BIN PACKING *algorithm with a.c.r $(1+\varepsilon)$ and additive term $o(n)$ must have* amortized *migration factor of $\Omega(\varepsilon^{-1})$.*

Our matching lower bound relies on the well-known Sylvester sequence [34], given by the recurrence relation $k_1 = 2$ and $k_{i+1} = \left(\prod_{j\leq i} k_j\right) + 1$, the first few terms of which are $2, 3, 7, 43, 1807, \ldots$ While this sequence has been used previously in the context of BIN PACKING, our proof relies on more fine-grained divisibility properties. In particular, letting c be a positive integer specified later and $\varepsilon := 1/\prod_{\ell=1}^{c} k_\ell$, we use the following properties:

(P1) $\frac{1}{k_1} + \frac{1}{k_2} + \ldots + \frac{1}{k_c} = 1 - \frac{1}{\prod_{\ell=1}^{c} k_\ell} = 1 - \varepsilon$.

(P2) If $i \neq j$, then k_i and k_j are relatively prime.

(P3) For all $i \in [c]$, the value $1/k_i = \prod_{\ell\in[c]\setminus\{i\}} k_\ell / \prod_{\ell=1}^{c} k_\ell$ is an integer product of ε.

(P4) If $i \neq j \in [c]$, then $1/k_i = \prod_{\ell\in[c]\setminus\{i\}} k_\ell / \prod_{\ell=1}^{c} k_\ell$ is an integer product of $k_j \cdot \epsilon$.

We define a vector of item sizes $\vec{s} \in [0,1]^{c+1}$ in our instances as follows: for $i \in [c]$ we let $s_i = \frac{1}{k_i} \cdot (1 - \frac{\varepsilon}{2})$, and $s_{c+1} = \varepsilon \cdot (\frac{3}{2} - \frac{\varepsilon}{2})$. The adversarial input sequence will alternate between two instances, $\mathcal{I}$ and $\mathcal{I}'$. For some large N a product of $\prod_{\ell=1}^{c} k_\ell$, Instance $\mathcal{I}$ consists of N items of sizes s_i for all $i \in [c+1]$. Instance $\mathcal{I}'$ consists of N items of all sizes but s_{c+1}.

Properties 1-4 imply on the one hand that $\mathcal{I}$ can be packed into completely full bins containing one item of each size, while any bin which does not contain exactly one item of each size has at least $\Omega(\varepsilon)$ free space. Similarly, an optimal packing of $\mathcal{I}'$ packs items of the same size in one set of bins, using up exactly $1 - \frac{\epsilon}{2}$ space, while any bin which contains items of at least two sizes has at least ϵ free space. These observations imply the following.

▶ **Lemma 4.3.** *Any algorithm $\mathcal{A}$ with $(1+\varepsilon/7)$-a.c.r and $o(n)$ additive term packs instance $\mathcal{I}$ such that at least $2N/3$ bins contain exactly one item of each size s_i, and packs instance $\mathcal{I}'$ such that at least $N/2$ bins contain items of exactly one size.*

Theorem 4.2 follows from Lemma 4.3 in a rather straightforward fashion. The full details of this proof and the lemmas leading up to it can be found in [15].

References

1 János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A new and improved algorithm for online bin packing. *arXiv preprint arXiv:1707.01728*, 2017.

2 János Balogh, József Békési, and Gábor Galambos. New lower bounds for certain classes of bin packing algorithms. *Theor. Comput. Sci.*, 440-441:1–13, 2012.

3 János Balogh, József Békési, Gábor Galambos, and Mihály Csaba Markót. Improved lower bounds for semi-online bin packing problems. *Computing*, 84(1-2):139–148, 2009. doi:10.1007/s00607-008-0023-6.

4 János Balogh, József Békési, Gábor Galambos, and Gerhard Reinelt. Lower bound for the online bin packing problem with restricted repacking. *SIAM J. Comput.*, 38(1):398–410, 2008. doi:10.1137/050647049.

5 János Balogh, József Békési, Gábor Galambos, and Gerhard Reinelt. On-line bin packing with restricted repacking. *J. Comb. Optim.*, 27(1):115–131, 2014.

6 Sebastian Berndt, Klaus Jansen, and Kim-Manuel Klein. Fully dynamic bin packing revisited. In *APPROX-RANDOM*, volume 40 of *LIPIcs*, pages 135–151. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

7 Edward G Coffman Jr, János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: survey and classification. In *Handbook of Combinatorial Optimization*, pages 455–531. Springer, 2013.

8 Robert M Corless, Gaston H Gonnet, D EG Hare, David J Jeffrey, and Donald E Knuth. On the Lambert-*W* function. *Advances in Computational mathematics*, 5(1):329–359, 1996.

9 Leah Epstein and Asaf Levin. A robust APTAS for the classical bin packing problem. *Math. Program.*, 119(1):33–49, 2009. doi:10.1007/s10107-007-0200-y.

10 Leah Epstein and Asaf Levin. A robust APTAS for the classical bin packing problem. *Math. Program.*, 119(1):33–49, 2009.

11 Björn Feldkord, Matthias Feldotto, and Sören Riechers. A tight approximation for fully dynamic bin packing without bundling. *arXiv preprint arXiv:1711.01231*, 2017.

12 Giorgio Gambosi, Alberto Postiglione, and Maurizio Talamo. New algorithms for on-line bin packing. In *Algorithms and Complexity, Proceedings of the First Italian Conference*, pages 44–59, 1990.

13 Giorgio Gambosi, Alberto Postiglione, and Maurizio Talamo. Algorithms for the relaxed online bin-packing model. *SIAM J. Comput.*, 30(5):1532–1551, 2000. doi:10.1137/S0097539799180408.

14 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

15 Anupam Gupta, Guru Guruganesh, Amit Kumar, and David Wajc. Fully-dynamic bin packing with limited repacking. *arXiv preprint arXiv:1711.02078*, 2017.

16 Sandy Heydrich and Rob van Stee. Beating the harmonic lower bound for online bin packing. In *43rd International Colloquium on Automata, Languages, and Programming*. Schloss Dagstuhl, 2016.

17 Rebecca Hoberg and Thomas Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2616–2625. SIAM, 2017.

18 Dorit S Hochbaum. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1996.

19 Zoran Ivković. *Fully dynamic approximation algorithms*. PhD thesis, University of Delaware, 1996.

20 Zoran Ivković and Errol L. Lloyd. A fundamental restriction on fully dynamic maintenance of bin packing. *Inf. Process. Lett.*, 59(4):229–232, 1996.

21 Zoran Ivković and Errol L. Lloyd. Fully dynamic algorithms for bin packing: Being (mostly) myopic helps. *SIAM J. Comput.*, 28(2):574–611, 1998. doi:10.1137/S0097539794276749.

22 Zoran Ivković and Errol L. Lloyd. Fully dynamic bin packing. In *Fundamental Problems in Computing*, pages 407–434. Springer, 2009.

23 Klaus Jansen and Kim-Manuel Klein. A robust AFPTAS for online bin packing with polynomial migration,. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 589–600. Springer, 2013. doi:10.1007/978-3-642-39206-1_50.

24 David S. Johnson. Fast allocation algorithms. In *13th Annual Symposium on Switching and Automata Theory, College Park, Maryland, USA, October 25-27, 1972*, pages 144–154. IEEE Computer Society, 1972. doi:10.1109/SWAT.1972.4.

25 David S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

26 David S. Johnson. Fast algorithms for bin packing. *J. Comput. Syst. Sci.*, 8(3):272–314, 1974. doi:10.1016/S0022-0000(74)80026-7.

27 David S. Johnson, Alan Demers, Jeffrey D. Ullman, Michael R Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

28 Edward G. Coffman Jr., M. R. Garey, and David S. Johnson. Dynamic bin packing. *SIAM J. Comput.*, 12(2):227–258, 1983. doi:10.1137/0212014.

29 Chung C. Lee and Der-Tsai Lee. A simple on-line bin-packing algorithm. *J. ACM*, 32(3):562–572, 1985. doi:10.1145/3828.3833.

30 Prakash Ramanan, Donna J Brown, Chung-Chieh Lee, and Der-Tsai Lee. On-line bin packing in linear time. *Journal of Algorithms*, 10(3):305–326, 1989.

31 Michael B Richey. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics*, 34(1-3):203–227, 1991.

32 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009.

33 Steven S. Seiden. On the online bin packing problem. *J. ACM*, 49(5):640–671, 2002. doi:10.1145/585265.585269.

34 James J Sylvester. On a point in the theory of vulgar fractions. *American Journal of Mathematics*, 3(4):332–335, 1880.

35 Jeffrey D. Ullman. The performance of a memory allocation algorithm. Technical report, Princeton University. Department of Electrical Engineering. Computer Science Laboratory, 1971.

36 André van Vliet. *Lower and Upper Bounds for On-line Bin Packing and Scheduling Heuristics: Onder-en Bovengrenzen Voor On-line Bin Packing en Scheduling Heuristieken*. Thesis Publ., 1995.

37 Gerhard Woeginger. Improved space for bounded-space, on-line bin-packing. *SIAM Journal on Discrete Mathematics*, 6(4):575–581, 1993.

38 Prudence W. H. Wong, Fencol C. C. Yung, and Mihai Burcea. An 8/3 lower bound for online dynamic bin packing. In *ISAAC*, volume 7676 of *Lecture Notes in Computer Science*, pages 44–53. Springer, 2012.

39 Andrew Chi-Chih Yao. New algorithms for bin packing. *Journal of the ACM (JACM)*, 27(2):207–227, 1980.

A Sublinear Tester for Outerplanarity (and Other Forbidden Minors) With One-Sided Error

Hendrik Fichtenberger[1]
TU Dortmund, Dortmund, Germany
hendrik.fichtenberger@tu-dortmund.de
https://orcid.org/0000-0003-3246-5323

Reut Levi[2]
Weizmann Institute of Science, Rehovot, Israel
reut.levi@weizmann.ac.il
https://orcid.org/0000-0003-3167-1766

Yadu Vasudev[3]
Indian Institute of Technology Madras, Chennai, India
yadu@cse.iitm.ac.in
https://orcid.org/0000-0001-7918-7194

Maximilian Wötzel[4]
BGSMath and UPC Barcelona, Barcelona, Spain
maximilian.wotzel@upc.edu
https://orcid.org/0000-0001-7591-0998

Abstract

We consider one-sided error property testing of $\mathcal{F}$-minor freeness in bounded-degree graphs for any finite family of graphs $\mathcal{F}$ that contains a minor of $K_{2,k}$, the k-circus graph, or the $(k \times 2)$-grid for any $k \in \mathbb{N}$. This includes, for instance, testing whether a graph is outerplanar or a cactus graph. The query complexity of our algorithm in terms of the number of vertices in the graph, n, is $\tilde{O}(n^{2/3}/\epsilon^5)$. Czumaj et al. (2014) showed that cycle-freeness and C_k-minor freeness can be tested with query complexity $\tilde{O}(\sqrt{n})$ by using random walks, and that testing H-minor freeness for any H that contains a cycles requires $\Omega(\sqrt{n})$ queries. In contrast to these results, we analyze the structure of the graph and show that either we can find a subgraph of sublinear size that includes the forbidden minor H, or we can find a pair of disjoint subsets of vertices whose edge-cut is large, which induces an H-minor.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases graph property testing, minor-free graphs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.52

Related Version A full version of the paper is available at [7], `https://arxiv.org/abs/1707.06126`.

[1] Supported by ERC grant n° 307696.
[2] Supported by ERC-CoG grant 772839.
[3] Supported by ERC grant n° 307696.
[4] Supported by ERC grant n° 307696, the Spanish Ministerio de Economía y Competitividad projects MTM2014-54745-P, MTM2017-82166-P and an FPI grant under the María de Maetzu research grant MDM-2014-0445.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 52; pp. 52:1–52:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We would like to thank Christian Sohler for his helpful suggestions and his comments on a draft of this paper.

1 Introduction

The study of graph minors began with the work of Wagner [18] and Kuratowski [14]. In a seminal work on graph minors, Robertson and Seymour [17] proved the Graph-Minor Theorem, which states that every family of graphs that is closed under forming minors is characterized by a finite list of forbidden minors. As a consequence of their work, they gave a classical decision algorithm with a running time of $\tilde{O}(n^3)$ to verify whether a fixed graph H is a minor of G.

If time complexity which is polynomial, or even linear, in n is considered too costly, then it is often useful to study a relaxed version of the decision problem. In graph property testing, the goal is to test whether a given input graph has a property or is far from the property (according to some metric) while looking at a very small part of the graph (sublinear in the number of vertices). This was first studied by Goldwasser, Goldreich and Ron [10], where the graph was represented as an adjacency matrix. While this model captures properties of dense graphs well, a more natural one for sparse graphs is the *bounded degree* model, first studied by Goldreich and Ron [12], where the graph is given as adjacency lists. In the bounded degree model, the metric is the fraction ϵ of edges of the input graph that have to be modified out of the maximum possible number of edges (the number of vertices times the bound on the maximum degree), and the *query complexity* of a tester is the number of adjacency list entries that the tester looks at. Two-sided (error) property testers are randomized algorithms that are allowed to err on all graphs, while one-sided (error) property testers are required to present a witness against the property when they reject (for more information on property testing, refer to [9, 8]).

In the setting of property testing, Benjamini, Schramm and Shapira [1] conjectured that for any fixed H, H-minor freeness can be tested with $\tilde{O}(\sqrt{n})$ queries by a one-sided tester on bounded degree graphs. Czumaj et al. [3] provided H-minor freeness testers with one-sided error. They showed that C_k-freeness is testable with $\tilde{O}(\sqrt{n})$ queries for bounded degree graphs for $k \geq 3$ and that any one-sided tester for H-minor freeness requires $\Omega(\sqrt{n})$ queries when H contains a cycle. When H is a forest, they showed that there is a one-sided tester for H-minor freeness whose query complexity depends only on ϵ. They consider the problem whether H-minor freeness can be tested with query complexity $o(n)$ for every minor H "the most begging open problem" [3] left open by their work. Apart from cycles and forests, the only further progress they make towards answering this question is for H which is the 4-vertex graph consisting of a triangle and an additional edge. In this paper we make a significant progress on answering this question and on closing the gap between the conjecture of Benjamini et al. [1] and what is known for one-sided error testing of general forbidden minors that contain a cycle.

1.1 Our results

We extend the study of testing H-minor freeness for a fixed graph H with one-sided error. For a finite family of minors $\mathcal{F}$, we say that a graph is $\mathcal{F}$-minor free if it is H-minor free for every $H \in \mathcal{F}$. We obtain a property tester with query complexity $\tilde{O}(n^{2/3}/\epsilon^5)$ for $\mathcal{F}$-minor freeness, where $\mathcal{F}$ is any family of forbidden minors that contains some graph H that is a minor of either the complete bipartite graph $K_{2,k}$, the k-circus graph (see [2]) or the $(k \times 2)$-grid. This implies, for example, that one can test with one-sided error whether a

graph is outerplanar or a cactus graph. We prove the following result (see Theorem 26 for a more general, technical version).

▶ **Theorem 1.** *Given a graph G with degree at most Δ, and a parameter ϵ, for every constant k, there is a one-sided (error) ϵ-tester with query complexity $\tilde{O}(n^{2/3}/\epsilon^5)$ for $\mathcal{F}$-minor freeness, where $\mathcal{F}$ is a family of forbidden minors such that there exists $H \in \mathcal{F}$ and H is a minor of $K_{2,k}$, the k-circus graph or the $(k \times 2)$-grid.*

Setting $\mathcal{F} = \{K_{2,3}, K_4\}$ and $\mathcal{F}$ to consist of the diamond graph (C_4 with a chord), it follows that outerplanarity, and being a cactus graph, can be tested with one-sided error, respectively.

▶ **Corollary 2.** *Testing outerplanarity with one-sided error has query complexity $\tilde{O}(n^{2/3}/\epsilon^5)$.*

▶ **Corollary 3.** *Testing the property of being a cactus graph with one-sided error has query complexity $\tilde{O}(n^{2/3}/\epsilon^5)$.*

1.2 Related Work

As described above, the work most closely related to ours is the paper by Czumaj et al. [3]. Other work in property testing on H-minor freeness studies two-sided error testers. Goldreich and Ron [12] showed that K_3-minor freeness (i.e. cycle-freeness) can be tested with two-sided error and query complexity that is a polynomial in $1/\epsilon$ only. Czumaj, Shapira and Sohler [4] studied a partitioning of graphs that have low expansion (like minor free graphs), which yields two-sided tests for hereditary properties. The problem of testing general H-minor freeness was studied by Benjamini, Schramm and Shapira in [1], where they showed that every minor-closed property of sparse graphs is testable with two-sided error and query complexity independent of the graph's size, although they could only give an upper bound on the query complexity that was triple-exponential in $1/\epsilon$. Hassidim et al. [13] used partition oracles to give an easier two-sided tester for the property of H-minor freeness with query complexity $2^{\text{poly}(1/\epsilon)}$. This was further improved by Levi and Ron [16] to obtain a two-sided tester with query complexity that is quasi-polynomial in $1/\epsilon$. Yoshida and Ito [19] provided testers with two-sided error for outerplanarity and being a cactus graph with query complexity which is only polynomial in $1/\epsilon$ and the bound on the maximum degree.

1.3 Challenges and Techniques

The results of Benjamini et al. [1] imply that the optimal complexity of a two-sided H-minor freeness tester may depend on the size of H (and ϵ and Δ) only. On the other hand, it was proved by Czumaj et al. [3] that the hardness of the one-sided error problem depends on the structure of H. Since all embeddings of H into G may be much larger than H itself, the challenge lies in exploring the proper subgraph to find a witness. However, finding a witness can be worthwhile because it can make the decision of the tester more comprehensible (cf. the discussion in [3]). In [3], the problem of finding C_k-minors is reduced to a tester for bipartiteness [11], which in turn finds odd cycles when two random walks that start from the same vertex collide.

Our algorithm is based on a different approach that employs a partitioning of the graph into sublinear parts. Specifically, the main ingredients in our algorithm are a method to employ a partition of the graph that is derived from a partition into connected parts of size roughly $n^{1/3}$ by Lenzen and Levi [15], and combinatorial lemmas about the existence of H-minors that depends on the number of cut edges between two parts.

In contrast to testing via partition oracles [13, 16], it is not sufficient to only approximate the number of edges between the parts of our partition and, in turn, to reject if this approximated value is too high. This would seem like a limitation to obtaining sublinear query complexity since we can no longer assume that the graph is $\Theta(\epsilon)$-far from being H-minor free after removing the edges going across the parts. In particular, we might have to find a minor that crosses the cut of two parts. An additional obstacle is that we cannot recover the part of a vertex in general because it can be rather large. However, we can show that if the input graph is ϵ-far from being H-minor free, then we can either find a large cut between two parts, which implies the existence of an H-minor, or we can recover a superset of a part that contains an H-minor.

Suppose that G is ϵ-far from being $\mathcal{F}$-minor free, where $\mathcal{F}$ is a family of forbidden minors as in Theorem 1. The algorithm uses a partition into *core clusters* and *remote clusters* of size $\tilde{O}(n^{1/3})$. We draw a uniform sample of edges of constant size (which contains an edge of a minor with constant probability) and distinguish two cases. If an edge belongs to a forbidden minor that is contained in a single cluster, then it suffices to check the cluster for this minor. For core clusters, we can do this by using a partition oracle, but for remote clusters we use a promise on the diameter of the cluster to recover a superset of the remote cluster.

The other case is that a forbidden minor lies across clusters. In particular, we argue that the minor must then lie across core clusters. We show that if the cut between two clusters is greater than some threshold $f = f(H)$, this implies an H-minor (recall that H is either $K_{2,k}$, the k-circus graph or the $(k \times 2)$-grid). In fact, we show that this is true for every pair of disjoint subsets of vertices such that their respective induced subgraphs are connected. Given that, we analyze the edge cut of a coarser partition into *super clusters* and show that if the total size of all cuts that exceed the above-mentioned threshold is small, then actually all edges between clusters can be removed. To obtain access to the partition into super clusters we make use of another coarser partition into *Voronoi cells* for which we also do not have a partition oracle but, roughly speaking, can answer membership queries efficiently.

While we do not attain the upper bound of $\tilde{O}(\sqrt{n})$ conjectured by Benjamini et al. [1], our techniques are significantly different from the ones of Czumaj et al. [3], which is the only other work that gives one-sided testers for minor freeness that we are aware of. Our work throws open two natural questions. The first is whether the technique of partitioning can be used to obtain a one-sided tester for H-minor freeness that matches the conjectured $\tilde{O}(\sqrt{n})$ upper bound. Roughly speaking, the complexity of the algorithm given in Theorem 1, in terms of n, results from the fact that the size of the parts is $\tilde{O}(n^{1/3})$, and the fact that checking to which part a vertex belongs to takes $\tilde{O}(n^{1/3})$ as well. The second question is whether similar techniques can be used to design one-sided testers for a larger class of minors with sublinear query complexity. The limitation of our current approach to the aforementioned minors arises from the inner structure of the parts that we can assume, namely connectivity and bounded diameter. Extending these guarantees, one may hope to find other minors, for example: $(k \times k)$-grid minors; or $K_{3,3}$, which implies testing planarity in sublinear time.

2 Preliminaries

The graphs we consider are simple, undirected, and have a known degree bound Δ. We denote the number of vertices in the graph at hand by n and we assume that each vertex v has a unique id, which for simplicity we also denote by v. There is a total order on the ids, i.e., given any two distinct ids u and v, we can decide whether $u < v$ or $v < u$. The

total order on the vertices induces a total order r on the edges of the graph in the following straightforward manner: $r(\{u,v\}) < r(\{u',v'\})$ if and only if $\min\{u,v\} < \min\{u',v'\}$ or $\min\{u,v\} = \min\{u',v'\}$ and $\max\{u,v\} < \max\{u',v'\}$. The total order over the vertices also induces an order over those vertices visited by a Breadth First Search (BFS) starting from any given vertex v, and whenever we refer to a BFS, we mean that it is performed according to this order. Whenever referring to one of the above orders, we may refer to the *rank* of an element in the respective order. This is simply the index of the respective element when listing all the elements according to the order starting with the smallest.

Let $G = (V,E)$ be a graph, where $V = [n]$. We will say that a graph G is ϵ-far from a property P if at least $\epsilon n\Delta$ edges of G have to be modified in order to convert it into a graph that satisfies the property P. In this paper, the property P that is of interest is H-minor freeness. We will assume that the graph G is represented by a function $f_G : [n] \times [\Delta] \to [n] \cup \{\star\}$, where $f(v,i)$ denotes the i^{th} neighbor of v if v has at least i neighbors. Otherwise, $f_G(v,i) = \star$. We will now define the notion of *one-sided* (error) property testers.

▶ **Definition 4** (One-sided testers)**.** A one-sided (error) ϵ-tester for a property P of bounded degree graphs with query complexity q is a randomized algorithm $\mathcal{A}$ that makes q queries to f_G for a graph G. The algorithm $\mathcal{A}$ accepts if G has the property P. If G is ϵ-far from P, then $\mathcal{A}$ rejects with probability at least $2/3$.

We denote the distance between two vertices u and v in G by $d_G(u,v)$. For vertex $v \in V$ and an integer r, let $\Gamma_r(v,G)$ denote the set of vertices at distance at most r from v. When the graph G is clear from the context, we shall use the shorthands $d(u,v)$ and $\Gamma_r(v)$ for $d_G(u,v)$ and $\Gamma_r(v,G)$, respectively. For a subset of vertices $S \subseteq V$, we denote by $G[S]$ the subgraph induced on S in G.

▶ **Definition 5** (Graph minors)**.** A graph H is a minor of G, if H can be obtained from G by a sequence of vertex deletions, edge deletions and *edge contractions*: For an edge $(u,v) \in G$, delete the vertices u,v, and create a new vertex w. For each neighbor z of either u or v in the graph, add a new edge (w,z).

▶ **Definition 6** ($(k\times 2)$-grid)**.** The $(k\times 2)$-grid is the graph whose vertex set is $\{x_i\}_{i=1}^{k} \cup \{y_i\}_{i=1}^{k}$ and edge set is $\{x_i, x_{i+1}\}_{i=1}^{k-1} \cup \{y_i, y_{i+1}\}_{i=1}^{k-1} \cup \{x_i, y_i\}_{i=1}^{k}$.

▶ **Definition 7** (k-circus graph ([2]))**.** The k-circus graph is the graph whose vertex set is $\{x\} \cup \{y_i\}_{i=1}^{k} \cup \{z_i\}_{i=1}^{k}$ and edge set is $\{(y_i,z_i)\}_{i=1}^{k} \cup \{z_i, z_{i+1}\}_{i=1}^{k-1} \cup \{x, y_i\}_{i=1}^{k}$.

For a graph $G = (V,E)$ and a pair of disjoint subsets of vertices $A \subset V$ and $B \subset V$ let $E_G(A,B) \stackrel{\text{def}}{=} \{(u,v) \in E \,|\, u \in A \wedge v \in B\}$. When it is clear from the context, we omit the subscript. We say that a pair of subsets of vertices A and B is *adjacent* if $E_G(A,B) \neq \emptyset$.

▶ **Definition 8** (Separability)**.** A graph $G = (V,E)$ is (f,g)-*separable* if for every disjoint sets of vertices A and B such that $G[A]$ and $G[B]$ are connected and the diameter of $G[A]$ is at most g it holds that $|E(A,B)| \leq f$.

We shall use the following theorem by Erdős and Szekeres.

▶ **Lemma 9** (Erdős and Szekeres [6])**.** *Given a sequence of natural numbers $S = (s_i)_{i\in[n]}$ of length n, there exists a subsequence of length $\sqrt{n}$ that is either monotonically increasing or monotonically decreasing.*

3 Separability and Evidence for Minors

In this section we prove combinatorial lemmas that give sufficient conditions for the existence of a $(k \times 2)$-grid minor, a k-circus minor and a $K_{2,k}$ minor in a graph. To that end, we will consider the following auxiliary graph, which is defined with respect to a partition of another graph's vertex set.

▶ **Definition 10.** Let $G = (V, E)$ be a graph and $\mathcal{P}$ a partition of its vertex set. The graph $G[\mathcal{P}]$ is defined as the graph with vertex set $\mathcal{P}$, and $\{P, P'\} \subseteq \mathcal{P}$ is an edge if and only if there are vertices $v \in P, u \in P'$ such that $\{u, v\} \in E$.

Notice that if $G[P]$ is connected for every $P \in \mathcal{P}$, then $G[\mathcal{P}]$ is isomorphic to the minor of G obtained by contracting every edge of $G[P]$, for all $P \in \mathcal{P}$, so we will often refer to this minor also by $G[\mathcal{P}]$.

▶ **Lemma 11.** *Given a tree $T = (V, E)$ of bounded degree Δ and a subset of* relevant *vertices $Q \subseteq V$, there exists a partitioning $\mathcal{P}$ of V such that $T[\mathcal{P}]$ is a path minor of T of length $\log_\Delta |Q|$ and for every $P \in \mathcal{P}$, it holds that $P \cap Q \neq \emptyset$.*

Proof. We will construct the claimed partition. Initially, let $\mathcal{P} = \{\{v\} \mid v \in V\}$. While there exists an edge $\{P, P'\}$ in $T[\mathcal{P}]$ such that P' does not contain a relevant vertex and the degree of P' is at most two, update $\mathcal{P} = \mathcal{P} \setminus P'$ and $P = P \cup P'$. Repeat this process until no such edges remain. Note that the resulting $T[\mathcal{P}]$ is still a tree of maximum degree Δ and that every part contains at most one relevant vertex, that is, $|P \cap Q| \leq 1$ for all $P \in \mathcal{P}$.

Since $T[\mathcal{P}]$ is a tree with at least $|Q|$ vertices, it has diameter $\ell \geq \log_\Delta |Q|$. Let $L = (P_1, P_2, \ldots, P_\ell)$ be a simple path in $T[\mathcal{P}]$ between a pair of leaves, of maximum length. For every part $P_i \in L$, let T_i be the subtree rooted at P_i in $T[\mathcal{P}]$ that is obtained by (virtually) removing the (at most two) edges between P_i and its neighbors in L. Every part $P_i \in L$ that contains no relevant vertex has at least one neighbor that is not in L, otherwise it is of degree at most two and would have been merged previously. For every such part, the tree T_i must contain at least one part that has non-empty intersection with Q, otherwise it would have been contracted. We update the partition by setting $P_i = \bigcup_{P \in T_i} P$ for every part $i \in [\ell]$ and setting $\mathcal{P} = \{P_i\}_{i=1}^{\ell}$.

Now, we have that for every $i \in [\ell]$, $P_i \cap Q \neq \emptyset$, and we have also not shortened the path. Therefore, $T[\mathcal{P}]$ is the desired path minor of T. ◀

▶ **Lemma 12.** *Given a rooted tree $T = (V, E)$ of bounded degree Δ with height h (for some $h \in \mathbb{N}$) and a subset of* relevant *vertices $Q \subseteq V$, there exists a partition, $\mathcal{P}$, of V such that $T[\mathcal{P}]$ is a star minor of T with $|Q|/(2h)$ leaves and for every $P \in \mathcal{P}$, it holds that $P \cap Q \neq \emptyset$.*

Proof. W.l.o.g., assume that the root is not in Q. Let S be the set of maximal paths in T that start at the root and end at some vertex in Q. The length of each path in S is bounded by h, and therefore there exist at least $|Q|/h$ such paths. Since all paths in S start at the same vertex and they are maximal, their end vertices are pairwise distinct. Removing all vertices that are not contained in any path of S and contracting all but the end edges of the paths in S, gives a star minor with at least $|Q|/h$ relevant leaves. Finally, we can contract an arbitrary leaf into the root to make the part of the root relevant. ◀

The common idea of the following proofs is to consider a partition of a graph into two parts such that there is a large cut, and to apply Lemma 11 and / or Lemma 12 to these parts in order to construct the desired minors.

▶ **Lemma 13.** *Let $G = (V, E)$ be a graph of bounded degree Δ that does not contain the $(k \times 2)$-grid as a minor. Then, G is $(\Delta^{1+\Delta^{k^2+1}}, n)$-separable.*

Proof. We prove the contrapositive of the statement of the lemma. Let $V_1 \dot\cup V_2$ be a partition of V such that $E(V_1, V_2) > \Delta^{1+\Delta^{k^2+1}}$. Let T_1 (resp. T_2) be a spanning tree of $G[V_1]$ (resp. $G[V_2]$). Let Q_1 be the set of vertices in V_1 that have a neighbor in V_2. Since $|E(V_1, V_2)| \geq \Delta^{1+\Delta^{k^2+1}}$, the size of the set Q_1 is at least $\Delta^{\Delta^{k^2+1}}$. By Lemma 11, there exists a partition $\mathcal{P}_1$ of V_1 such that $T_1[\mathcal{P}_1]$ is a path minor of T_1 of length $r \geq \log_\Delta |Q_1| \geq \Delta^{k^2+1}$. Let $(u_1, u_2, \cdots, u_r)$ be this path minor. For each vertex u_j in the path minor $T_1[\mathcal{P}_1]$, it holds that $|\Gamma(u_j) \cap V_2| \geq 1$. Now, for each vertex u_j in the path minor, remove all the edges to V_2 except the one of lowest rank, so that $|\Gamma(u_j) \cap V_2| = 1$.

Since the degree of G is at most Δ, the number of vertices in V_2 adjacent to Q_1 that remain after these edge deletions is at least Δ^{k^2}. Denote this set of vertices by Q_2. By Lemma 11, there exists a partition $\mathcal{P}_2$ of V_2 such that $T_2[\mathcal{P}_2]$ is a path minor of length $t \geq \log_\Delta |Q_2| \geq k^2$. Let $\{v_1, v_2, \cdots, v_t\}$ be this path. Since $|\Gamma(u_j) \cap T_2[\mathcal{P}_2]| = 1$ for all $u_j \in T_1[\mathcal{P}_1]$, we have $t \leq r$. Furthermore, each vertex $v_j \in T_2[\mathcal{P}_2]$ has at least one neighbor in $T_1[\mathcal{P}_1]$. Therefore, by Hall's theorem, there is matching of size at least k^2 between $T_1[\mathcal{P}_1]$ and $T_2[\mathcal{P}_2]$.

By Lemma 9, there is a set of k vertices, say $u_{i_1}, u_{i_2}, \cdots, u_{i_k}$ and $v_{j_1}, v_{j_2}, \cdots, v_{j_k}$, such that $i_1 \leq i_2 \leq \cdots \leq i_k$ and $j_1 \leq j_2 \leq \cdots \leq j_k$, and (u_{i_l}, v_{j_l}) is an edge in the matching. Contract the edges in the path to the vertices $u_{i_1}, u_{i_2}, \cdots, u_{i_k}$ and $v_{j_1}, v_{j_2}, \cdots, v_{j_k}$ to obtain the $(k \times 2)$-grid minor. ◀

▶ **Lemma 14.** *Let $G = (V, E)$ be a graph of bounded degree Δ that does not contain the k-circus as a minor. Then, G is $(2h\Delta^{2+k^2}, h)$-separable for every $h \in \mathbb{N}$.*

Proof. We prove the contrapositive of the statement of the lemma. This proof is very similar to the proof of Lemma 13. The only difference is that (i) we consider a partition $V_1 \dot\cup V_2$ of V such that $E(V_1, V_2) > 2h\Delta^{2+k^2}$ and the diameter of $G[V_1]$ is at most h and (ii) we apply Lemma 12 instead of Lemma 11 to $G[V_1]$. In particular, let T_1 (resp. T_2) be a spanning tree of $G[V_1]$ (resp. $G[V_2]$). Let Q_1 be the set of vertices in V_1 that have a neighbor in V_2. Since $|E(V_1, V_2)| \geq 2h\Delta^{2+k^2}$, the size of the set Q_1 is at least $2h\Delta^{1+k^2}$. By Lemma 12, there exists a partition $\mathcal{P}_1$ of V_1 such that $T_1[\mathcal{P}_1]$ is a star minor of T_1 with $r \geq \Delta^{1+k^2}$ leaves. Let $\{u_1, u_2, \cdots, u_r\}$ be the leaves of this star minor. For each leaf u_j of the star minor $T_1[\mathcal{P}_1]$, it holds that $|\Gamma(u_j) \cap V_2| \geq 1$. Now, for each leaf u_j in the star minor, remove all the edges to V_2 except the one of lowest rank, so that $|\Gamma(u_j) \cap V_2| = 1$. The remaining proof is analogous to the proof of Lemma 13. ◀

The proof of the following lemma appears in the arXiv version [7].

▶ **Lemma 15.** *Let $G = (V, E)$ be a graph of bounded degree Δ that does not contain the complete bipartite graph $K_{2,k}$ as a minor. Then, G is $(2\Delta kh, h)$-separable for every $h \in \mathbb{N}$.*

4 Underlying Partitions

In this section we describe a method to partition the graph into small connected parts with certain properties that enable us to apply Lemmas 13 to 15. The partition technique is very similar to the one that appears in [15], which is used for the local construction of sparse spanning subgraphs. We make minor adaptations to suit our needs. Omitted proofs appear in the arXiv version [7]. In Section 5, we will show how to utilize these partitions for testing the forbidden minors.

Three different partitions are described next, one of which is a refinement of the other two. As described in more detail in the next sections, the properties of these partitions are as follows. The refined partition into *core clusters* can be locally recovered. The edge cut of this partition is not necessarily small, even if the input graph excludes the forbidden minor. The second partition into *Voronoi cells* will be useful for checking the edge cut of the third partition into *super clusters*, which in turn is guaranteed to have a small edge cut in case the graph excludes the forbidden minor. See the arXiv version [7] for an illustration of the following definitions.

Parameters. The input parameters are α and γ. We sample ℓ uniformly at random from $[b \log n/\log(1+\gamma), b\log n/\log(1+\gamma) + \Delta/\gamma]$, and let $t \stackrel{\text{def}}{=} cn^{1/3}\ln n \cdot \ell\Delta/\alpha$ where c and b are sufficiently large constants. The parameter ℓ affects the diameter of the parts of the partition. It is picked randomly so as to ensure that only a small fraction of the edges are in the edge cut of the partition.

Centers. Pick a set $S \subset |V|$ of $\Theta(\alpha n^{2/3}/\ln n)$ vertices at random. We shall refer to the vertices in S as *centers*. For each vertex $v \in V$, its *center*, denoted by $c(v)$, is the center which is closest to v among all centers (break ties between centers according to the rank).

Remote Vertices. Define $R \stackrel{\text{def}}{=} \{v \mid \Gamma_\ell(v) \cap S = \emptyset\}$ where S is the set of centers. We call the vertices in R *remote* and abbreviate $\bar{R} \stackrel{\text{def}}{=} V \setminus R$.

Voronoi cells. The *Voronoi cell* of a vertex $v \in \bar{R}$ is $\mathsf{Vor}(v) \stackrel{\text{def}}{=} \{u \in \bar{R} \mid c(u) = c(v)\}$.

We deal with the partitioning of remote vertices later. Given a vertex $v \in \bar{R}$, one can determine its center by exploring its ℓ-hop neighborhood. However, it is much more costly to find all vertices that belong to its Voronoi cell, which may have size $\Omega(n)$. We now describe how to further refine the partition given by the Voronoi cells so that the number of vertices in each cluster is $\tilde{O}(n^{1/3}\Delta/\alpha)$.

Core clusters. For each Voronoi cell, consider the BFS tree spanning it, as described in Section 2, which is rooted at the respective center. For every $v \in V$, let $p(v)$ denote the *parent* of v in this BFS tree. If v is a center then $p(v) = v$. For every $v \in V \setminus S$, let $T(v)$ denote the subtree of v in the above-mentioned BFS tree when we remove the edge $\{v, p(v)\}$. Now consider a Voronoi cell. We define the *core cluster* of a vertex v as follows:

1. If $|\mathsf{Vor}(v)| \leq t$ then the core cluster of v is $\mathsf{Vor}(v)$.
2. If $|T(v)| \geq t$, where $|T(v)|$ denotes the number of vertices in $T(v)$, then the core cluster of v is the singleton $\{v\}$.
3. Otherwise, v has a unique ancestor u for which $|T(u)| < t$ and $|T(p(u))| \geq t$. The core cluster of v is the set of vertices in $T(u)$.

For a vertex $v \in \bar{R}$, let $\mathsf{cluster}(v)$ denote the cluster of v. For a cluster C, let $c(C)$ denote the center of the vertices in C (all the vertices in the cluster have the same center). Let $\mathsf{Vor}(C)$ denote the Voronoi cell of the vertices in C.

This describes a partition of V into R and $\bar{R}$, a refinement of $\bar{R}$ into Voronoi cells, and a refinement of this partition into core clusters. It was shown in [15] that the number of core clusters is not much higher than the number of Voronoi cells.

▶ **Lemma 16** (Lemma 1 in [15]). *The number of core clusters, denoted by s, is at most $|S| + n\ell(\Delta+1)/t$.*

Note that core clusters are, like Voronoi cells, connected.

▶ **Lemma 17.** *For every vertex $v \in \bar{R}$, $\mathsf{cluster}(v)$ is connected.*

Even more, Voronoi cells are still connected if one removes a cluster that is not a singleton.

▶ **Lemma 18.** *Let $v \in \bar{R}$ be a vertex such that $\mathsf{cluster}(v)$ is not a singleton. Then, $G[\mathsf{Vor}(v) \setminus \mathsf{cluster}(v)]$ is connected.*

In contrast to Voronoi cells, core clusters are guaranteed to be sufficiently small by construction, which allows us to fully explore them in an efficient manner. An explicit procedure for this is given in Section 5.1. However, it might still be possible for the overall edge cut to be large, even if there are only few edges in individual cuts between two core clusters. To this end, we group core clusters and consider the cut between pairs of a core cluster and such a resulting *super cluster* instead.

▶ **Definition 19.** For a core cluster A, define its *adjacent vertices* $\partial A \stackrel{\text{def}}{=} \{v \mid u \in A \wedge v \in \bar{R} \setminus A \wedge \{u, v\} \in E\}$, i.e., the set of vertices that are adjacent to a vertex in A, excluding A.

▶ **Definition 20.** Define the *adjacent centers* of a set of vertices $A \subseteq \bar{R}$ to be $\{c(v) \mid v \in A\}$.

Marked Clusters. Each center is *marked* independently with probability $p \stackrel{\text{def}}{=} 1/n^{1/3}$. If a center is marked, then we say that its Voronoi cell is marked and all the clusters in this cell are marked as well.

Super Clusters. Let A be a cluster which is not marked but is adjacent to at least one marked cluster. Let $\{u, v\}$ be the edge with minimum rank such that $u \in A$ and $v \in B$, where B is a marked cluster. We say that the cluster A *joins* the cluster B. The *super cluster* of B consists of B and all the clusters which join B.

After considering pairs of clusters and super clusters (and bounding the number of edges between them), only few pairs of core clusters (A, B) are left such that neither A nor B are member of a super cluster with high probability.

▶ **Lemma 21.** *With probability at least $1 - o(1)$, it holds that $|c(\partial A)| \leq n^{1/3} \log n$ for every core cluster A that is not adjacent to a marked cluster.*

This settles the three partitions of vertices from $\bar{R}$ into Voronoi cells, core clusters and super clusters. We now describe a way to partition the remote vertices into remote clusters such that (with high probability) the total number of edges that go out from each remote cluster is at most $O(\gamma n \Delta)$ even if the graph is far from being H-minor free. Basically, this implies that one can test a remote cluster isolated from the remaining graph because all outgoing edges can be removed such that G which was $(\alpha + \gamma)$-far from being H-minor free is still α-far from the property. The partitioning uses ideas of Elkin and Neiman [5].

Remote clusters. We will first describe the algorithm of Elkin and Neiman [5]. Given an integer h and a parameter $0 < \delta \leq 1$, each vertex v draws r_v according to the exponential distribution with parameter $\beta = \ln(n/\delta)/h$. By Claim 2.3 in [5], with probability at least $1 - \delta$, it holds that $r_v < h$ for all $v \in V$. Each vertex v receives r_u from every vertex u within distance at most h, and stores the values $m_u(v) = r_u - d(u, v)$. We use this technique to obtain a partition of R as follows. Every vertex $v \in R$ is *assigned* to the vertex $u \in R$ such that $m_u(v) = \max_{w \in R}\{m_w(v)\}$, if there is more than one such vertex, pick the one with minimum rank. We say that u is the *leader* of v denoted by $L(v)$ and that $\{w \in R \,|\, L(w) = L(v)\}$ is the *remote cluster* of v. We note that we run this algorithm on $G[R]$ (namely, we calculate $m_u(v) = r_u - d_{G[R]}(u, v)$), therefore a vertex u can not be assigned to a vertex on a different connected component in $G[R]$.

Like core clusters, remote clusters are also connected.

▶ **Lemma 22.** *For every $v \in R$, the subgraph induced on the remote cluster of v is connected.*

Similarly[5] as in [5], we define $C(v) = \{u \mid m_u(v) \geq \max_{w \in V}\{m_w(v) - 1\}\}$, for every $v \in R$. We will use an observation about the size of this set to argue that the number of cut-edges between remote clusters is small.

▶ **Lemma 23** (Proof of Lemma 2.2 in [5]). *For every $v \in R$, $\mathrm{Exp}[C(v)] \leq (n/\delta)^{1/h}$.*

For $\delta = 1/n^{b-1}$ and $h = \ell$, we obtain that $\mathrm{Exp}[C(v)] \leq (1+\gamma)$. Define the *edge-cut of R* to be $K = \{\{u,v\} \in E \mid v \in R \ \wedge u \in R \wedge (L(u) \neq L(v))\}$.

▶ **Lemma 24.** *With probability at least $99/100$, $|K| \leq 100\gamma\Delta|R| \leq 100\gamma\Delta n$.*

It remains to bound the number of edges between remote clusters and core clusters.

▶ **Lemma 25** (Lemma 6 in [15]). $\mathrm{Exp}[|E(R, \bar{R})|] \leq \gamma n$.

5 The Algorithm

We prove the following result. Theorem 1 follows by plugging in Lemmas 13 to 15.

▶ **Theorem 26.** *Let $\mathcal{F}$ be a finite family of graphs such that there exists $H \in \mathcal{F}$ and $f = f(n, \mathcal{F}, \Delta)$, $g = g(n, \mathcal{F}, \Delta), g \geq \ell$ (where ℓ is defined in Section 4) such that every n-vertex graph that is H-minor free is (f, g)-separable. For every $\epsilon > 0$, there is a one-sided ϵ-tester that given query access to an n-vertex graph, G, with maximum degree Δ, tests whether G is $\mathcal{F}$-minor free (i.e., G is R-minor free, for every $R \in \mathcal{F}$). The query complexity of the tester is $\tilde{O}(n^{2/3}f^3/\epsilon^5)$. If $\mathcal{F}$ includes a planar graph, then the running time is $\tilde{O}(n^{2/3}f^3/\epsilon^5)$ as well.*

We first analyze a global version of the tester (see Algorithm 1) and show how to turn it into a local algorithm in Section 5.1. Our tester draws $\Theta(f/\epsilon)$ edges at random from the input graph G. It follows that at least one of these edges is part of a forbidden minor with constant probability if G is ϵ-far from being $\mathcal{F}$-minor free. For the sake of this exposition, think of the graph being partitioned into core clusters and remote clusters. To reveal a forbidden minor from $\mathcal{F}$, the algorithm employs this partitioning.

We conduct the following case analysis. Either, an edge that is one out of many that connect a cluster A and an adjacent (disjoint) super cluster B will be sampled. Since non-remote clusters have diameter at most $\ell \leq g$, by the (f, g)-separability of $\mathcal{F}$-minor free graphs, a large cut between A and B implies the existence of H as a minor (see steps 2c and 2d). Otherwise, one can show that the total number of edges between clusters is at most $\epsilon n\Delta/2$. This implies that the edges between clusters can be removed such that the graph is still $\epsilon/2$-far from being $\mathcal{F}$-minor free. Then, it suffices to look for a minor-instance of a graph from $\mathcal{F}$ inside the clusters of each edge (see step 2b).

Therefore, it suffices to bound the number of edges between clusters under the promise that the edge-cut between every cluster and an adjacent (disjoint) super cluster is small. Since a naive bound over all pairs of clusters is quite costly, we classify edges and analyze each class independently. First, we observe that the total number of edges between remote clusters and clusters is $O(\epsilon n\Delta)$ with constant probability. It remains to bound the number of edges between core clusters. To this end, we analyze the total number of edges between core clusters within the same Voronoi cell, the total number of edges between two unmarked core clusters and the total number of edges between core clusters and super clusters separately.

[5] This set is defined slightly differently in [5], but Lemma 23 applies to this definition as well.

Algorithm 1 Test $\mathcal{F}$-minor freeness.

1. Partition V according to the partition described in Section 4 with parameters $\gamma = \Theta(\epsilon)$ and $\alpha = \Theta(\epsilon/f)$.
2. Sample $\Theta(f/\epsilon)$ random edges from G. For each sampled edge $\{u, v\}$:
   a. Find the cluster for each endpoint, denoted by C_v and C_u, respectively.
   b. If both u and v belong to the same cluster C, check that $G[C]$ is $\mathcal{F}$-minor free, if it is not, then return REJECT.
   c. If either for $w = u$ or for $w = v$ it holds that: $w \in \bar{R}$, $\mathsf{cluster}(w)$ is not a singleton, and $|E(\mathsf{Vor}(w) \setminus \mathsf{cluster}(w), \mathsf{cluster}(w))| > f$, then return REJECT.
   d. If both u and v are in $\bar{R}$ and and both C_u and C_v are not singletons then:
      i. If $|E(C_v, C_u)| > f$ then return REJECT.
      ii. If C_v (and symmetrically for C_u) joins a cluster $C \neq C_u$, then let $A \stackrel{\text{def}}{=} \bigcup_{v \in \partial C \setminus C_u} \mathsf{Vor}(v)$. If $|E((A \cup C) \setminus C_u, C_u)| > f$ then return REJECT.
3. Return ACCEPT.

This covers all relevant edges between clusters at least once and gives an upper bound on their total number.

Theorem 26 follows from the the efficient implementation (Section 5.1) and the correctness of the tester (Section 5.2).

The proof of the running time appears in the arXiv version [7].

5.1 Efficient Implementation

In this subsection we describe how Algorithm 1 can be implemented in query complexity $\tilde{O}(n^{2/3}f^3/\epsilon^5) \cdot \mathrm{poly}(\Delta)$. For a vertex $v \in V$, define $i_v \stackrel{\text{def}}{=} \min_i\{\Gamma_i(v) \geq y\}$ where $y \stackrel{\text{def}}{=} \Theta(n^{1/3}\log^2 n/\alpha)$. Let E denote the event that $\Gamma_{i_v}(v) \cap S \neq \emptyset$ for all $v \in V$. Since w.h.p. E occurs, henceforth we condition on this event. The following subroutines are sufficient in order to implement Algorithm 1:

1. Given a vertex $v \in \bar{R}$, finding $\mathsf{cluster}(v)$. The query complexity of finding $c(v)$ is bounded by $y \cdot \Delta$. That is, $c(v)$ is found after performing a BFS from v for at most i_v levels. Furthermore, the path connecting v and $c(v)$ in $T(c(v))$ is also found (this is the shortest path between v and $c(v)$ of smallest lexicographical order). Therefore it is possible to explore $T(c(v))$ with query complexity $O(y \cdot \Delta)$ per step. In order to determine $\mathsf{cluster}(v)$, it is sufficient to explore $T(v)$ and $T(a)$ up to t vertices, for any ancestor, a, of v. Therefore, the total query complexity is at most $O(\ell t y \Delta) = \tilde{O}(n^{2/3}f^2/\epsilon^4) \cdot \mathrm{poly}(\Delta)$ per iteration.
2. Given a subset of vertices A, finding $c(\partial A)$ can be done in query complexity $y \cdot \Delta$ times the total number of edges which are incident to vertices in A. If A is a cluster then the latter is bounded by $t\Delta$. Therefore we obtain a bound of $\tilde{O}(n^{2/3}f^2/\epsilon^3) \cdot \mathrm{poly}(\Delta)$ queries per iteration of step 2c and step 2(d)ii.
3. Finally, instead of finding the remote-cluster of a vertex $v \in R$ it suffices to find a connected induced subgraph that contains the remote-cluster of v. This is achievable by first finding the corresponding leader and then exploring the ℓ-hop neighborhood of the leader. To find the leader, it is sufficient to explore the ℓ-hop neighborhood of v and then for every vertex in it, to determine whether it is in R or not (determining whether a vertex is in R takes $O(y \cdot \Delta)$ time). With this at hand, it is possible to simulate the result of the leader-decomposition algorithm for v. Observe that both v and the leader of

v are in R, therefore (under the assumption that E occurred) it is possible to explore their ℓ-hop neigberhood in $O(y \cdot \Delta)$ time. The query complexity for each iteration of this step is bounded by $\tilde{O}(n^{2/3}f^2/\epsilon^2) \cdot \text{poly}(\Delta)$.

5.2 Correctness

▶ **Lemma 27.** *Algorithm 1 accepts every graph G that is $\mathcal{F}$-minor free.*

Proof. The completeness of the test is based on the separability of H-minor free graphs. We show that if the algorithm rejects, then G contains a graph from $\mathcal{F}$ as minor.

Step 2b rejects only if G contains a graph from $\mathcal{F}$ as a minor. To apply (f, g)-separability to step 2c, it suffices to note that for any $w \in \bar{R}$ such that $\mathsf{cluster}(w)$ is not a singleton, $G[\mathsf{Vor}[w] \setminus \mathsf{cluster}(w)]$ is connected by Lemma 18.

To apply the separability to step 2(d)i, it suffices to note that C_v and C_u are disjoint and that $G[C_v]$ and $G[C_u]$ are both connected by Lemma 17. To apply the separability to step 2(d)ii, we need to show that $G[(A \cup C) \setminus C_u]$ is connected, since it is clearly disjoint from C_u, the correctness then follows. There are two cases. If $(A \cup C) \cap C_u = \emptyset$, then $G[(A \cup C) \setminus C_u] = G[A \cup C]$ is connected. Otherwise, since C_u is not a singleton and since ∂C contains a vertex v which is in $\mathsf{Vor}(C_u) \setminus C_u$, the claim follows from Lemma 18. ◀

▶ **Lemma 28.** *Algorithm 1 rejects every graph G that is ϵ-far from being H-minor free with probability* 2/3.

Proof. Assume that G is ϵ-far from being H-minor free. Let $\mathcal{P}$ denote the partition obtained by the algorithm (namely, the partition of the entire graph as described in Section 4 with parameters $\gamma = \Theta(\epsilon)$ and $\alpha = \Theta(\epsilon/f)$). We say that an edge $e = \{u, v\}$ violates the separability property with respect to $\mathcal{P}$ if either:

1. There exist a core cluster $A \in \bar{R}$ and a cluster or a super cluster, $B \in \bar{R}$ such that $e \in E(A, B)$ and $|E(A, B)| > f$,
2. or, if either for $w = u$ or for $w = v$ it holds that: $w \in \bar{R}$, $\mathsf{cluster}(w)$ is not a singleton, and $|E(\mathsf{Vor}(w) \setminus \mathsf{cluster}(w), \mathsf{cluster}(w))| > f$.

Let $\mathcal{E}$ denote the set of edges which violate the f-separability property with respect to $\mathcal{P}$. If $|\mathcal{E}| > \alpha n\Delta$, then with probability at least 99/100, the algorithm finds a violation in one of the steps: step 2c, step 2(d)i or step 2(d)ii. Note that we do not need to check any edges between a core cluster and a remote cluster nor any edges between two remote clusters. By Markov's inequality and Lemma 25, with probability at least 99/100, $|E(R, \bar{R})| \leq 100\gamma n$. By Lemma 24, with probability at least 99/100, $|K| \leq 100\gamma n\Delta$. Thus, after removing these $|E(R, \bar{R}) \cup K| \leq \epsilon n\Delta/3$ edges, the graph is still $\epsilon/3$-far from being $\mathcal{F}$-minor free.

Assume that $|\mathcal{E}| \leq \alpha n\Delta$. We will show that with probability at least 96/100, we can separate G into clusters by removing at most $\alpha n\Delta \cdot 500f = \epsilon n\Delta/2$ edges. Therefore, the resulting graph is $\epsilon/2$-far from being $\mathcal{F}$-minor free, and with probability at least 2/3, the algorithm rejects in step 2b.

Separating G into clusters

1. As argued above, $|E(R, \bar{R}) \cup K| \leq 200\gamma n\Delta$ with probability at least 98/100. Therefore, we can separate G into $G[\bar{R}]$ and $G[R_1], \ldots, G[R_j]$, where $R_1, \ldots, R_j$ is the partition of R into remote clusters.
2. Next, we remove all the edges in $\mathcal{E}$ (at most $\alpha n\Delta$). In order to separate each Voronoi cell into its core clusters we simply remove all the edges between different clusters in

the same Voronoi cell. The number of edges which are incident to singleton clusters are at most Δs. Since we removed the edges in $\mathcal{E}$, for a cluster A which is not a singleton, we have that $E(A, \mathsf{Vor}(A) \setminus A) \leq f$. Therefore by removing at most $s(\Delta + f)$ edges we separate all the Voronoi cells into clusters.

3. By Lemma 21, with probability at least $1 - o(1)$, we can separate all the clusters, A, for which $c(\partial A)$ does not contain a marked center by removing at most $3sfp^{-1} \ln n$ edges.
4. The expected number of marked clusters is sp, therefore with probability at least $99/100$ the number of marked clusters is at most $100sp$. Thus, with probability at least $99/100$ the number of pairs $A, B \in \bar{R}$ such that A is a cluster and B is a super cluster is at most $s \cdot 100sp$. Since we removed all edges in $\mathcal{F}$, we have that $E(A, B) < f$ for each such pair. Therefore, the number of edges between clusters and super clusters is at most $100fs^2p$.

Recalling from Lemma 16 that $s = \Theta(\alpha n^{2/3}/\ln n)$, we can choose the constants in α and γ to be small enough such that

$$200\gamma n\Delta + \left[\alpha n\Delta + \Theta\left(\frac{\alpha n^{2/3}}{\ln n}\right)(\Delta + f)\right] + \Theta(\alpha n f) + \Theta\left(\frac{\alpha^2 n f}{\ln^2 n}\right) \leq \epsilon n\Delta/2.$$

Hence, with probability at least $96/100$, we can separate G into clusters and remote clusters by removing at most $\epsilon n\Delta/2$ edges. ◀

References

1. I. Benjamini, O. Schramm, and A. Shapira. Every minor-closed property of sparse graphs is testable. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 393–402, 2008. doi:10.1145/1374376.1374433.
2. H. L. Bodlaender. On Linear Time Minor Tests with Depth-First Search. *Journal of Algorithms*, 14(1):1–23, 1993. doi:10.1006/jagm.1993.1001.
3. A. Czumaj, O. Goldreich, D. Ron, C. Seshadhri, A. Shapira, and C. Sohler. Finding cycles and trees in sublinear time. *Random Structure and Algorithms*, 45(2):139–184, 2014. doi:10.1002/rsa.20462.
4. A. Czumaj, A. Shapira, and C. Sohler. Testing Hereditary Properties of Nonexpanding Bounded-Degree Graphs. *SIAM Journal on Computing*, 38(6):2499–2510, 2009. doi:10.1137/070681831.
5. M. Elkin and O. Neiman. Efficient algorithms for constructing very sparse spanners and emulators. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 652–669, 2017.
6. P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.
7. H. Fichtenberger, R. Levi, Y. Vasudev, and M. Wötzel. A Sublinear Tester for Outerplanarity (and Other Forbidden Minors) With One-Sided Error. *arXiv:1707.06126*, 2018. arXiv:1707.06126.
8. O. Goldreich. Introduction to testing graph properties. In *Property Testing*, pages 105–141. Springer, 2010. URL: http://link.springer.com/chapter/10.1007/978-3-642-16367-8_7.
9. O. Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.
10. O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.
11. O. Goldreich and D. Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999. doi:10.1007/s004930050060.

12 O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002. doi:10.1007/s00453-001-0078-7.

13 A. Hassidim, J. A. Kelner, H. N. Nguyen, and K. Onak. Local graph partitions for approximation and testing. In *Proceedings of the Fiftieth Annual Symposium on Foundations of Computer Science (FOCS)*, pages 22–31, 2009. doi:10.1109/FOCS.2009.77.

14 C. Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15(1):271–283, 1930. URL: http://eudml.org/doc/212352.

15 C. Lenzen and R. Levi. A local algorithm for the sparse spanning graph problem. *arXiv:1703.05418*, 2017. URL: http://arxiv.org/abs/1703.05418.

16 R. Levi and D. Ron. A quasi-polynomial time partition oracle for graphs with an excluded minor. *ACM Trans. Algorithms*, 11(3):24:1–24:13, 2015.

17 N. Robertson and P. D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. doi:10.1016/j.jctb.2004.08.001.

18 K. Wagner. über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114(1):570–590, 1937. doi:10.1007/BF01594196.

19 Y. Yoshida and H. Ito. Testing outerplanarity of bounded degree graphs. *Algorithmica*, 73(1):1–20, 2015. doi:10.1007/s00453-014-9897-1.

Parameterized Low-Rank Binary Matrix Approximation

Fedor V. Fomin
Department of Informatics, University of Bergen, Norway
Fedor.Fomin@uib.no
https://orcid.org/0000-0003-1955-4612

Petr A. Golovach
Department of Informatics, University of Bergen, Norway
Petr.Golovach@uib.no
https://orcid.org/0000-0002-2619-2990

Fahad Panolan
Department of Informatics, University of Bergen, Norway
Fahad.Panolan@uib.no
https://orcid.org/0000-0001-6213-8687

Abstract

We provide a number of algorithmic results for the following family of problems: For a given binary $m \times n$ matrix $\mathbf{A}$ and a nonnegative integer k, decide whether there is a "simple" binary matrix $\mathbf{B}$ which differs from $\mathbf{A}$ in at most k entries. For an integer r, the "simplicity" of $\mathbf{B}$ is characterized as follows.

- Binary r-Means: Matrix $\mathbf{B}$ has at most r different columns. This problem is known to be NP-complete already for $r = 2$. We show that the problem is solvable in time $2^{\mathcal{O}(k \log k)} \cdot (nm)^{\mathcal{O}(1)}$ and thus is fixed-parameter tractable parameterized by k. We also complement this result by showing that when being parameterized by r and k, the problem admits an algorithm of running time $2^{\mathcal{O}(r^{3/2} \cdot \sqrt{k \log k})}(nm)^{\mathcal{O}(1)}$, which is subexponential in k for $r \in o((k/\log k)^{1/3})$.
- Low GF(2)-Rank Approximation: Matrix $\mathbf{B}$ is of $\mathsf{GF}(2)$-rank at most r. This problem is known to be NP-complete already for $r = 1$. It is also known to be W[1]-hard when parameterized by k. Interestingly, when parameterized by r and k, the problem is not only fixed-parameter tractable, but it is solvable in time $2^{\mathcal{O}(r^{3/2} \cdot \sqrt{k \log k})}(nm)^{\mathcal{O}(1)}$, which is subexponential in k for $r \in o((k/\log k)^{1/3})$.
- Low Boolean-Rank Approximation: Matrix $\mathbf{B}$ is of Boolean rank at most r. The problem is known to be NP-complete for $k = 0$ as well as for $r = 1$. We show that it is solvable in subexponential in k time $2^{\mathcal{O}(r2^r \cdot \sqrt{k \log k})}(nm)^{\mathcal{O}(1)}$.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms, Theory of computation → Fixed parameter tractability

Keywords and phrases Binary matrices, clustering, low-rank approximation, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.53

Related Version A full version of the paper is available at [22], `https://arxiv.org/abs/1803.06102`.

Funding The research leading to these results have been supported by the Research Council of Norway via the projects "CLASSIS" and "MULTIVAL".

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 53; pp. 53:1–53:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We thank Daniel Lokshtanov, Syed Mohammad Meesum and Saket Saurabh for helpful discussions on the topic of the paper.

1 Introduction

In this paper we consider the following generic problem. Given a binary $m \times n$ matrix, that is a matrix with entries from domain $\{0,1\}$, $\mathbf{A} = (a_{ij}) \in \{0,1\}^{m \times n}$, the task is to find a "simple" binary $m \times n$ matrix $\mathbf{B}$ which approximates $\mathbf{A}$ subject to some specified constrains. One of the most widely studied error measures is the *Frobenius norm*, which for a matrix $\mathbf{A}$ is defined as

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}.$$

Here the sums are taken over $\mathbb{R}$. Then for a given nonnegative integer k, we want to decide whether there is a matrix $\mathbf{B}$ with certain properties such that $\|\mathbf{A}-\mathbf{B}\|_F^2 \leq k$.

We consider the binary matrix approximation problems when for a given integer r, the approximation binary matrix $\mathbf{B}$

(A1) has at most r pairwise-distinct columns,

(A2) is of $\mathsf{GF}(2)$-rank at most r, and

(A3) is of Boolean rank at most r.

Each of these variants is very well-studied. Before defining each of the problems formally and providing an overview of the relevant results, the following observation is in order. Since we approximate a binary matrix by a binary matrix, in this case minimizing the Frobenius norm of $\mathbf{A}-\mathbf{B}$ is equivalent to minimizing the ℓ_0-norm of $\mathbf{A}-\mathbf{B}$, where the measure $\|\mathbf{A}\|_0$ is the number of non-zero entries of matrix $\mathbf{A}$. We also will be using another equivalent way of measuring the quality of approximation of a binary matrix $\mathbf{A}$ by a binary matrix $\mathbf{B}$ by taking the sum of the Hamming distances between their columns. Let us recall that the *Hamming distance* between two vectors $\mathbf{x},\mathbf{y} \in \{0,1\}^m$, where $\mathbf{x} = (x_1,\ldots,x_m)^\intercal$ and $\mathbf{y} = (y_1,\ldots,y_m)^\intercal$, is $d_H(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{m}|x_i - y_i|$ or, in words, the number of positions $i \in \{1,\ldots,m\}$ where x_i and y_i differ. Then for binary $m \times n$ matrix $\mathbf{A}$ with columns $\mathbf{a}^1,\ldots,\mathbf{a}^n$ and matrix $\mathbf{B}$ with columns $\mathbf{b}^1,\ldots,\mathbf{b}^n$, we define $d_H(\mathbf{A},\mathbf{B}) = \sum_{i=1}^{n} d_H(\mathbf{a}^i,\mathbf{b}^i)$. In other words, $d_H(\mathbf{A},\mathbf{B})$ is the number of positions with different entries in matrices $\mathbf{A}$ and $\mathbf{B}$. Then we have the following.

$$\|\mathbf{A}-\mathbf{B}\|_F^2 = \|\mathbf{A}-\mathbf{B}\|_0 = d_H(\mathbf{A},\mathbf{B}) = \sum_{i=1}^{n} d_H(\mathbf{a}^i,\mathbf{b}^i). \tag{1}$$

Problem (A1): Binary r-Means. By (1), the problem of approximating a binary $m \times n$ matrix $\mathbf{A}$ by a binary $m \times n$ matrix $\mathbf{B}$ with at most r different columns (problem (A1)) is equivalent to the following clustering problem. For given a set of n binary m-dimensional vectors $\mathbf{a}^1,\ldots,\mathbf{a}^n$ (which constitute the columns of matrix $\mathbf{A}$) and a positive integer r, Binary r-Means aims to partition the vectors in at most r clusters, as to minimize the sum of within-clusters sums of Hamming distances to their binary means. More formally,

Binary r-Means

Input: An $m \times n$ matrix $\mathbf{A}$ with columns $(\mathbf{a}^1,\ldots,\mathbf{a}^n)$, $r \in \mathbb{N}$ and a nonnegative integer k.

Question: Is there a positive integer $r' \leq r$, a partition $\{I_1,\ldots,I_{r'}\}$ of $\{1,\ldots,n\}$ and vectors $\mathbf{c}^1,\ldots,\mathbf{c}^{r'} \in \{0,1\}^m$ such that $\sum_{i=1}^{r'}\sum_{j\in I_i} d_H(\mathbf{c}^i,\mathbf{a}^j) \leq k$?

To see the equivalence of Binary r-Means and problem (A1), it is sufficient to observe that the pairwise different columns of an approximate matrix $\mathbf{B}$ such that $\|\mathbf{A}-\mathbf{B}\|_0 \leq k$ can be used as vectors $\mathbf{c}^1,\ldots,\mathbf{c}^{r'}$, $r' \leq r$. As far as the mean vectors are selected, a partition of columns of $\mathbf{A}$ can be obtained by assigning each column-vector $\mathbf{a}^i$ to its closest mean vector $\mathbf{c}^j$ (ties breaking arbitrarily). Then for such clustering the total sum of distances from vectors within cluster to their centers does not exceed k. Similarly, a solution to Binary r-Means can be used as columns (with possible repetitions) of matrix $\mathbf{B}$ such that $\|\mathbf{A}-\mathbf{B}\|_0 \leq k$. For that we put $\mathbf{b}^i = \mathbf{c}^j$, where $\mathbf{c}^j$ is the closest vector to $\mathbf{a}^i$.

This problem was introduced by Kleinberg, Papadimitriou, and Raghavan [37] as one of the examples of segmentation problems. Approximation algorithms for optimization versions of this problem were given by Alon and Sudakov [3] and Ostrovsky and Rabani [53], who referred to it as clustering in the Hamming cube. In bioinformatics, the case when $r = 2$ is known under the name Binary-Constructive-MEC (Minimum Error Correction) and was studied as a model for the Single Individual Haplotyping problem [13].

Binary r-Means can be seen as a discrete variant of the well-known k-Means Clustering. (Since in problems (A2) and (A3) we use r for the rank of the approximation matrix, we also use r in (A1) to denote the number of clusters which is commonly denoted by k in the literature on means clustering.) This problem has been studied thoroughly, particularly in the areas of computational geometry and machine learning. We refer to [1, 6, 39] for further references to the works on k-Means Clustering.

Problem (A2): Low GF(2)-Rank Approximation. Let $\mathbf{A}$ be a $m \times n$ binary matrix. In this case we view the elements of $\mathbf{A}$ as elements of $\mathsf{GF}(2)$, the Galois field of two elements. Then the $\mathsf{GF}(2)$-rank of $\mathbf{A}$ is the minimum r such that $\mathbf{A} = \mathbf{U} \cdot \mathbf{V}$, where $\mathbf{U}$ and $\mathbf{V}$ are $m \times r$ and $r \times n$ binary matrices respectively, and arithmetic operations are over $\mathsf{GF}(2)$. Equivalently, this is the minimum number of binary vectors, such that every column (row) of $\mathbf{A}$ is a linear combination (over $\mathsf{GF}(2)$) of these vectors. Then (A2) is the following problem.

Low GF(2)-Rank Approximation

Input: An $m \times n$-matrix $\mathbf{A}$ over $\mathsf{GF}(2)$, $r \in \mathbb{N}$ and a nonnegative integer k.
Question: Is there a binary $m \times n$-matrix $\mathbf{B}$ with $\mathsf{GF}(2)$-rank $\leq r$ and $\|\mathbf{A}-\mathbf{B}\|_F^2 \leq k$?

Low GF(2)-Rank Approximation arises naturally in applications involving binary data sets and serves as an important tool in dimension reduction for high-dimensional data sets with binary attributes, see [17, 35, 31, 38, 54, 57, 62] for further references and numerous applications of the problem.

Low GF(2)-Rank Approximation can be rephrased as a special variant (over $\mathsf{GF}(2)$) of the problem finding the rigidity of a matrix. (For a target rank r, the *rigidity* of a matrix A over a field $\mathbb{F}$ is the minimum Hamming distance between A and a matrix of rank at most r.) Rigidity is a classical concept in Computational Complexity Theory studied due to its connections with lower bounds for arithmetic circuits [29, 30, 58, 55]. We refer to [41] for an extensive survey on this topic.

Low GF(2)-Rank Approximation is also a special case of a general class of problems approximating a matrix by a matrix with a small non-negative rank. Already Non-negative Matrix Factorization (NMF) is a nontrivial problem and it appears in many settings. In particular, in machine learning, approximation by a non-negative low rank matrix has gained extreme popularity after the influential article in Nature by Lee and Seung [40]. NMF is an ubiquitous problem and besides machine learning, it has been independently introduced

and studied in combinatorial optimization [21, 61], and communication complexity [2, 42]. An extended overview of applications of NMF in statistics, quantum mechanics, biology, economics, and chemometrics, can be found in the work of Cohen and Rothblum [15] and recent books [12, 51, 25].

Problem (A3): Low Boolean-Rank Approximation. Let $\mathbf{A}$ be a binary $m \times n$ matrix. This time we view the elements of $\mathbf{A}$ as *Boolean* variables. The *Boolean rank* of $\mathbf{A}$ is the minimum r such that $\mathbf{A} = \mathbf{U} \wedge \mathbf{V}$ for a Boolean $m \times r$ matrix $\mathbf{U}$ and a Boolean $r \times n$ matrix $\mathbf{V}$, where the product is Boolean, that is, the logical $\wedge$ plays the role of multiplication and $\vee$ the role of sum. Here $0 \wedge 0 = 0$, $0 \wedge 1 = 0$, $1 \wedge 1 = 1$, $0 \vee 0 = 0$, $0 \vee 1 = 1$, and $1 \vee 1 = 1$. Thus the matrix product is over the Boolean semi-ring $(0, 1, \wedge, \vee)$. This can be equivalently expressed as the normal matrix product with addition defined as $1+1 = 1$. Binary matrices equipped with such algebra are called *Boolean matrices*. Equivalently, $\mathbf{A} = (a_{ij}) \in \{0,1\}^{m \times n}$ has the Boolean rank 1 if $\mathbf{A} = \mathbf{x}^\intercal \wedge \mathbf{y}$, where $\mathbf{x} = (x_1, x_2, \ldots, x_m) \in \{0,1\}^m$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in \{0,1\}^n$ are nonzero vectors and the product is Boolean, that is, $a_{ij} = x_i \wedge y_j$. Then the Boolean rank of $\mathbf{A}$ is the minimum integer r such that $\mathbf{A} = \mathbf{a^{(1)}} \vee \cdots \vee \mathbf{a^{(r)}}$, where $\mathbf{a^{(1)}}, \ldots, \mathbf{a^{(r)}}$ are matrices of Boolean rank 1; zero matrix is the unique matrix with the Boolean rank 0. Then Low Boolean-Rank Approximation is defined as follows.

Low Boolean-Rank Approximation

Input: A Boolean $m \times n$ matrix $\mathbf{A}$, $r \in \mathbb{N}$ and a nonnegative integer k.
Question: Is there a Boolean $m \times n$ matrix $\mathbf{B}$ of Boolean rank $\leq r$ and $d_H(\mathbf{A}, \mathbf{B}) \leq k$?

For $r = 1$ Low Boolean-Rank Approximation coincides with Low GF(2)-Rank Approximation but for $r > 1$ these are different problems.

Boolean low-rank approximation has attracted much attention, especially in the data mining and knowledge discovery communities. In data mining, matrix decompositions are often used to produce concise representations of data. Since much of the real data is binary or even Boolean in nature, Boolean low-rank approximation could provide a deeper insight into the semantics associated with the original matrix. There is a big body of work done on Low Boolean-Rank Approximation, see e.g. [7, 9, 17, 43, 48, 49].

P-Matrix Approximation. While at first glance Low GF(2)-Rank Approximation and Low Boolean-Rank Approximation look very similar, algorithmically the latter problem is more challenging. The fact that GF(2) is a field allows to play with different equivalent definitions of rank like row rank and column ranks. We exploit this strongly in our algorithm for Low GF(2)-Rank Approximation. For Low Boolean-Rank Approximation the matrix product is over the Boolean semi-ring and nice properties of the GF(2)-rank cannot be used here (see, e.g. [32]). Our algorithm for Low Boolean-Rank Approximation is based on solving an auxiliary $\mathbf{P}$-Matrix Approximation problem, where the task is to approximate a matrix $\mathbf{A}$ by a matrix $\mathbf{B}$ whose block structure is defined by a given pattern matrix $\mathbf{P}$. It appears, that $\mathbf{P}$-Matrix Approximation is also an interesting problem on its own.

More formally, let $\mathbf{P} = (p_{ij}) \in \{0,1\}^{p \times q}$ be a binary $p \times q$ matrix. We say that a binary $m \times n$ matrix $\mathbf{B} = (b_{ij}) \in \{0,1\}^{m \times n}$ is a $\mathbf{P}$-*matrix* if there is a partition $\{I_1, \ldots, I_p\}$ of $\{1, \ldots, m\}$ and a partition $\{J_1, \ldots, J_q\}$ of $\{1, \ldots, n\}$ such that for every $i \in \{1, \ldots, p\}$, $j \in \{1, \ldots, q\}$, $s \in I_i$ and $t \in J_j$, $b_{st} = p_{ij}$. In words, the columns and rows of $\mathbf{B}$ can be permuted such that the block structure of the resulting matrix is defined by $\mathbf{P}$.

P-Matrix Approximation

Input: An $m \times n$ binary matrix $\mathbf{A}$, a pattern binary matrix $\mathbf{P}$ and a nonnegative integer k.
Question: Is there an $m \times n$ $\mathbf{P}$-matrix $\mathbf{B}$ such that $\|\mathbf{A} - \mathbf{B}\|_F^2 \leq k$?

The notion of $\mathbf{P}$-matrix was implicitly defined by Wulff et al. [60] as an auxiliary tool for their approximation algorithm for the related monochromatic biclustering problem. Since Low GF(2)-Rank Approximation remains NP-complete for $r = 1$ [26], we have that $\mathbf{P}$-Matrix Approximation is NP-complete already for the very simple pattern matrix $P = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$.

1.1 Related work

In this subsection we give an overview of previous related algorithmic and complexity results for problems (A1)–(A3), as well as related problems. Since each of the problems has many practical applications, there is a tremendous amount of literature on heuristics and implementations. In this overview we concentrate on known results about algorithms with proven guarantee, with emphasis on parameterized complexity.

Problem (A1): Binary r-Means. Binary r-Means is trivially solvable in polynomial time for $r = 1$, and as was shown by Feige in [20], is NP-complete for every $r \geq 2$.

PTAS (polynomial time approximation scheme) for optimization variants of Binary r-Means were developed in [3, 53]. Approximation algorithms for more general k-Means Clustering is a thoroughly studied topic [1, 6, 39]. Inaba et al. [33] have shown that the general k-Means Clustering is solvable in time n^{mr+1} (here n is the number of vectors, m is the dimension and r the number of required clusters). We are not aware of any, except the trivial brute-force, exact algorithm for Binary r-Means prior to our work.

Problem (A2): Low GF(2)-Rank Approximation. When the low-rank approximation matrix $\mathbf{B}$ is not required to be binary, then the optimal Frobenius norm rank-r approximation of (not necessarily binary) matrix $\mathbf{A}$ can be efficiently found via the singular value decomposition (SVD). This is an extremely well-studied problem and we refer to surveys for an overview of algorithms for low rank approximation [36, 44, 59]. However, SVD does not guarantee to find an optimal solution in the case when additional structural constrains on the low-rank approximation matrix $\mathbf{B}$ (like being non-negative or binary) are imposed.

In fact, most of these constrained variants of low-rank approximation are NP-hard. In particular, Gillis and Vavasis [26] and Dan et al. [17] have shown that Low GF(2)-Rank Approximation is NP-complete for every $r \geq 1$. Approximation algorithms for the optimization version of Low Boolean-Rank Approximation were considered in [34, 35, 17, 38, 57, 10] among others.

Most of the known results about the parameterized complexity of the problem follows from the results for Matrix Rigidity. Fomin et al. have proved in [24] that for every finite field, and in particular GF(2), Matrix Rigidity over a finite field is W[1]-hard being parameterized by k. This implies that Low GF(2)-Rank Approximation is W[1]-hard when parameterized by k. However, when parameterized by k and r, the problem becomes fixed-parameter tractable. For Low GF(2)-Rank Approximation, the algorithm from [24] runs in time $2^{\mathcal{O}(f(r)\sqrt{k}\log k)}(nm)^{\mathcal{O}(1)}$, where f is some function of r. While the function $f(r)$ is not specified in [24], the algorithm in [24] invokes enumeration of all $2^r \times 2^r$ binary matrices of rank r, and thus the running time is at least double-exponential in r.

Meesum, Misra, and Saurabh [46], and Meesum and Saurabh [47] considered parameterized algorithms for related problems about editing of the adjacencies of a graph (or directed graph) targeting a graph with adjacency matrix of small rank.

Problem (A3): Low Boolean-Rank Approximation. It follows from the rank definitions that a matrix is of Boolean rank $r = 1$ if and only if its $\mathsf{GF}(2)$-rank is 1. Thus by the results of Gillis and Vavasis [26] and Dan et al. [17] Low Boolean-Rank Approximation is NP-complete already for $r = 1$.

While computing $\mathsf{GF}(2)$-rank (or rank over any other field) of a matrix can be performed in polynomial time, deciding whether the Boolean rank of a given matrix is at most r is already an NP-complete problem. Thus Low Boolean-Rank Approximation is NP-complete already for $k = 0$. This follows from the well-known relation between the Boolean rank and covering edges of a bipartite graph by bicliques [28]. Let us briefly describe this equivalence. For Boolean matrix $\mathbf{A}$, let G_A be the corresponding bipartite graph, i.e. the bipartite graph whose biadjacency matrix is $\mathbf{A}$. By the equivalent definition of the Boolean rank, $\mathbf{A}$ has Boolean rank r if and only if it is the logical disjunction of r Boolean matrices of rank 1. But for every bipartite graph whose biadjacency matrix is a Boolean matrix of rank at most 1, its edges can be covered by at most one biclique (complete bipartite graph). Thus deciding whether a matrix is of Boolean rank r is exactly the same as deciding whether edges of a bipartite graph can be covered by at most r bicliques. The latter Biclique Cover problem is known to be NP-complete [52]. Biclique Cover is solvable in time $2^{2^{\mathcal{O}(r)}}(nm)^{\mathcal{O}(1)}$ [27] and unless Exponential Time Hypothesis (ETH) fails, it cannot be solved in time $2^{2^{o(r)}}(nm)^{\mathcal{O}(1)}$ [11].

For the special case $r = 1$ Low Boolean-Rank Approximation and $k \leq \|A\|_0/240$, Bringmann, Kolev and Woodruff gave an exact algorithm of running time $2^{k/\sqrt{\|A\|_0}}(nm)^{\mathcal{O}(1)}$ [10]. (Let us remind that the ℓ_0-norm of a matrix is the number of its non-zero entries.) More generally, exact algorithms for NMF were studied by Cohen and Rothblum in [15]. Arora et al. [5] and Moitra [50], who showed that for a fixed value of r, NMF is solvable in polynomial time. Related are also the works of Razenshteyn et al. [56] on weighted low-rank approximation, Clarkson and Woodruff [14] on robust subspace approximation, and Basu et al. [8] on PSD factorization.

Observe that all these problems could be seen as matrix editing problems. For Binary r-Means, we can assume that $r \leq n$ as otherwise we have a trivial YES-instance. Then the problem asks whether it is possible to edit at most k entries of the input matrix, that is, replace some 0-s by 1-s and some 1-s by 0-s, in such a way that the obtained matrix has at most r pairwise-distinct columns. Respectively, Low GF(2)-Rank Approximation asks whether it is possible to edit at most k entries of the input matrix to obtain a matrix of rank at most r. In $\mathbf{P}$-Matrix Approximation, we ask whether we can edit at most k elements to obtain a $\mathbf{P}$-matrix. A lot of work in graph algorithms has been done on graph editing problems, in particular parameterized subexponential time algorithms were developed for a number of problems, including various cluster editing problems [19, 23].

1.2 Our results and methods

We study the parameterized complexity of Binary r-Means, Low GF(2)-Rank Approximation and Low Boolean-Rank Approximation. We refer to the recent books of Cygan et al. [16] and Downey and Fellows [18] for the introduction to Parameterized Algorithms and Complexity. Our results are summarized in Table 1.

Table 1 Parameterized complexity of low-rank approximation. GF(2) Approx stands for Low GF(2)-Rank Approximation and Bool Approx for Low Boolean-Rank Approximation. We omit the polynomial factor $(nm)^{\mathcal{O}(1)}$ in running times.

	k	r	$k+r$
Binary r-Means	$2^{\mathcal{O}(k \log k)}$, Thm 1	NP-c for $r \geq 2$ [20]	$2^{\mathcal{O}(r^{3/2} \cdot \sqrt{k \log k})}$, Thm 3
GF(2) Approx	W[1]-hard [24]	NP-c for $r \geq 1$ [26, 17]	$2^{\mathcal{O}(r^{3/2} \cdot \sqrt{k \log k})}$, Thm 4
Boolean Approx	NP-c for $k = 0$ [52]	NP-c for $r \geq 1$ [26, 17]	$2^{\mathcal{O}(r2^r \cdot \sqrt{k \log k})}$, Thm 2

Our first main result concerns Binary r-Means. We show (Theorem 1) that the problem is solvable in time $2^{\mathcal{O}(k \log k)} \cdot (nm)^{\mathcal{O}(1)}$. Therefore, Binary r-Means is FPT parameterized by k. Since Low GF(2)-Rank Approximation parameterized by k is W[1]-hard and Low Boolean-Rank Approximation is NP-complete for any fixed $k \geq 0$, we find Theorem 1 quite surprising. The proof of Theorem 1 is based on a fundamental result of Marx [45] about the complexity of a problem on strings, namely Consensus Patterns. We solve Binary r-Means by constructing a two-stage FPT Turing reduction to Consensus Patterns. First, we use the color coding technique of Alon, Yuster, and Zwick from [4] to reduce Binary r-Means to some special auxiliary problem and then show that this problem can be reduced to Consensus Patterns, and this allows us to apply the algorithm of Marx [45].

Our second main result concerns Low Boolean-Rank Approximation. As we mentioned above, the problem is NP-complete for $k = 0$, as well as for $r = 1$, and hence is intractable being parameterized by k or by r only. On the other hand, a simpler Low GF(2)-Rank Approximation is not only FPT parameterized by $k+r$, by [24] it is solvable in time $2^{\mathcal{O}(f(r)\sqrt{k} \log k)}(nm)^{\mathcal{O}(1)}$, where f is some function of r, and thus is *subexponential* in k. It is natural to ask whether a similar complexity behavior could be expected for Low Boolean-Rank Approximation. Our second main result, Theorem 2, shows that this is indeed the case: Low Boolean-Rank Approximation is solvable in time $2^{\mathcal{O}(r2^r \cdot \sqrt{k \log k})}(nm)^{\mathcal{O}(1)}$. The proof of this theorem is technical and consists of several steps. We first develop a subexponential algorithm for solving auxiliary **P**-Matrix Approximation, and then construct an FPT Turing reduction from Low Boolean-Rank Approximation to **P**-Matrix Approximation.

Let us note that due to the relation of Boolean rank computation to Biclique Cover, the result of [11] implies that unless Exponential Time Hypothesis (ETH) fails, Low Boolean-Rank Approximation cannot be solved in time $2^{2^{o(r)}} f(k)(nm)^{\mathcal{O}(1)}$ for any function f. Thus the dependence in r in our algorithm cannot be improved significantly unless ETH fails.

Interestingly, the technique developed for solving **P**-Matrix Approximation can be used to obtain algorithms of running times $2^{\mathcal{O}(r^{3/2} \cdot \sqrt{k \log k})}(nm)^{\mathcal{O}(1)}$ for Binary r-Means and Low GF(2)-Rank Approximation (Theorems 3 and 4 respectively). For Binary r-Means, Theorems 3 provides much better running time than Theorem 1 for values of $r \in o((k \log k)^{1/3})$.

For Low GF(2)-Rank Approximation, comparing Theorem 4 and the running time $2^{\mathcal{O}(f(r)\sqrt{k} \log k)}(nm)^{\mathcal{O}(1)}$ from [24], let us note that Theorem 4 not only slightly improves the exponential dependence in k by $\sqrt{\log k}$; it also drastically improves the exponential dependence in r, from 2^{2^r} to $2^{r^{3/2}}$.

Due to space restrictions, we only give high level descriptions of our algorithms. In Section 2 we sketch the algorithm for Binary r-Means parameterized by k, and in Section 3 we explain how we construct FPT algorithms for Binary r-Means and Low GF(2)-Rank Approximation parameterized by k and r that are subexponential in k. The full proofs and further results can be found in [22].

2 Binary r-Means parameterized by k

In this section we give a description of our FPT algorithm for Binary r-Means that runs in time $2^{\mathcal{O}(k \log k)} \cdot (nm)^{\mathcal{O}(1)}$ (Theorem 1).

Let $(\mathbf{A}, r, k)$ be an instance of Binary r-Means where $\mathbf{A}$ is a matrix with columns $(\mathbf{a}^1, \ldots, \mathbf{a}^n)$. We say that a partition $\{I_1, \ldots, I_{r'}\}$ of $\{1, \ldots, n\}$ for $r' \leq r$ is a *solution* for this instance if there are vectors $\mathbf{c}^1, \ldots, \mathbf{c}^{r'} \in \{0,1\}^m$ such that $\sum_{i=1}^{r'} \sum_{j \in I_i} d_H(\mathbf{c}^i, \mathbf{a}^j) \leq k$. We say that each I_i or, equivalently, the multiset of columns $\{\mathbf{a}^j \mid j \in I_i\}$ (some columns could be the same) is a *cluster* and $\mathbf{c}^i$ the *mean* of the cluster. Observe that given a cluster $I \subseteq \{1, \ldots, n\}$, one can easily compute an optimal mean $\mathbf{c} = (c_1, \ldots, c_m)^\intercal$ as follows. Let $\mathbf{a}^j = (a_{1j}, \ldots, a_{mj})^\intercal$ for $j \in \{1, \ldots, n\}$. For each $i \in \{1, \ldots, m\}$, consider the multiset $S_i = \{a_{ij} \mid j \in I\}$ and put $c_i = 0$ or $c_i = 1$ according to the *majority* of elements in S_i, that is, $c_i = 0$ if at least half of the elements in S_i are 0-s and $c_i = 1$ otherwise. We refer to this construction of $\mathbf{c}$ as the *majority rule*.

An *initial cluster* is an inclusion maximal set $I \subseteq \{1, \ldots, n\}$ such that all the columns in the initial cluster are equal. The property of initial clusters we build upon is that there is always an optimal solution that does not split any of the initial clusters. More formally, we say that a partition $\{I_1, \ldots, I_{r'}\}$ of the columns of matrix $\mathbf{A}$ is *regular* if for every initial cluster I, there is $i \in \{1, \ldots, r'\}$ such that $I \subseteq I_i$. Respectively, if $(\mathbf{A}, r, k)$ is a yes-instance of Binary r-Means, then there is a solution $\{I_1, \ldots, I_{r'}\}$ forming a regular partition and we call such a solution *regular*. In words, in a regular solution every two equal columns of $\mathbf{A}$ are placed in the same cluster. Thus in a regular solution $\{I_1, \ldots, I_{r'}\}$, each cluster I_i is either *simple*, that is, contains exactly one initial cluster and all its columns are equal, or I_i is *composite*, that is composed of several initial clusters. One can show that there is always an optimal solution to Binary r-Means that is regular.

Moreover, we can assume that a solution we seek for is not only regular but has stronger property. Indeed, let $\{I_1, \ldots, I_{r'}\}$ be a regular solution for instance $(\mathbf{A}, r, k)$. Denote by $\mathbf{c}^1, \ldots, \mathbf{c}^{r'}$ the corresponding means of the clusters. Let I_i be a composite cluster of $\{I_1, \ldots, I_{r'}\}$ that contains $h \geq 2$ initial clusters. Then $\sum_{j \in I_i} (\mathbf{c}^i, \mathbf{a}^j) \geq h - 1$. Therefore, for every yes-instance, a regular solution $\{I_1, \ldots, I_{r'}\}$ contains at most k composite clusters; all the remaining clusters are simple. Moreover, the total number of initial clusters used to form the composite clusters is at most $2k$. Note also that if I_i is a simple cluster then for $\mathbf{c}^i = \mathbf{a}^h$ for an arbitrary $h \in I_i$, we have that $\sum_{j \in I_i} (\mathbf{c}^i, \mathbf{a}^j) = 0$, that is, simple clusters do not contribute to the total cost of the solution.

Thus the essence of the problem is to find the way of composing initial clusters into composite ones. More precisely, for the instance $(\mathbf{A}, r, k)$, let $\mathcal{I}$ be the family of the initial clusters. Let $s = |\mathcal{I}|$. Then finding a solution for Binary r-Means is equivalent to finding a set $\mathcal{I}' \subseteq \mathcal{I}$ of size at most $2k$ such that $\mathcal{I}'$ can be used to form at most $r - s + |\mathcal{I}'|$ composite clusters. In other words, we are looking for $\mathcal{I}' \subseteq \mathcal{I}$ of size at most $2k$ such that there is a partition $\{P_1, \ldots, P_t\}$ of $\mathcal{I}'$ with $t \leq r - s + |\mathcal{I}'|$ (each set P_i says which initial clusters of $\mathcal{I}'$ are used to form a composite cluster) and vectors $\mathbf{s}^1, \ldots, \mathbf{s}^t \in \{0,1\}^m$ with the property that

$$\sum_{i=1}^{t} \sum_{I \in P_i} \sum_{j \in I} d_H(\mathbf{s}^i, \mathbf{a}^j) \leq k.$$

If $s \leq r$, then $(\mathbf{A}, r, k)$ is a trivial yes-instance of the problem with $\mathcal{I}$ being a solution. If $r + k < s$, then $(\mathbf{A}, r, k)$ is a trivial no-instance. From now on we assume that $r < s \leq r + k$.

If we color uniformly at random initial clusters with $2k$ colors, then with a "reasonable" probability each composite cluster is composed from initial clusters of different colors. This

will provide us with an additional structural information, which will bring us much closer to a solution. We use the classic *color coding* technique of Alon, Yuster, and Zwick from [4] to distinguish initial clusters of $\mathcal{I}'$ from each other. At the end we obtain a deterministic algorithm but, for simplicity, we describe a randomized Monte-Carlo algorithm here. We color the elements of $\mathcal{I}$ independently and uniformly at random by $2k$ colors $1,\ldots,2k$. Observe that if $(\mathbf{A},r,k)$ is a yes-instance, then at most $2k$ initial clusters in a solution that are included in composite clusters are colored by distinct colors with the probability at least $\frac{(2k)!}{(2k)^{2k}} \geq e^{-2k}$. We say that a solution $\{I_1,\ldots,I_{r'}\}$ for $(\mathbf{A},r,k)$ is a *colorful* solution if all initial clusters that are included in composite clusters of $\{I_1,\ldots,I_{r'}\}$ are colored by distinct colors. We construct an algorithm for finding a colorful solution (if it exists).

Let us fix some coloring of initial clusters in $2k$ colors. For $i\in\{1,\ldots,2k\}$, let $\mathcal{I}_i$ be the set of initial clusters colored by color i. Note that some sets $\mathcal{I}_i$ could be empty. We consider all possible partitions $\mathcal{P}=\{P_1,\ldots,P_t\}$ of nonempty subsets of $\{1,\ldots,2k\}$ such that each set of $\mathcal{P}$ contains at least two elements. Notice that if $(\mathbf{A},r,k)$ has a colorful solution $\{I_1,\ldots,I_{r'}\}$, then there is $\mathcal{P}=\{P_1,\ldots,P_t\}$ such that a cluster I_i of the solution is formed from initial clusters colored with colors P_j for some $j\in\{1,\ldots,t\}$. Moreover, two different composite clusters are colored with colors from different sets of $\mathcal{P}$. We go through all possible partitions $\mathcal{P}$, and if $(\mathbf{A},r,k)$ has a colorful solution, we will find the corresponding partition $\mathcal{P}$. Let us fix a partition $\mathcal{P}=\{P_1,\ldots,P_t\}$. If $s-|P_1|-\cdots-|P_t|+t>r$, we discard the current choice of $\mathcal{P}$. Assume from now that this is not the case. For each $i\in\{1,\ldots,t\}$, we do the following. Let $P_i=\{i_1,\ldots,i_p\}\subseteq\{1,\ldots,2k\}$. Then with this notation $\mathcal{I}_{i_j}$ is the set of initial clusters colored by color i_j. For $j\in\{1,\ldots,p\}$, we use $J^i_j=\bigcup_{I\in\mathcal{I}_{i_j}} I$ to denote the set of indices contained in clusters colored by i_j. We also define $J^i=J^i_1\cup\cdots\cup J^i_p$, which is the set of indices contained in clusters colored by colors from $\{i_1,\ldots,i_p\}$. Denote by $\mathbf{A}_i$ the submatrix of $\mathbf{A}$ containing the columns $\mathbf{a}^h$ with $h\in J^i$. We want to solve an auxiliary problem of finding the minimum integer $d_i\leq k$ such that there is a set of initial clusters $L^i_1,\ldots,L^i_p$ and a vector $\mathbf{s}^i\in\{0,1\}^m$ such that $L^i_j\subseteq J^i_j$ for $i\in\{1,\ldots,p\}$ and $\sum_{i=1}^p\sum_{j\in L^i_j} d_H(\mathbf{s}^i,\mathbf{a}^j)\leq d_i$. In words, for a set of colors $P_i=\{i_1,\ldots,i_p\}$, we want to find the best selection of initial clusters $L^i_1,\ldots,L^i_p$ such that each of the clusters L^i_j is colored by i_j; the best is in the sense that the total Hamming distance d_i from the columns corresponding to the selected set of clusters to their means is the minimum over all such selections.

Assume that we have an algorithm for this auxiliary problem. If such a value of d_i does not exist for some $i\in\{1,\ldots,t\}$, we discard the current choice of $\mathcal{P}$. Otherwise, we find the set of clusters $L^i_1,\ldots,L^i_p$ and $\mathbf{s}^i$. Let $L^i=L^i_1\cup\cdots\cup L^i_p$. We check whether $d_1+\cdots+d_t\leq k$. If it holds, we return the colorful solution with the composite clusters $L^1,\ldots,L^t$ whose means are $\mathbf{s}^1,\ldots,\mathbf{s}^t$ respectively and the remaining clusters are simple. Otherwise, we discard the choice of $\mathcal{P}$. If for one of the choices of $\mathcal{P}$ we find a colorful solution, we return it and stop. If we fail to find a solution for all possible $\mathcal{P}$, we return the answer NO and stop. If the described algorithm produces a solution, then because simple clusters do not contribute to the total cost of the solution it is possible to verify that the produced solution is a colorful solution to $(\mathbf{A},r,k)$.

So everything boils down to finding the optimal value d_i together with the corresponding initial clusters and their means. Let us remind that a regular partition $\{I_1,\ldots,I_p\}$ of $\{1,\ldots,n\}$ is a partition where any two indices corresponding to equal columns of $\mathbf{A}$ are assigned to the same cluster I_i. We call this the auxiliary problem Cluster Selection.

Cluster Selection

Input: An $m \times n$-matrix $\mathbf{A}$ with columns $\mathbf{a}^1, \ldots, \mathbf{a}^n$, a regular partition $\{I_1, \ldots, I_p\}$ of $\{1, \ldots, n\}$, and a nonnegative integer d.

Question: Is there a set of initial clusters $J_1, \ldots, J_p$ and a vector $\mathbf{c} \in \{0,1\}^m$ such that $J_i \subseteq I_i$ for $i \in \{1, \ldots, p\}$ and $\sum_{i=1}^{p} \sum_{j \in J_i} d_H(\mathbf{c}, \mathbf{a}^j) \leq d$?

Thus in Cluster Selection, for each cluster I_i, we have to select exactly one initial cluster J_i contained in I_i such that the total Hamming distance of the columns of the selected clusters to their means does not exceed d.

We prove that Cluster Selection is FPT when parameterized by d. The proof of this result is based on a reduction to the problem about strings. More precisely, we apply the result of Marx [45] about the Consensus Patterns problem. Recall that for two strings a and b of the same length, the *Hamming distance* $d_H(a, b)$ between strings is defined as the number of position where the strings differ.

Consensus Patterns

Input: A set of p strings $\{s_1, \ldots, s_p\}$ over an alphabet Σ, a positive integer t and a nonnegative integer d.

Question: Is there a string s of length t over Σ, and a length t substring s'_i of s_i for every $i \in \{1, \ldots, p\}$ such that $\sum_{i=1}^{p} d_H(\mathbf{s}, s'_i) \leq d$?

Marx proved in [45] that Consensus Patterns can be solved in time $\delta^{\mathcal{O}(\delta)} \cdot |\Sigma|^{\delta} \cdot L^9$ where $\delta = d/p$ and L the input size, i.e., the total length of all the strings in the input. This implies that Consensus Patterns can be solved in time $2^{\mathcal{O}(d \log d)} \cdot L^9$ if $|\Sigma|$ is fixed. We construct an FPT Turing reduction from Cluster Selection to Consensus Patterns. The reduction is technical but the rough idea is the following. We guess the number of elements in every cluster of the solution $J_1, \ldots, J_p$. For each guess, $(\ell_1 = |J_1|, \ldots, \ell_p = |J_p|)$, we delete from I_i all initial clusters which size is not equal to ℓ_i. Then for each I_i, we make ℓ_i equal strings. Each of theses strings consists of substrings corresponding to distinct columns of $\mathbf{A}$ with indices from I_i which are separated by special splitting substrings constructed by making use of two additional symbols. Thus in total we use alphabet Σ with 4 letters. This way, the choice of substrings corresponds to the choice of initial clusters.

Summarizing, our algorithm for Binary r-Means consists of two FPT Turing reductions. First, we design a reduction to Cluster Selection that, in its turn, is reduced to Consensus Patterns. This gives our first main result.

▶ **Theorem 1.** Binary r-Means *is solvable in time* $2^{\mathcal{O}(k \log k)} \cdot (nm)^{\mathcal{O}(1)}$.

3 Subexponential algorithms

The main result of this section is an FPT algorithm for Low Boolean-Rank Approximation parameterized by k and r that is subexponential in k. Note that Low Boolean-Rank Approximation is more complicated than Low GF(2)-Rank Approximation. The main reason to that is that the elements of the matrices do not form a field and thus many nice properties of matrix-rank cannot be used here. The way we handle this issue is to solve the $\mathbf{P}$-Matrix Approximation problem.

Let $\mathbf{A}$ be a Boolean $m \times n$-matrix with the Boolean rank $r \geq 1$. Then $\mathbf{A} = \mathbf{A}^{(1)} \vee \ldots \vee \mathbf{A}^{(r)}$ where $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(r)}$ are matrices of Boolean rank 1. It implies that $\mathbf{A}$ has at most 2^r pairwise-distinct rows and at most 2^r pairwise-distinct columns. Hence, the Boolean rank of $\mathbf{A}$ is at most r if and only if there is a $2^r \times 2^r$-matrix $\mathbf{P}$ of Boolean rank at most r such that

$\mathbf{A}$ is a $\mathbf{P}$-matrix. Respectively, the Low Boolean-Rank Approximation problem can be reformulated as follows: Decide whether there is a $2^r \times 2^r$-pattern matrix $\mathbf{P}$ with the Boolean rank at most r and an $m \times n$ $\mathbf{P}$-matrix $\mathbf{B}$ such that $\|\mathbf{A} - \mathbf{B}\|_F^2 \leq k$. We generate all $2^r \times 2^r$-matrices $\mathbf{P}$ of Boolean rank at most r, and then for each matrix $\mathbf{P}$, we solve $\mathbf{P}$-Matrix Approximation for the instance $(\mathbf{A}, \mathbf{P}, k)$. We return YES if we obtain at least one yes-instance of $\mathbf{P}$-Matrix Approximation, and we return NO otherwise. Thus, using the algorithm for $\mathbf{P}$-Matrix Approximation from Theorem 5 (See later in this section), we obtain the following theorem.

▶ **Theorem 2.** Low Boolean-Rank Approximation *is solvable in* $2^{\mathcal{O}(r2^r\sqrt{k \log k})} \cdot (nm)^{\mathcal{O}(1)}$ *time.*

Now we sketch the main ideas behind our algorithm for $\mathbf{P}$-Matrix Approximation. In fact, we exploit the same ideas to construct subexponential in k time algorithms for Binary r-Means and Low GF(2)-Rank Approximation, and these algorithms are great deal less technical. Hence, we concentrate here on these problems.

Let $(\mathbf{A}, r, k)$ be an instance of Binary r-Means where $\mathbf{A}$ is a matrix with columns $(\mathbf{a}^1, \ldots, \mathbf{a}^n)$. Recall that given a solution $\{I_1, \ldots, I_{r'}\}$, one can compute the corresponding means $\mathbf{c}^1, \ldots, \mathbf{c}^{r'} \in \{0,1\}^m$ using the majority rule. Note that in the opposite direction, given a set of means $\mathbf{c}^1, \ldots, \mathbf{c}^{r'}$, we can construct clusters $\{I_1, \ldots, I_{r'}\}$ as follows: for each column $\mathbf{a}^j$, find the closest $\mathbf{c}^i$, $i \in \{1, \ldots, r'\}$, that is such that $d_H(\mathbf{c}^i, \mathbf{a}^j)$ is minimum and assign j to I_i. Note that this procedure does not guarantee that all clusters are nonempty but we can simply delete empty clusters. Hence, we can define a solution as a set of means $C = \{\mathbf{c}^1, \ldots, \mathbf{c}^{r'}\}$.

It can be observed that we can restrict ourself by considering only solutions of special type. Let $\mathbf{a}_1, \ldots, \mathbf{a}_m$ be the rows of $\mathbf{A}$. We say that a vector $\mathbf{c} = (c_1, \ldots, c_m)^\intercal \in \{0,1\}^m$ *agrees* with $\mathbf{A}$ if $c_i = c_j$ whenever $\mathbf{a}_i = \mathbf{a}_j$ for $i, j \in \{1, \ldots, m\}$. If $(\mathbf{A}, r, k)$ is a yes-instance of Binary r-Means, then it can be shown that $(\mathbf{A}, r, k)$ has a solution such that for each cluster of the solution its mean agrees with $\mathbf{A}$. Also it could be seen that if $(\mathbf{A}, r, k)$ is a yes-instance of Binary r-Means, then $\mathbf{A}$ has at most $r + k$ pairwise-distinct columns and at most $2^r + k$ pairwise-distinct rows. These observations allow us to construct a recursive branching algorithm for Binary r-Means.

First, we preprocess the instance $(\mathbf{A}, r, k)$: if $\mathbf{A}$ has at least $r + k + 1$ pairwise-distinct columns or at least $2^r + k + 1$ distinct rows, we return the answer NO and stop. Now on we assume that this is not the case.

Assume that we are given a partial clustering of some columns of $\mathbf{A}$ represented by a family of means $\{\mathbf{c}_1, \ldots, \mathbf{c}_s\}$, a budget d and the set I of remaining columns of $\mathbf{A}$ which have to be clustered. The algorithm tries to extend the partial solution by not exceeding the budget $d \leq k$. Some of the columns from I can go to the existing cluster and some can form new clusters. Suppose that we know the minimum Hamming distance $h \leq d$ from vectors in new clusters to their means (in the algorithm we consider all possible values of h). Then all vectors which are within distance less than h to the already existing means, can be assigned to the existing clusters. Then we will be basically left with two options. Either the number of columns to be assigned to new clusters does not exceed $\sqrt{d \log(2^r + d)} \leq \sqrt{k \log(2^r + k)}$; in this case we brute-force in all possible partitions of I. Or we can upper bound $h \leq \sqrt{d/\log(2^r + d)} \leq \sqrt{k/\log(2^r + k)}$. In the latter case we branch on all possible vectors that agree with $\mathbf{A}$ and are at distance at most h from one of the at most $r + k$ columns of I. Due to the fact that the number of distinct rows of $\mathbf{A}$ is at most $2^r + k$, the number of branches for each column of I is at most $(2^r + k)^h$. In each

branch of our algorithm, we add a new mean to the partial solution and call our algorithm recursively. Note that the depth of the recursion is upper bounded by r.

This way we obtain the following theorem.

▶ **Theorem 3.** Binary r-Means *is solvable in time* $2^{\mathcal{O}(r\sqrt{k\log(2^r+k)})}\cdot(nm)^{\mathcal{O}(1)}$.

The general idea of the subexponential time algorithm for Low GF(2)-Rank Approximation is similar but the algorithm is more complicated. Let $(\mathbf{A},r,k)$ be an instance of Low GF(2)-Rank Approximation where $\mathbf{a}^1,\ldots,\mathbf{a}^n$ are the columns of $\mathbf{A}$. We observe that $(\mathbf{A},r,k)$ is a yes-instance if and only if there is a positive integer $r'\leq r$ and linearly independent vectors $\mathbf{c}^1,\ldots,\mathbf{c}^{r'}\in\{0,1\}^m$ over $\mathsf{GF}(2)$ such that

$$\sum_{i=1}^{n}\min\{d_H(\mathbf{s},\mathbf{a}^i)\mid \mathbf{s}=\bigoplus_{j\in I}\mathbf{c}^j,\ I\subseteq\{1,\ldots,r'\}\}\leq k;$$

we use "$\bigoplus$" to denote the summation over $\mathsf{GF}(2)$. Respectively, we construct the recursive branching algorithm for Low GF(2)-Rank Approximation that tries to extend a partial solution represented by a family of linearly independent vectors. We use the properties that if $(\mathbf{A},r,k)$ is a yes-instance, then the number of pairwise distinct columns and rows is at most 2^r+k and we can select new vectors that agree with $\mathbf{A}$. We obtain the following theorem.

▶ **Theorem 4.** Low GF(2)-Rank Approximation *is solvable in time* $2^{\mathcal{O}(r\sqrt{k\log(2^r+k)})}\cdot(nm)^{\mathcal{O}(1)}$.

For $\mathbf{P}$-Matrix Approximation, we use the same approach based on the combination of branching and local search but because we have to follow the structure of the pattern matrix P, the algorithm becomes technical. We get the following running time for the problem.

▶ **Theorem 5.** $\mathbf{P}$-Matrix Approximation *is solvable in time* $2^{\mathcal{O}((p+q)\sqrt{k\log(p+k)+p\log p+q\log q+q\log(q+k)})}\cdot(nm)^{\mathcal{O}(1)}$.

Note that the running time in Theorem 5 is asymmetric in p and q due to the fact that we treat rows and columns in different way but, trivially, the instances $(\mathbf{A},\mathbf{P},k)$ and $(\mathbf{A}^\intercal,\mathbf{P}^\intercal,k)$ of $\mathbf{P}$-Matrix Approximation are equivalent.

4 Conclusion and open problems

In this paper we provide a number of parameterized algorithms for a number of binary matrix-approximation problems. Our results uncover some parts of the complexity landscape of these fascinating problems. We hope that our work will facilitate further investigation of this important and exciting area. We conclude with the following concrete open problems about bivariate complexity of Binary r-Means, Low GF(2)-Rank Approximation, and Low Boolean-Rank Approximation.

For Binary r-Means we have shown that the problem is solvable in time $2^{\mathcal{O}(k\log k)}\cdot(nm)^{\mathcal{O}(1)}$. A natural question is whether this running time is optimal. While the lower bound of the kind $2^{o(k)}\cdot(nm)^{\mathcal{O}(1)}$ or $2^{o(k\log k)}\cdot(nm)^{\mathcal{O}(1)}$ seems to be most plausible here, we do not know any strong argument against, say a $2^{o(k)}\cdot(nm)^{\mathcal{O}(1)}$-time algorithm. At least for the number of distinct columns $r\in o((k/\log k)^{1/3})$ we have a subexponential in k algorithm, so maybe we can solve the problem in time subexponential in k for any value of r?

For Low GF(2)-Rank Approximation we have an algorithm solving the problem in time $2^{\mathcal{O}(r^{3/2}\cdot\sqrt{k\log k})}(nm)^{\mathcal{O}(1)}$. Here, shaving off the $\sqrt{\log k}$ factor in the exponent seems to

be a reasonable thing. However, we do not know how to do it even at the cost of a worse dependence in r. In other words, could the problem be solvable in time $2^{\mathcal{O}(f(r)\cdot\sqrt{k})}(nm)^{\mathcal{O}(1)}$ for some function f? On the other hand, we also do not know how to rule out algorithms running in time $2^{o(r)\cdot o(k)}(nm)^{\mathcal{O}(1)}$.

For Low Boolean-Rank Approximation, how far is our upper bound $2^{\mathcal{O}(r2^r\cdot\sqrt{k\log k})}(nm)^{\mathcal{O}(1)}$ from the optimal? For example, we know that for any function f, the solvability of the problem in time $2^{2^{o(r)}}f(k)(nm)^{\mathcal{O}(1)}$ implies the failure of ETH. Could we rule out any $2^{o(\sqrt{k})}f(r)(nm)^{\mathcal{O}(1)}$ algorithm?

References

1. Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, 2004. doi:10.1145/1008731.1008736.
2. Alfred V. Aho, Jeffrey D. Ullman, and Mihalis Yannakakis. On notions of information transfer in VLSI circuits. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 133–139. ACM, 1983. doi:10.1145/800061.808742.
3. Noga Alon and Benny Sudakov. On two segmentation problems. *J. Algorithms*, 33(1):173–184, 1999. doi:10.1006/jagm.1999.1024.
4. Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.
5. Sanjeev Arora, Rong Ge, Ravindran Kannan, and Ankur Moitra. Computing a nonnegative matrix factorization - provably. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 145–162. ACM, 2012.
6. Mihai Badoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 250–257. ACM, 2002. doi:10.1145/509907.509947.
7. Eduard Bartl, Radim Belohlávek, and Jan Konecny. Optimal decompositions of matrices with grades into binary and graded matrices. *Annals of Mathematics and Artificial Intelligence*, 59(2):151–167, Jun 2010. doi:10.1007/s10472-010-9185-y.
8. Amitabh Basu, Michael Dinitz, and Xin Li. Computing approximate PSD factorizations. *CoRR*, abs/1602.07351, 2016. URL: http://arxiv.org/abs/1602.07351, arXiv:1602.07351.
9. Radim Belohlávek and Vilém Vychodil. Discovery of optimal factors in binary data via a novel method of matrix decomposition. *J. Computer and System Sciences*, 76(1):3–20, 2010. doi:10.1016/j.jcss.2009.05.002.
10. Karl Bringmann, Pavel Kolev, and David P. Woodruff. Approximation algorithms for ℓ_0-low rank approximation. In *Advances in Neural Information Processing Systems 30 (NIPS)*, pages 6651–6662, 2017. URL: http://papers.nips.cc/paper/7242-approximation-algorithms-for-ell_0-low-rank-approximation.
11. L. Sunil Chandran, Davis Issac, and Andreas Karrenbauer. On the parameterized complexity of biclique cover and partition. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 63 of *LIPIcs*, pages 11:1–11:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.11.
12. Andrzej Cichocki, Rafal Zdunek, Anh Huy Phan, and Shun-ichi Amari. *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation*. John Wiley & Sons, 2009.
13. Rudi Cilibrasi, Leo van Iersel, Steven Kelk, and John Tromp. The complexity of the single individual SNP haplotyping problem. *Algorithmica*, 49(1):13–36, 2007. doi:10.1007/s00453-007-0029-z.

14 Kenneth L. Clarkson and David P. Woodruff. Input sparsity and hardness for robust subspace approximation. In *Proceedings of the 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 310–329. IEEE Computer Society, 2015.

15 Joel E Cohen and Uriel G Rothblum. Nonnegative ranks, decompositions, and factorizations of nonnegative matrices. *Linear Algebra and its Applications*, 190:149–168, 1993.

16 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

17 Chen Dan, Kristoffer Arnsfelt Hansen, He Jiang, Liwei Wang, and Yuchen Zhou. On low rank approximation of binary matrices. *CoRR*, abs/1511.01699, 2015. arXiv:1511.01699.

18 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

19 Pål Grønås Drange, Felix Reidl, Fernando Sanchez Villaamil, and Somnath Sikdar. Fast biclustering by dual parameterization. *CoRR*, abs/1507.08158, 2015.

20 Uriel Feige. NP-hardness of hypercube 2-segmentation. *CoRR*, abs/1411.0821, 2014. arXiv:1411.0821.

21 Samuel Fiorini, Serge Massar, Sebastian Pokutta, Hans Raj Tiwary, and Ronald de Wolf. Exponential lower bounds for polytopes in combinatorial optimization. *J. ACM*, 62(2):17, 2015. doi:10.1145/2716307.

22 Fedor V. Fomin, Petr A. Golovach, and Fahad Panolan. Parameterized low-rank binary matrix approximation. *CoRR*, abs/1803.06102, 2018.

23 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of cluster editing with a small number of clusters. *J. Computer and System Sciences*, 80(7):1430–1447, 2014.

24 Fedor V. Fomin, Daniel Lokshtanov, Syed Mohammad Meesum, Saket Saurabh, and Meirav Zehavi. Matrix rigidity from the viewpoint of parameterized complexity. In *Proceedings of the 34th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 66 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:14, 2017. doi:10.4230/LIPIcs.STACS.2017.32.

25 Yun Fu. *Low-Rank and Sparse Modeling for Visual Analysis*. Springer International Publishing, 1 edition, 2014.

26 Nicolas Gillis and Stephen A. Vavasis. On the complexity of robust PCA and ℓ_1-norm low-rank matrix approximation. *CoRR*, abs/1509.09236, 2015. URL: http://arxiv.org/abs/1509.09236, arXiv:1509.09236.

27 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Data reduction and exact algorithms for clique cover. *ACM Journal of Experimental Algorithmics*, 13, 2008. doi:10.1145/1412228.1412236.

28 David A. Gregory, Norman J. Pullman, Kathryn F. Jones, and J. Richard Lundgren. Biclique coverings of regular bigraphs and minimum semiring ranks of regular matrices. *J. Combinatorial Theory Ser. B*, 51(1):73–89, 1991. doi:10.1016/0095-8956(91)90006-6.

29 Dmitry Grigoriev. Using the notions of separability and independence for proving the lower bounds on the circuit complexity (in russian). *Notes of the Leningrad branch of the Steklov Mathematical Institute, Nauka*, 1976.

30 Dmitry Grigoriev. Using the notions of separability and independence for proving the lower bounds on the circuit complexity. *Journal of Soviet Math.*, 14(5):1450–1456, 1980.

31 Harold W. Gutch, Peter Gruber, Arie Yeredor, and Fabian J. Theis. ICA over finite fields - separability and algorithms. *Signal Processing*, 92(8):1796–1808, 2012. doi:10.1016/j.sigpro.2011.10.003.

32 Alexander E. Guterman. Rank and determinant functions for matrices over semirings. In *Surveys in contemporary mathematics*, volume 347 of *London Math. Soc. Lecture Note Ser.*, pages 1–33. Cambridge Univ. Press, Cambridge, 2008.

33 Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering. In *Proceedings of the 10th annual symposium on Computational Geometry*, pages 332–339. ACM, 1994.

34 Peng Jiang and Michael T. Heath. Mining discrete patterns via binary matrix factorization. In *ICDM Workshops*, pages 1129–1136. IEEE Computer Society, 2013.

35 Peng Jiang, Jiming Peng, Michael Heath, and Rui Yang. *A Clustering Approach to Constrained Binary Matrix Factorization*, pages 281–303. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

36 Ravindran Kannan and Santosh Vempala. Spectral algorithms. *Foundations and Trends in Theoretical Computer Science*, 4(3-4):157–288, 2009. doi:10.1561/0400000025.

37 Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan. Segmentation problems. *J. ACM*, 51(2):263–280, 2004. doi:10.1145/972639.972644.

38 Mehmet Koyutürk and Ananth Grama. Proximus: A framework for analyzing very high dimensional discrete-attributed datasets. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 147–156, New York, NY, USA, 2003. ACM. doi:10.1145/956750.956770.

39 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2):5:1–5:32, 2010. doi:10.1145/1667053.1667054.

40 Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

41 Satyanarayana V. Lokam. Complexity lower bounds using linear algebra. *Found. Trends Theor. Comput. Sci.*, 4:1–155, 2009.

42 László Lovász and Michael E. Saks. Lattices, Möbius functions and communication complexity. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 81–90. IEEE, 1988.

43 Haibing Lu, Jaideep Vaidya, Vijayalakshmi Atluri, and Yuan Hong. Constraint-aware role mining via extended boolean matrix decomposition. *IEEE Trans. Dependable Sec. Comput.*, 9(5):655–669, 2012. doi:10.1109/TDSC.2012.21.

44 Michael W. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends in Machine Learning*, 3(2):123–224, 2011. doi:10.1561/2200000035.

45 Dániel Marx. Closest substring problems with small distances. *SIAM J. Comput.*, 38(4):1382–1410, 2008. doi:10.1137/060673898.

46 S. M. Meesum, Pranabendu Misra, and Saket Saurabh. Reducing rank of the adjacency matrix by graph modification. *Theoret. Comput. Sci.*, 654:70–79, 2016. doi:10.1016/j.tcs.2016.02.020.

47 Syed Mohammad Meesum and Saket Saurabh. Rank reduction of directed graphs by vertex and edge deletions. In *Proceedings of the 12th Latin American Symposium on (LATIN)*, volume 9644 of *Lecture Notes in Comput. Sci.*, pages 619–633. Springer, 2016. doi:10.1007/978-3-662-49529-2_46.

48 Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. *IEEE Trans. Knowl. Data Eng.*, 20(10):1348–1362, 2008. doi:10.1109/TKDE.2008.53.

49 Pauli Miettinen and Jilles Vreeken. Model order selection for boolean matrix factorization. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 51–59. ACM, 2011. doi:10.1145/2020408.2020424.

50 Ankur Moitra. An almost optimal algorithm for computing nonnegative rank. *SIAM J. Comput.*, 45(1):156–173, 2016. doi:10.1137/140990139.

51 Ganesh R Naik. *Non-negative Matrix Factorization Techniques*. Springer, 2016.

52 James Orlin. Contentment in graph theory: covering graphs with cliques. *Nederl. Akad. Wetensch. Proc. Ser. A* **80**=*Indag. Math.*, 39(5):406–424, 1977.

53 Rafail Ostrovsky and Yuval Rabani. Polynomial-time approximation schemes for geometric min-sum median clustering. *J. ACM*, 49(2):139–156, 2002. doi:10.1145/506147.506149.

54 Amichai Painsky, Saharon Rosset, and Meir Feder. Generalized independent component analysis over finite alphabets. *IEEE Trans. Information Theory*, 62(2):1038–1053, 2016. doi:10.1109/TIT.2015.2510657.

55 A A Razborov. On rigid matrices. *Manuscript in russian*, 1989.

56 Ilya P. Razenshteyn, Zhao Song, and David P. Woodruff. Weighted low rank approximations with provable guarantees. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 250–263. ACM, 2016. doi:10.1145/2897518.2897639.

57 Bao-Hong Shen, Shuiwang Ji, and Jieping Ye. Mining discrete patterns via binary matrix factorization. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 757–766, New York, NY, USA, 2009. ACM. doi:10.1145/1557019.1557103.

58 Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In *Mathematical Foundations of Computer Science (MFCS)*, volume 53 of *Lecture Notes in Comput. Sci.*, pages 162–176. Springer, 1977.

59 David P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1–2):1–157, 2014. doi:10.1561/0400000060.

60 Sharon Wulff, Ruth Urner, and Shai Ben-David. Monochromatic bi-clustering. In *Proceedings of the 30th International Conference on Machine Learning, (ICML)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 145–153. JMLR.org, 2013. URL: http://jmlr.org/proceedings/papers/v28/.

61 Mihalis Yannakakis. Expressing combinatorial optimization problems by linear programs. *J. Comput. Syst. Sci.*, 43(3):441–466, 1991. doi:10.1016/0022-0000(91)90024-Y.

62 Arie Yeredor. Independent component analysis over Galois fields of prime order. *IEEE Trans. Information Theory*, 57(8):5342–5359, 2011. doi:10.1109/TIT.2011.2145090.

Towards Blackbox Identity Testing of Log-Variate Circuits

Michael A. Forbes
University of Illinois at Urbana-Champaign, USA
miforbes@illinois.edu

Sumanta Ghosh
Department of Computer Science, IIT Kanpur, India
sumghosh@cse.iitk.ac.in

Nitin Saxena
Department of Computer Science, IIT Kanpur, India
nitin@cse.iitk.ac.in

Abstract

Derandomization of blackbox identity testing reduces to extremely special circuit models. After a line of work, it is known that focusing on circuits with constant-depth and constantly many variables is enough (Agrawal,Ghosh,Saxena, STOC'18) to get to general hitting-sets and circuit lower bounds. This inspires us to study circuits with few variables, eg. logarithmic in the size s.

We give the first poly(s)-time blackbox identity test for $n = O(\log s)$ variate size-s circuits that have poly(s)-dimensional partial derivative space; eg. depth-3 diagonal circuits (or $\Sigma \wedge \Sigma^n$). The former model is well-studied (Nisan,Wigderson, FOCS'95) but no poly($s2^n$)-time identity test was known before us. We introduce the concept of *cone-closed* basis isolation and prove its usefulness in studying log-variate circuits. It subsumes the previous notions of rank-concentration studied extensively in the context of ROABP models.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory, Theory of computation → Fixed parameter tractability, Theory of computation → Pseudorandomness and derandomization, Computing methodologies → Algebraic algorithms, Mathematics of computing → Combinatoric problems

Keywords and phrases hitting-set, depth-3, diagonal, derandomization, polynomial identity testing, log-variate, concentration, cone closed, basis isolation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.54

Related Version A full version of the paper is available at `https://www.cse.iitk.ac.in/users/nitin/papers/log-var-hsg.pdf`.

Acknowledgements M.F. & N.S. are grateful to the organizers of algebraic complexity workshops in 2014 (MPI Saarbrücken & TIFR Mumbai) that initiated the early discussions. N.S. thanks Manindra Agrawal for useful discussions. N.S. thanks the funding support from DST (DST/SJF/MSA-01/2013-14).

1 Introduction

Polynomial Identity Testing (PIT) problem is to decide whether a multivariate polynomial is *zero*, where the input polynomial is given as an *algebraic circuit*. Algebraic circuits are the algebraic analog of boolean circuits that use ring operations $\{+, \times\}$ and computes polynomials (say) over a field. Since a polynomial computed by a circuit can have exponentially many

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 54; pp. 54:1–54:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

monomials wrt the circuit size, one cannot solve PIT in polynomial time by explicitly expanding the polynomial. On the other hand, using circuits we can efficiently evaluate polynomials at any point. This helps us to get a polynomial time randomized algorithm for PIT by evaluating the circuit at a random point, since any non-zero polynomial evaluated at a random point outputs a non-zero value with high probability [10, 58, 54]. However, finding a deterministic polynomial time algorithm for PIT is a longstanding open question in algebraic complexity theory. The PIT problem has been studied in two different paradigms: **1)** *whitebox* – allowed to see the internal structure of the circuit, and **2)** *blackbox* – can only use the circuit as an oracle to evaluate at points (from a small field extension). It has deep connections with both circuit lower bounds [29, 31, 1, 2] and many other algorithmic problems [41, 4, 35, 11, 13]. For more details on PIT, see the surveys [51, 52, 55] or review articles [56, 42].

Despite a lot of effort, little progress has been made on the PIT problem in general. However, efficient (deterministic poly-time) PIT algorithms are known for many special circuit models. For example, blackbox PIT for depth-2 circuits (or sparse polynomials) [8, 34, 39], PIT algorithms for subclasses of depth-3 circuits [33, 50, 53], subclasses of depth-4 circuits [5, 7, 46, 15, 36, 37, 45], read-once algebraic branching programs (ROABP) and related models [19, 6, 18, 3, 26, 25], certain types of symbolic determinants [12, 27], as well as non-commutative models [38, 22].

1.1 Our results

In the first result, *we give a polynomial time blackbox PIT algorithm of log-variate depth-*3 *diagonal circuits* $\Sigma \wedge \Sigma$ (i.e. number of variables is logarithmic wrt circuit size). Depth-3 diagonal circuits compute a sum of power of linear polynomials. This model was first introduced by [51] and has since drawn significant attention of PIT research community. Saxena [51] first gave a polynomial time whitebox algorithm and exponential lower bound for this model, by introducing a duality trick. In a subsequent work Kayal [32] gave an alternate polynomial time whitebox algorithm for depth-3 diagonal circuits based on the partial derivative method, which was first introduced by [44] to prove circuit lower bounds; as, $\Sigma \wedge \Sigma$ circuits have a low-dimension partial derivative space. However, one limitation of these approaches was that they depend on the characteristic of the underlying field. Later, [16] gave an alternative proof of duality trick which depends only on the field size (as mentioned in [24, Lem.4.7]) and Saptharishi [48, Chap.3] extended Kayal's idea for large enough field.

Although this model is very weak (it cannot even compute $x_1 \cdots x_n$ efficiently), studying this model has proved quite fruitful. Duality trick was crucially used in the work by [23], where they showed that depth-3 circuits, in some sense, capture the complexity of general arithmetic circuits.

Like whitebox PIT, a series of work has been done on *blackbox* PIT for depth-3 diagonal circuits. Both [6] and [19] gave two independent and different quasi-polynomial time blackbox PIT algorithms for this model. Later, [18] gave an $s^{O(\log\log s)}$-time (s is the circuit size) blackbox PIT algorithm for this model. Mulmuley [43, 40] related depth-3 diagonal blackbox PIT to construction of normalization maps for the invariants of the group SL_m for constant m. We can not give the detailed notation here and would like to refer to [40, Sec.9.3]. Despite a lot of effort, no polynomial time blackbox PIT for this model is known. After depth-2 circuits (or sparse polynomials), this can be thought of as the simplest model for which no polynomial time blackbox PIT is known. Because of its simplicity, this model is a good test case for generating new ideas for the PIT problem.

Log-variate models: Now we discuss why studying PIT for log-variate models is so important. The PIT algorithms in current literature always try to achieve a sub-exponential dependence on n, the number of variables. In a recent development, [2] showed that for some constant c a poly(s)-time blackbox PIT for size-s degree-s and $\log^{\circ c} s$-variate[1] circuits is sufficient to *completely* solve PIT. Most surprisingly, they also showed that a poly(s)-time blackbox PIT for size-s and $\log^{\star} s$-variate[2] $\Sigma \wedge \Sigma\Pi$ circuits will 'partially' solve PIT (in quasi-polynomial time) and prove that "either E$\not\subseteq$#P/poly or VP$\neq$VNP" (a weaker version of [2, Thm.21]). For example, even a poly(s)-time blackbox PIT for size-s and $(\log\log s)$-variate depth-4 circuits would be tremendous progress. A similar result also holds for $\Sigma \wedge^{a} \Sigma\Pi(n)$ circuits, where both a and n are 'arbitrarily small' unbounded functions (i.e. time-complexity may be arbitrary in terms of both a and n), see [2, Thm.21].

The above discussion motivates us to discover techniques and measures that are specialized to this low-variate regime. Many previous works are based on 'support size of a monomial' as a measure for rank-concentration [6, 18, 26]. For a monomial m, its *support* is the set of variables whose exponents are positive. We investigate a 'larger' measure: *cone-size* (see Definition 3) which is the number of monomials that divide m (also see [14]). Using cone-size as a measure for rank-concentration, we give a blackbox PIT algorithm for circuit models with 'low' dimensional partial derivative space.

▶ **Theorem 1.** *Let $\mathbb{F}$ be a field of characteristic 0 or greater than d. Let $\mathcal{P}$ be a set of n-variate d-degree polynomials, over $\mathbb{F}$, computed by circuits of bitsize s such that: $\forall P \in \mathcal{P}$, the dimension of the partial derivative space of P is at most k. Then, blackbox PIT for $\mathcal{P}$ can be solved in $(sdk)^{O(1)} \cdot (3n/\log k)^{O(\log k)}$ time.*

Note that for $n = O(\log k) = O(\log sd)$, the above bound is poly-time and we get a polynomial time blackbox PIT algorithm for log-variate circuits (i.e. number of variables is logarithmic wrt circuit size) with low-dimensional partial derivative space. This was not known before our work. Prior to our work, [18] gave a $(sdk)^{O(\log\log sdk)}$-time algorithm for $\mathcal{P}$, using support size as the measure in the proof. Unlike our algorithm, in the log-variate case their algorithm remains super-polynomial time.

In particular, diagonal depth-3 circuit is a prominent model with low partial derivative space. So, our method gives a polynomial time PIT algorithm for log-variate depth-3 diagonal circuits. No poly-time blackbox PIT for this model was known before our work; again, $s^{O(\log\log s)}$ was the prior best [18].

Structure of log-variate polynomials? In the second result, we investigate a structural property of polynomials over vector spaces. For a polynomial $f(\mathbf{x})$ with coefficients over $\mathbb{F}^k$, let sp(f) be the subspace spanned by its coefficients. Informally, in *rank concentration* we try to concentrate the rank of sp(f) to the coefficients of "few" monomials. It was first introduced by [6]. Many works in PIT achieve rank concentration on low-support monomials, mainly, in the ROABP model [6, 18, 26, 25]. One way of strengthening low-support concentration is through *low-cone concentration*, where rank is concentrated in the low cone-size monomials. This concept was not used before in designing PIT algorithms. Our first result (Theorem 1) can be seen from this point of view. There, we developed a method to get polynomial time blackbox PIT for log-variate models which satisfy 'low-cone concentration property'.

[1] The function $\log^{\circ c}$ denotes c times composition of the log function. For e.g. $\log^{\circ 2} s = \log\log s$.
[2] For any positive integer s, $\log^{\star} s = \min\{i \mid \log^{\circ i} s \leq 1\}$.

We introduce the concept of cone-closed basis, a much stronger notion of concentration than the previous ones. We say f has a cone-closed basis, if there is a set of monomials B whose coefficients form a basis of $\mathrm{sp}(f)$ and B is closed under sub-monomials. This definition is motivated by a special depth-3 diagonal model, which have this property naturally (see Lemma 18). We prove that this notion is a strengthening of both low-support and low-cone concentration ideas (see Lemma 11). Recently, and independently, this notion of closure has also appeared as an 'abstract simplicial complex' in [21].

In the following result, we relate cone-closed basis with 'basis isolating weight assignment' (Defn.12)– another well studied concept in PIT. It was first introduced by [3] and also used in many other subsequent works [26, 12, 28]. Here, we show that a general polynomial f over $\mathbb{F}^k$, when shifted by a basis isolating weight assignment [3], becomes cone-closed. It strengthens some previously proven properties; eg., a polynomial over $\mathbb{F}^k$ when shifted 'randomly' becomes low-support concentrated [17, Cor.3.22] (extended version of [18]) or, when shifted by a basis isolating weight assignment becomes low-support concentrated [26, Lem.5.2].

Notations. For any $n \in \mathbb{N}$, $[n]$ denotes the set of first n positive integers. By $\mathbf{x}$, we denote $(x_1, \ldots, x_n)$, a tuple of n-variables. For any $\mathbf{e} = (e_1, \ldots, e_n) \in \mathbb{N}^n$, $\mathbf{x}^{\mathbf{e}}$ denotes the monomial $\prod_{i=1}^{n} x_i^{e_i}$. For a polynomial f and a monomial m, $\mathrm{coef}_m(f)$ denotes the coefficient of the monomial m in f. An *weight assignment* $\mathbf{w}$ on the variables $\mathbf{x}$ is an n-tuple $(w_1, \ldots, w_n) \in \mathbb{N}^n$, where w_i is the weight assigned to the variable x_i.

▶ **Theorem 2.** *Let $f(\mathbf{x}) \in \mathbb{F}[\mathbf{x}]^k$ be an n-variate d-degree polynomial over $\mathbb{F}^k$ and char $\mathbb{F} = 0$ or $> d$. Let $\mathbf{w} = (w_1, \ldots, w_n) \in \mathbb{N}^n$ be a basis isolating weight assignment of $f(\mathbf{x})$. Then, $f(\mathbf{x} + t^{\mathbf{w}}) := f(x_1 + t^{w_1}, \ldots, x_n + t^{w_n})$ has a cone-closed basis over $\mathbb{F}(t)$.*

1.2 Proof ideas

Proof idea of Theorem 1: The proof of Theorem 1 has two steps. In the first step, we show that with respect to any monomial ordering (say lexicographic monomial ordering), the dimension k of the partial derivative space of a polynomial is lower bounded by the cone-size of its leading monomial. For a polynomial $f \in \mathbb{F}[\mathbf{x}]$, the leading monomial, wrt a monomial ordering, is the largest monomial in the set $\{\mathbf{x}^{\mathbf{e}} \mid \mathrm{coef}_{\mathbf{x}^{\mathbf{e}}}(f) \neq 0\}$. So, for every nonzero $P \in \mathcal{P}$ there is a monomial with nonzero coefficient and cone-size $\leq k$. The second step is to check whether the coefficients of all the monomials in P, with cone-size $\leq k$, are zero. We show that the number of such monomials is small (Lemma 5); the number is quasi-polynomial in general, but, merely polynomial in the log-variate case. Next, we give a new method to efficiently extract a monomial of cone-size$\leq k$, out of a potentially exponential space of monomials (Lemma 4). These facts, combined with the estimates stated in Theorem 1, prove Corollary 6; which gives a polynomial time blackbox PIT algorithm for log-variate circuits with low dimensional partial derivative space.

Next, we discuss the idea to get a polynomial time blackbox PIT algorithm for depth-3 diagonal circuits where *rank* of the linear polynomials is logarithmic wrt the circuit size (see Definition 7 & Theorem 9). Here, the proof has two steps. First, in Lemma 8, we show how to efficiently reduce a low-rank depth-3 diagonal circuit to a low-*variate* depth-3 diagonal circuit while preserving nonzeroness. This we do by a Vandermonde based linear map on the variables. Since a depth-3 diagonal circuit has low-dimensional partial derivative space (i.e. polynomial wrt circuit size), we apply Corollary 6 on the low-variate depth-3 diagonal circuits and get Theorem 9.

Proof idea of Theorem 2: First, wrt the weight assignment $\mathbf{w}$, we define an ordering among the set of bases (see Section 3). Then, we show that wrt the basis isolating weight assignment $\mathbf{w}$, there exists a *unique minimum basis* and its weight is strictly less than the weight of every other basis (Lemma 13). Let B be the set of monomials whose coefficients form the least basis, wrt $\mathbf{w}$, of f.

Now, we consider the set of all sub-monomials of those in B and identify a subset A that is cone-closed. We define A in an algorithmic way (see Algorithm 1). Besides the cone-closed property, A also satisfies an algebraic property (Lemma 17)— In the *transfer matrix* T, that captures the variable-shift transformation (Equation 3), the sub-matrix $T_{A,B}$ is *full* rank. We prove that A is exactly a basis of the shifted f by studying the action of the shift on the coefficient vectors. The properties proved above and Cauchy-Binet Formula [57] are crucially used in the study of the coefficient vectors after the variable-shift.

Theorem 2 has an immediate consequence that any polynomial f over $\mathbb{F}^k$, when shifted by formal (or random) variables, becomes cone-closed; since the weight induced by the formal variables on the monomials is a basis isolating weight assignment. This seems quite a nontrivial and an interesting property of general polynomials (over vector spaces).

2 Low-cone concentration and hitting-sets– Proof of Theorem 1

In this section we initiate a study of properties that are relevant for low-variate circuits (or the log-variate regime).

Notations. For a circuit C, $|C|$ denotes the size of C. For a monomial m, by $\text{coef}_m(C)$, we denote the coefficient of monomial m in the polynomial computed by C. For a circuit C, we also use C to denote the polynomial computed by C.

▶ **Definition 3** (Cone of a monomial)**.** A monomial $\mathbf{x}^{\mathbf{e}}$ is called a *sub-monomial* of $\mathbf{x}^{\mathbf{f}}$, if $\mathbf{e} \leq \mathbf{f}$ (i.e. coordinate-wise). We say that $\mathbf{x}^{\mathbf{e}}$ is a *proper sub-monomial* of $\mathbf{x}^{\mathbf{f}}$, if $\mathbf{e} \leq \mathbf{f}$ and $\mathbf{e} \neq \mathbf{f}$.

For a monomial $\mathbf{x}^{\mathbf{e}}$, the *cone of* $\mathbf{x}^{\mathbf{e}}$ is the set of all sub-monomials of $\mathbf{x}^{\mathbf{e}}$. The cardinality of this set is called *cone-size of* $\mathbf{x}^{\mathbf{e}}$. It equals $\prod(\mathbf{e}+\mathbf{1}) := \prod_{i\in[n]}(e_i+1)$, where $\mathbf{e} = (e_1, \ldots, e_n)$.

A set S of monomials is called *cone-closed* if for every monomial in S all its sub-monomials are also in S.

▶ **Lemma 4** (Coef. extraction)**.** *Let C be a blackbox circuit which computes an n-variate and degree-d polynomial over a field of size greater than d. Then for any monomial $m = \prod_{i\in[n]} x_i^{e_i}$, we have a $\text{poly}(|C|d, \text{cs}(m))$-time algorithm to compute the coefficient of m in C, where $\text{cs}(m)$ denotes the cone-size of m.*

Proof. Our proof is in two steps. First, we inductively build a circuit computing a polynomial which has two parts; one is $\text{coef}_m(C) \cdot m$ and the other one is a "junk" polynomial where every monomial is a proper super-monomial of m. Second, we construct a circuit which extracts the coefficient of m. In both these steps the key is a classic interpolation trick.

We induct on the variables. For each $i \in [n]$, let $m_{[i]}$ denote $\prod_{j\in[i]} x_j^{e_j}$. We will construct a circuit $C^{(i)}$ which computes a polynomial of the form,

$$C^{(i)}(\mathbf{x}) \;=\; \text{coef}_{m_{[i]}}(C) \cdot m_{[i]} \;+\; C^{(i)}_{\text{junk}} \tag{1}$$

where, for every monomial m' in the support of $C^{(i)}_{\text{junk}}$, $m_{[i]}$ is a proper submonomial of $m'_{[i]}$.

Base case: Since $C =: C^{(0)}$ computes an n-variate degree-d polynomial, $C(\mathbf{x})$ can be written as $C(\mathbf{x}) = \sum_{j=0}^{d} c_j x_1^j$ where, $c_j \in \mathbb{F}[x_2, \ldots, x_n]$. Let $\alpha_0, \ldots, \alpha_{e_1}$ be some $e_1 + 1$ distinct elements in $\mathbb{F}$. For every α_j, let $C_{\alpha_j x_1}$ denote the circuit $C(\alpha_j x_1, x_2, \ldots, x_n)$ which computes $c_0 + c_1\alpha_j x_1 + \ldots + c_{e_1}\alpha_j^{e_1} x_1^{e_1} + \cdots + c_d \alpha_j^d x_1^d$. Since

$$M = \begin{bmatrix} 1 & \alpha_0 & \ldots & \alpha_0^{e_1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha_{e_1} & \ldots & \alpha_{e_1}^{e_1} \end{bmatrix}$$

is an invertible Vandermonde matrix, one can find an $\mathbf{a} = [a_0, \ldots, a_{e_1}] \in \mathbb{F}^{e_1+1}$, $\mathbf{a} \cdot M = [\,0,\ 0,\ \ldots,\ 1\,]$. Using this $\mathbf{a}$, we get the circuit $C^{(1)} := \sum_{j=0}^{e_1} a_j C^{(0)}_{\alpha_j x_1}$. Its least monomial wrt x_1 has $\deg_{x_1} \geq e_1$, which is the property that we wanted.

Induction step $(i \to i+1)$*:* From induction hypothesis, we have the circuit $C^{(i)}$ with the properties mentioned in Eqn.1. The polynomial can also be written as $b_0 + b_1 x_{i+1} + \ldots + b_{e_{i+1}} x_{i+1}^{e_{i+1}} + \ldots b_d x_{i+1}^d$, where every b_j is in $\mathbb{F}[x_1, \ldots, x_i, x_{i+2}, \ldots, x_n]$. Like the proof of the base case, for $e_{i+1}+1$ distinct elements $\alpha_0, \ldots, \alpha_{e_{i+1}} \in \mathbb{F}$, we get $C^{(i+1)} = \sum_{j=0}^{e_{i+1}} a_j C^{(i)}_{\alpha_j x_{i+1}}$, for some $\mathbf{a} = [a_0, \ldots, a_{e_{i+1}}] \in \mathbb{F}^{e_{i+1}+1}$ and the structural constraint of $C^{(i+1)}$ is easy to verify, completing the induction.

Now we describe the second step of the proof. After first step, we get

$$C^{(n)}(\mathbf{x}) \;=\; \mathrm{coef}_m(C) \cdot m \;+\; C^{(n)}_{\mathrm{junk}}\,,$$

where for every monomial m' in the support of $C^{(n)}_{\mathrm{junk}}$, m is a proper submonomial of m'. Consider the polynomial $C^{(n)}(x_1 t, \ldots, x_n t)$ for a fresh variable t. Then, using interpolation wrt t we can construct a $O(|C^{(n)}| \cdot d)$-size circuit for $\mathrm{coef}_m(C) \cdot m$, by extracting the coefficient of $t^{\deg(m)}$, since the degree of every monomial appearing in $C^{(n)}_{\mathrm{junk}}$ is $> \deg(m)$. Now evaluating at $\mathbf{1}$, we get $\mathrm{coef}_m(C)$. The size, or time, constraint of the final circuit clearly depends polynomially on $|C|, d$ and $\mathrm{cs}(m)$. ◀

But, how many low-cone monomials can there be? Fortunately, in the log-variate regime they are not too many [47]. Though, in general, they are quasi-polynomially many.

▶ **Lemma 5** (Counting low-cones)**.** *The number of n-variate monomials with cone-size at most k is $O(rk^2)$, where $r := (3n/\log k)^{\log k}$.*

Proof. First, we prove that for any fixed support set, the number of cone-size $\leq k$ monomials is less than k^2. Next, we multiply by the number of possible support sets to get the estimate.

Let $T(k, \ell)$ denote the number of cone-size$\leq k$ monomials m with support set, say, exactly $\{x_1, \ldots, x_\ell\}$. Since the exponent of x_ℓ in such an m is at least 1 and at most $k-1$, we have the following by the disjoint-sum rule: $T(k, \ell) \leq \sum_{i=2}^{k} T\,(k/i, \ell - 1)$. This recurrence affords an easy inductive proof as, $T(k, \ell) < \sum_{i=2}^{k} (k/i)^2 < k^2 \cdot \sum_{i=2}^{k} \left(\frac{1}{i-1} - \frac{1}{i}\right) < k^2$.

From the definition of cone, a cone-size $\leq k$ monomial can have support size at most $\ell := \lfloor \log k \rfloor$. The number of possible support sets, thus, is $\sum_{i=0}^{\ell} \binom{n}{i}$. Using the binomial estimates [30, Chapter 1], we get $\sum_{i=0}^{\ell} \binom{n}{i} \leq (3n/\ell)^{\ell}$. ◀

The partial derivative space of polynomials was first used by Nisan and Wigderson [44] to prove circuit lower bounds. Later, it was used in many other works. For more details see the following surveys [9, 49]. Here, using cone-size as a measure, we describe a blackbox PIT algorithm for circuits models with low dimensional partial derivative space. This algorithm runs in polynomial time when we are in log-variate regime. For a polynomial $f(\mathbf{x}) \in \mathbb{F}[\mathbf{x}]$, by $\partial_{\mathbf{x}<\infty}(f)$ we denote the space generated all partial derivatives of f.

Proof of Theorem 1. The proof has two steps. First, we show that with respect to any monomial ordering $\prec$ (say lexicographic monomial ordering), for all nonzero $P \in \mathcal{P}$, the dimension of the partial derivative space of P is lower bounded by the cone-size of the leading monomial in P. Using this, we can get a blackbox PIT algorithm for $\mathcal{P}$ by testing the coefficients of all the monomials of P of cone-size $\leq k$ for zeroness. Next, we analyze the time complexity to do this.

The first part is the same as the proof of [14, Corollary 4.14] (with origins in [20]). Here, we give a brief outline. Let $LM(\cdot)$ be the *leading monomial* operator wrt the monomial ordering $\prec$. It can be shown that for any polynomial $f(\mathbf{x})$, the dimension of its partial derivative space $\partial_{\mathbf{x}^{<\infty}}(f)$ is the same as $D := \#\{LM(g) \mid g \in \partial_{\mathbf{x}^{<\infty}}(f)\}$ (see [14, Lemma 8.4.12]). This means that $\dim \partial_{\mathbf{x}^{<\infty}}(f)$ is lower-bounded by the cone-size of $LM(f)$ [14, Corollary 8.4.13], which completes the proof of our first part.

Next, we apply Lemma 4, on the circuit of P and a monomial m of cone-size $\leq k$, to get the coefficient of m in C in poly(sdk)-time. Finally, Lemma 5 tells that we have to access at most $k^2 \cdot (3n/\log k)^{\log k}$ many monomials m. Multiplying these two expressions gives us the time bound. ◀

This gives us immediately,

▶ **Corollary 6.** *Let $\mathbb{F}$ be a field of characteristic 0 or $> d$. Let $\mathcal{P}$ be a set of n-variate d-degree polynomials, over $\mathbb{F}$, computable by circuits of bitsize s; with $n = O(\log sd)$. Suppose that, for all $P \in \mathcal{P}$, the dimension of the partial derivative space of P is poly(sd). Then, blackbox PIT for $\mathcal{P}$ can be solved in poly(sd)-time.*

Now we discuss our result regarding depth-3 diagonal circuits $\Sigma \wedge \Sigma$.

▶ **Definition 7** (Depth-3 diagonal circuit and its rank)**.** A *depth-3 diagonal circuit* is of the form $\Sigma \wedge \Sigma$ (sum-power-sum). It computes a polynomial presented as $C(\mathbf{x}) = \sum_{i\in[k]} c_i \ell_i^{d_i}$, where ℓ_i's are linear polynomials over $\mathbb{F}$ and c_i's in $\mathbb{F}$.

By $\mathrm{rk}(C)$ we denote the linear rank of the polynomials $\{\ell_i\}_{i\in[k]}$.

The next lemma introduces an efficient *nonzeroness preserving variable reduction* map $(n \mapsto \mathrm{rk}(C))$ for depth-3 diagonal circuits. For a set of n-variate circuits $\mathcal{C}$ over $\mathbb{F}$, a *polynomial map* $\Psi : \mathbb{F}^m \to \mathbb{F}^n$ is called nonzeroness preserving variable reduction map for $\mathcal{C}$, if $m < n$ and for all $C \in \mathcal{C}$, $C \neq 0$ if and only if $\Psi(C) \neq 0$.

▶ **Lemma 8** (Variable reduction)**.** *Let $P(\mathbf{x})$ be an n-variate d-degree polynomial computed by a size-s depth-3 diagonal circuit over some sufficiently large field $\mathbb{F}$. Then, there exists a* poly(nds)*-time computable nonzeroness preserving variable reduction map which converts P to another $rk(P)$-variate degree-d polynomial computed by* poly(s)*-size depth-3 diagonal circuit.*

For proof, see the full version linked on the first page.

▶ **Theorem 9** (Log-rank $\Sigma \wedge \Sigma$)**.** *Let $\mathbb{F}$ be a field of characteristic 0 or $> d$. Let $\mathcal{P}$ be the set of n-variate d-degree polynomials P, computable by depth-3 diagonal circuits of bitsize s, with $rk(P) = O(\log sd)$. Then, blackbox PIT for $\mathcal{P}$ can be solved in poly(sd)-time.*

Proof. The above description gives us a non-zeroness preserving variable reduction $(n \mapsto \mathrm{rk}(P))$ method that reduces P to an $O(\log(sd))$-variate and degree-d polynomial P' computed by poly(s)-size depth-3 diagonal circuit.

Since the dimension of the partial derivative space of P' is poly(sd) [14, Lem.8.4.8], Corollary 6 gives us a poly(sd)-time hitting-set for P'. ◀

3 Cone-closed basis after shifting– Proof of Theorem 2

In this section we will consider polynomials over a vector space, say $\mathbb{F}^k$. This viewpoint has been useful in studying algebraic branching programs (ABP), eg. [6, 18, 3, 26]. Let $D \in \mathbb{F}^k[\mathbf{x}]$ and let $\mathrm{sp}(D)$ be the vector space spanned by its coefficients. Now, we formally define various kinds of rank concentrations of D.

▶ **Definition 10** (Rank Concentration)**.** We say that D has a

1. *cone-closed basis* if there is a cone-closed set of monomials B (see Definition 3) whose coefficients in D form a basis of $\mathrm{sp}(D)$.
2. *ℓ-support concentration*, if there is a set of monomials B with support size less than ℓ whose coefficients form a basis of $\mathrm{sp}(D)$.
3. *ℓ-cone concentration*, if there is a set of monomials B with cone size less than ℓ (see Definition 3) whose coefficients form a basis of $\mathrm{sp}(D)$.

In the next lemma, we show that cone-closed basis notion subsumes the other two notions.

▶ **Lemma 11.** *Let $D(\mathbf{x})$ be a polynomial in $\mathbb{F}^k[\mathbf{x}]$. Suppose that $D(\mathbf{x})$ has a cone-closed basis. Then, $D(\mathbf{x})$ has $(k+1)$-cone concentration and $(\lg 2k)$-support concentration.*

Proof. Let B be a cone-closed set of monomials forming the basis of $\mathrm{sp}(D)$. Clearly, $|B| \leq k$. Thus, each $m \in B$ has cone-size $\leq k$. In other words, D is $(k+1)$-cone concentrated.

Moreover, each $m \in B$ has support-size $\leq \lg k$. In other words, D is $(\lg 2k)$-support concentrated. ◀

Next, we define the notions which will be used in the proof of Theorem 2.

Basis & weights. Consider a weight assignment $\mathbf{w} = (w_1, \ldots, w_n) \in \mathbb{N}^n$ on the variables $\mathbf{x} = (x_1, \ldots, x_n)$. It extends to monomials $m = \mathbf{x}^{\mathbf{e}}$ as $\mathbf{w}(m) := \langle \mathbf{e}, \mathbf{w} \rangle = \sum_{i=1}^{n} e_i w_i$. Sometimes, we also use $\mathbf{w}(\mathbf{e})$ to denote $\mathbf{w}(m)$. Similarly, for a set of monomials B, the weight of B is $\mathbf{w}(B) := \sum_{m \in B} \mathbf{w}(m)$.

Let $B = \{m_1, \ldots, m_\ell\}$ resp. $B' = \{m'_1, \ldots, m'_\ell\}$ be an ordered set of monomials (non-decreasing wrt $\mathbf{w}$) that forms a basis of the span of coefficients of $f \in \mathbb{F}^k[\mathbf{x}]$. Let $\mathbf{w}$ be a weight assignment on the variables. We say that $B < B'$ wrt $\mathbf{w}$, if there exists $i \in [\ell]$ such that $\forall j < i,\ \mathbf{w}(m_j) = \mathbf{w}(m'_j)$ but $\mathbf{w}(m_i) < \mathbf{w}(m'_i)$.

We say that $B \leq B'$ if either $B < B'$ or if $\forall i \in [\ell],\ \mathbf{w}(m_i) \leq \mathbf{w}(m'_i)$. A basis B is called a *least basis*, if for any other basis B', $B \leq B'$. Next, we describe a condition on $\mathbf{w}$ such that least basis will be unique.

▶ **Definition 12.** (Basis Isolating Weight Assignment [3, Defn.5])**.** A weight assignment $\mathbf{w}$ is called a *basis isolating weight assignment* for a polynomial $f(\mathbf{x}) \in \mathbb{F}^k[\mathbf{x}]$ if there exists a set of monomials B such that:

1. the coefficients of the monomials in B form a basis for $\mathrm{sp}(f)$,
2. weights of all monomials in B are distinct, and
3. the coefficient of every $m \in \mathrm{supp}(f) \setminus B$ is in the linear span of $\{\mathrm{coef}_{m'}(f) \mid m' \in B, \mathbf{w}(m') < \mathbf{w}(m)\}$.

▶ **Lemma 13.** *If $\mathbf{w}$ is a basis isolating weight assignment for $f \in \mathbb{F}^k[\mathbf{x}]$, then f has a unique least basis B wrt $\mathbf{w}$. In particular, for any other basis B' of f, we have $\mathbf{w}(B) < \mathbf{w}(B')$.*

Algorithm 1 Finding cone-closed set.

Input: A subset B of the n-tuples M.
Output: A cone-closed $A \subseteq M$ with full rank $T_{A,B}$.
function FIND-CONE-CLOSED(B, n)
  if $n = 1$ **then**
    $s \leftarrow |B|$;
   return $\{0, \ldots, s-1\}$;
  else
    Let π_n be the map which projects the set of monomials B on the first $n-1$ variables;
    Let ℓ be the maximum number of preimages under π_n;
    $\forall i \in [\ell]$, F_i collects those elements in $\mathrm{Img}(\pi_n)$ whose preimage size$\geq i$;
    $A_0 \leftarrow \emptyset$;
    for $i \leftarrow 1$ to ℓ **do**
      $S_i \leftarrow$ FIND-CONE-CLOSED$(F_i, n-1)$;
      $A_i \leftarrow A_{i-1} \bigcup (S_i \times \{i-1\})$;
    end for
   return A;
  end if
end function

For proof, see the full version linked on the first page. Next, we want to study the effect of shifting f by a basis isolating weight assignment. To do that we require an elaborate notation. As before $f(\mathbf{x})$ is a n-variate and degree-d polynomial over $\mathbb{F}^k$. For a weight assignment $\mathbf{w}$, by $f(\mathbf{x} + t^{\mathbf{w}})$ we denote the polynomial $f(x_1 + t^{w_1}, \ldots, x_n + t^{w_n})$. For $\mathbf{a} = (a_1, \ldots, a_n)$ and $\mathbf{b} = (b_1, \ldots, b_n)$ in $\mathbb{N}^n$, $\binom{\mathbf{a}}{\mathbf{b}}$ denotes $\prod_{i=1}^{n} \binom{a_i}{b_i}$, where $\binom{a_i}{b_i} = 1$ for $b_i = 0$ and $\binom{a_i}{b_i} = 0$ for $a_i < b_i$. Let $M_{n,d} = \{\mathbf{a} \in \mathbb{N}^n : |\mathbf{a}|_1 \leq d\}$ corresponds to the set of all n-variate d-degree monomials. For every $\mathbf{a} \in M_{n,d}$, $\mathrm{coef}_{\mathbf{x}^{\mathbf{a}}}(f(\mathbf{x} + t^{\mathbf{w}}))$ can be expanded using the binomial expansion, and we get:

$$\sum_{\mathbf{b} \in M_{n,d}} \binom{\mathbf{b}}{\mathbf{a}} \cdot t^{\mathbf{w}(\mathbf{b}) - \mathbf{w}(\mathbf{a})} \cdot \mathrm{coef}_{\mathbf{x}^{\mathbf{b}}}(f(\mathbf{x}))\,. \tag{2}$$

We express this data in matrix form as

$$F' = D^{-1} T D \cdot F, \tag{3}$$

where the matrices involved are,

1. F and F': rows are indexed by the elements of $M_{n,d}$ and columns are indexed by $[k]$. In F resp. F' the $\mathbf{a}$-th row is $\mathrm{coef}_{\mathbf{x}^{\mathbf{a}}}(f(\mathbf{x}))$ resp. $\mathrm{coef}_{\mathbf{x}^{\mathbf{a}}}(f(\mathbf{x} + t^{\mathbf{w}}))$.
2. D: is a diagonal matrix with both the rows and columns indexed by $M_{n,d}$. For $\mathbf{a} \in M_{n,d}$, $D_{\mathbf{a},\mathbf{a}} := t^{\mathbf{w}(\mathbf{x}^{\mathbf{a}})}$.
3. T: both the rows and columns are indexed by $M_{n,d}$. For $\mathbf{a}, \mathbf{b} \in M_{n,d}$, $T_{\mathbf{a},\mathbf{b}} := \binom{\mathbf{b}}{\mathbf{a}}$. It is known as *transfer matrix*.

We will prove the following combinatorial property of T: For any $B \subseteq M_{n,d}$, there is a *cone-closed* $A \subseteq M_{n,d}$ such that the submatrix $T_{A,B}$ has full rank. Our proof is an involved double-induction, so we describe the construction of A as Algorithm 1.

▶ **Lemma 14** (Comparison). *Let B and B' be two nonempty subsets of M such that $B \subseteq B'$. Let $A =$ FIND-CONE-CLOSED(B, n) and $A' =$ FIND-CONE-CLOSED(B', n) in Algorithm 1. Then $A \subseteq A'$.*

▶ **Lemma 15** (Closure). *Let B be a nonempty subset of M. If $A = \text{FIND-CONE-CLOSED}(B, n)$ in Algorithm 1, then A is cone-closed. Moreover, $|A| = |B|$.*

For proofs of the above two lemmas, see the full version linked on the first page. Next, we recall a fact that has been used for ROABP PIT.

▶ **Lemma 16.** *[25, Claim 3.3] Let $a_1, \ldots, a_n$ be distinct non-negative integers and char $\mathbb{F} = 0$ or greater than the maximum of all a_is. Let A be an $n \times n$ matrix with, $i, j \in [n]$, $A_{i,j} := \binom{a_j}{i-1}$. Then, A is full rank.*

In the following lemma, we prove that the sub-matrix $T_{A,B}$ has full rank, where $B \subseteq M_{n,d}$ and A is the output of Algorithm 1 on input A. It requires char $\mathbb{F} = 0$ or greater than d.

▶ **Lemma 17** (Full rank). *If $A = \text{FIND-CONE-CLOSED}(B, n)$, then $T_{A,B}$ has full rank.*

Proof. The proof will be by double-induction– outer induction on n and an inner induction on iteration i of the 'for' loop (Algorithm 1).

Base case: For $n = 1$, the claim is true due to Lemma 16.

Induction step $(n-1 \to n)$*:* To show $T_{A,B}$ full rank, we prove that for any vector $\mathbf{b} \in \mathbb{F}^{|B|}$: if $T_{A,B} \cdot \mathbf{b} = 0$ then $\mathbf{b} = 0$. For this we show that the following invariant holds at the end of each iteration i of the 'for' loop (Algorithm 1). Here, we assume the coordinates of $\mathbf{b}$ are indexed by the elements of B and for all $\mathbf{f} \in B$, $\mathbf{b_f}$ denotes the value of $\mathbf{b}$ at coordinate $\mathbf{f}$.

Invariant (n-variate & i-th iteration): For each $\mathbf{f} \in B$ such that the preimage size of $\pi_n(\mathbf{f})$ is at most i, the product $T_{A_i,B} \cdot \mathbf{b} = 0$ implies that $\mathbf{b_f} = 0$. Here,

At the end of iteration $i = 1$, we have the vector $T_{A_1,B} \cdot \mathbf{b}$. Recall that $A_1 = S_1 \times \{0\}$ and $F_1 = \pi_n(B)$. So $T_{A_1,B} \cdot \mathbf{b} = T_{S_1,F_1} \cdot \mathbf{c}$, where $\mathbf{c} \in \mathbb{F}^{|F_1|}$ and for $\mathbf{e} \in F_1$, $\mathbf{c_e} := \sum_{(\mathbf{e},k) \in \pi_n^{-1}(\mathbf{e})} \binom{k}{0} \mathbf{b}_{(\mathbf{e},k)}$. Thus, $T_{A_1,B} \cdot \mathbf{b} = 0$ implies $T_{S_1,F_1} \cdot \mathbf{c} = 0$. Since $S_1 = \text{FIND-CONE-CLOSED}(F_1, n-1)$, using induction hypothesis, we get that $\mathbf{c} = 0$. This means that for $\mathbf{e} \in B$ such that the preimage size of $\pi_n(\mathbf{e})$ is at most 1, we have $\mathbf{c_e} = 0$. This proves our invariant at the end of the iteration $i = 1$.

$(i-1 \to i)$: Suppose that at the end of $(i-1)$-th iteration, the invariant holds. We show that it also holds at the end of the i-th iteration. For each $j \in [i]$, let $\mathbf{v}_j$ denote the projection of $T_{A_i,B} \cdot \mathbf{b}$ on the coordinates indexed by $S_j \times \{j-1\}$. By focusing on the rows of $T_{A_j,B}$, we can see that $\mathbf{v}_j = T_{S_j,F_1} \cdot \mathbf{c}_j$ where the vector $\mathbf{c}_j \in \mathbb{F}^{|F_1|}$ is defined as, for $\mathbf{e} \in F_1$,

$$\mathbf{c}_{j_\mathbf{e}} := \sum_{(\mathbf{e},k) \in \pi_n^{-1}(\mathbf{e})} \binom{k}{j-1} \cdot \mathbf{b}_{(\mathbf{e},k)} \,. \tag{4}$$

Suppose that $T_{A_i,B} \cdot \mathbf{b} = 0$. Because of the invariant at $i-1$th round, for all $\mathbf{f} \in B$ with preimage size of $\pi_n(\mathbf{f})$ is less than i, $\mathbf{b_f} = 0$. So all we have to argue is that for every $\mathbf{f} \in B$ such that the preimage size of $\mathbf{e} := \pi_n(\mathbf{f})$ is i, the coordinate $\mathbf{b_f} = 0$.

To prove our goal, first we show that each $\mathbf{c}_j$ is a zero vector. Since $T_{A_i,B} \cdot \mathbf{b} = 0$, its projection $\mathbf{v}_j = T_{S_j,F_1} \cdot \mathbf{c}_j$ is zero too. By induction hypothesis (on $i-1$), for each $\mathbf{e} \in F_1$ with preimage size $< i$, the coordinate $\mathbf{c}_{j\mathbf{e}} = 0$. Thus, the vector $T_{S_j,F_1} \cdot \mathbf{c}_j = T_{S_j,F_j} \cdot \mathbf{c}'_j$ where the vector $\mathbf{c}'_j \in \mathbb{F}^{|F_j|}$ is defined as, for $\mathbf{e} \in F_j$, $\mathbf{c}'_{j\mathbf{e}} := \mathbf{c}_{j\mathbf{e}}$. Consequently, $T_{S_j,F_j} \cdot \mathbf{c}'_j = 0$, for $j \in [i]$. By induction hypothesis (on $n-1$), we know that T_{S_j,F_j} is full rank. So $\mathbf{c}'_j = 0$, which tells us that $\mathbf{c}_j = 0$, for $j \in [i]$.

Fix an $\mathbf{e} \in F_1$, with preimage size $= i$, and let the preimages be $\{(\mathbf{e}, k_1), \ldots, (\mathbf{e}, k_i)\}$

where k_j's are distinct nonnegative integers. From Equation 4, we can write

$$\begin{bmatrix} \mathbf{c}_{1\mathbf{e}} \\ \mathbf{c}_{2\mathbf{e}} \\ \vdots \\ \mathbf{c}_{i\mathbf{e}} \end{bmatrix} = \begin{bmatrix} \binom{k_1}{0} & \binom{k_2}{0} & \cdots & \binom{k_i}{0} \\ \binom{k_1}{1} & \binom{k_2}{1} & \cdots & \binom{k_i}{1} \\ \vdots & \vdots & \cdots & \vdots \\ \binom{k_1}{i-1} & \binom{k_2}{i-1} & \cdots & \binom{k_i}{i-1} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{b}_{(\mathbf{e},k_1)} \\ \mathbf{b}_{(\mathbf{e},k_2)} \\ \vdots \\ \mathbf{b}_{(\mathbf{e},k_i)} \end{bmatrix}.$$

Since for each $j \in [i]$, $\mathbf{c}_j$ is a zero vector, from the above equation we get

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \binom{k_1}{0} & \binom{k_2}{0} & \cdots & \binom{k_i}{0} \\ \binom{k_1}{1} & \binom{k_2}{1} & \cdots & \binom{k_i}{1} \\ \vdots & \vdots & \cdots & \vdots \\ \binom{k_1}{i-1} & \binom{k_2}{i-1} & \cdots & \binom{k_i}{i-1} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{b}_{(\mathbf{e},k_1)} \\ \mathbf{b}_{(\mathbf{e},k_2)} \\ \vdots \\ \mathbf{b}_{(\mathbf{e},k_i)} \end{bmatrix}.$$

Now invoking Lemma 16, we get $\mathbf{b}_{(\mathbf{e},k_j)} = 0$ for all $j \in [i]$. In other words, for any $\mathbf{f} \in B$ such that the preimage size of $\pi_n(\mathbf{f})$ is i, the coordinate $\mathbf{b}_{\mathbf{f}} = 0$.

($i = \ell$): Since $A = A_\ell$, the output of FIND-CONE-CLOSED(B, n), using our invariant at the end of ℓ-th iteration we deduce that $T_{A,B} \cdot \mathbf{b} = 0$ implies $\mathbf{b} = 0$. Thus, $T_{A,B}$ has full rank. ◀

Now we are ready to prove our main theorem using the transfer matrix equation.

Proof of Theorem 2. As we mentioned in Equation 2, the shifted polynomial $f(\mathbf{x} + t^{\mathbf{w}})$ yields a matrix equation $F' = D^{-1}TD \cdot F$. Let k' be the rank of F. We consider the following two cases.

Case 1 ($k' < k$): We reduce this case to the other one where $k' = k$. Let S be a subset of k' columns such that $F_{M,S}$ has rank k'. The matrix $F_{M,S}$ denotes the polynomial $f_S(\mathbf{x}) \in \mathbb{F}[\mathbf{x}]^{k'}$, where $f_S(\mathbf{x})$ is the projection of the 'vector' $f(\mathbf{x})$ on the coordinates indexed by S. So, any linear dependence relation among the coefficients of $f(\mathbf{x})$ is also valid for $f_S(\mathbf{x})$. So $\mathbf{w}$ is also a basis isolating weight assignment for $f_S(\mathbf{x})$. Now from our Case 2, we can claim that $f_S(\mathbf{x}+t^{\mathbf{w}})$ has a cone-closed basis A. Thus, coefficients of the monomials, corresponding to A, in $f(\mathbf{x})$ form a basis of $\mathrm{sp}(f)$. This implies that $f(\mathbf{x} + t^{\mathbf{w}})$ has a cone-closed basis A.

Case 2 ($k' = k$): Let B be the least basis of $f(\mathbf{x})$ wrt $\mathbf{w}$ and A = FIND-CONE-CLOSED(B, n). We prove that the coefficients of monomials in A form a basis of the coefficient space of $f(\mathbf{x} + t^{\mathbf{w}})$. To prove this, we show that $\det(F'_{A,[k]}) \neq 0$. Define $T' := TDF$ so that $F' = D^{-1}T'$. Using Cauchy-Binet formula [57], we get that

$$\det(F'_{A,[k]}) = \sum_{C \in \binom{M}{k}} \det(D^{-1}_{A,C}) \cdot \det(T'_{C,[k]}).$$

Since for all $C \in \binom{M}{k} \setminus \{A\}$, the matrix $D^{-1}_{A,C}$ is singular, we have $\det(F'_{A,[k]}) = \det(D^{-1}_{A,A}) \cdot \det(T'_{A,[k]})$. Again applying Cauchy-Binet formula for $\det(T'_{A,[k]})$, we get

$$\det(F'_{A,[k]}) = \det(D^{-1}_{A,A}) \cdot \sum_{C \in \binom{M}{k}} t^{\mathbf{w}(C)} \det(T_{A,C}) \cdot \det(F_{C,[k]}).$$

From Lemma 13, we have that for all basis $C \in \binom{M}{k} \setminus \{B\}$, $\mathbf{w}(C) > \mathbf{w}(B)$. The matrix $T_{A,B}$ is nonsingular by Lemma 17, and the other one $F_{B,[k]}$ is nonsingular since B is a basis. Hence, the sum is a nonzero polynomial in t. In particular, $\det(F'_{A,[k]}) \neq 0$, which ensures that the coefficients of the monomials corresponding to A form a basis of $\mathrm{sp}_{\mathbb{F}(t)}(f(\mathbf{x} + t^{\mathbf{w}}))$. Since Lemma 15 says that A is also cone-closed, we get that $f(\mathbf{x} + t^{\mathbf{w}})$ has a cone-closed basis. ◀

3.1 Models with a cone-closed basis

We give a simple proof showing that a typical diagonal depth-3 circuit is already cone-closed. Consider the polynomial $D(\mathbf{x}) = (\mathbf{1} + \mathbf{a}_1 x_1 + \ldots + \mathbf{a}_n x_n)^d$ in $\mathbb{F}^k[\mathbf{x}]$, where $\mathbb{F}^k$ is seen as an $\mathbb{F}$-algebra with coordinate-wise multiplication.

▶ **Lemma 18.** *$D(\mathbf{x})$ has a cone-closed basis.*

Proof. Consider the n-tuple $L := (\mathbf{a}_1, \ldots, \mathbf{a}_n)$. Then for every monomial $\mathbf{x}^\mathbf{e}$, the coefficient of $\mathbf{x}^\mathbf{e}$ in D is $L^\mathbf{e} := \prod_{i=1}^n \mathbf{a}_i^{e_i}$, with some nonzero scalar factor (note: here we seem to need char($\mathbb{F}$) zero or large). We ignore this constant factor, since it does not affect linear dependence relations. Consider *deg-lex* monomial ordering, i.e. first order the monomials by lower to higher total degree, then within each degree arrange them according to a lexicographic order. Now we prove that the 'least basis' of $D(\mathbf{x})$ with respect to this monomial ordering is cone-closed.

We incrementally devise a monomial set B as follows: Arrange all the monomials in ascending order. Starting from least monomial, put a monomial in B if its coefficient cannot be written as a linear combination of its previous (thus, smaller) monomials. From construction, the coefficients of monomials in B form the least basis for the coefficient space of $D(\mathbf{x})$. Now we show that B is cone-closed. We prove it by contradiction.

Let $\mathbf{x}^\mathbf{f} \in B$ and let $\mathbf{x}^\mathbf{e}$ be its submonomial that is not in B. Then we can write

$$L^\mathbf{e} = \sum_{\mathbf{x}^\mathbf{b} \prec \mathbf{x}^\mathbf{e}} c_\mathbf{b} L^\mathbf{b} \quad \text{with } c_\mathbf{b}\text{'s in } \mathbb{F}\,.$$

Multiplying by $L^{\mathbf{f}-\mathbf{e}}$ on both sides, we get

$$L^\mathbf{f} = \sum_{\mathbf{x}^\mathbf{b} \prec \mathbf{x}^\mathbf{e}} c_\mathbf{b} L^{\mathbf{b}+\mathbf{f}-\mathbf{e}} = \sum_{\mathbf{x}^{\mathbf{b}'} \prec \mathbf{x}^\mathbf{f}} c'_{\mathbf{b}'} L^{\mathbf{b}'}\,.$$

Note that $\mathbf{x}^{\mathbf{b}'} \prec \mathbf{x}^\mathbf{f}$ holds true by the way a monomial ordering is defined. This equation contradicts the fact that $\mathbf{x}^\mathbf{f} \in B$, and completes the proof. ◀

4 Conclusion

Since it is known that one could focus solely on the PIT of VP circuits that depend only on the first $o(\log s)$ variables, we initiate a study of properties that are useful in that regime. These properties are– low-cone concentration and cone-closed basis. Their usefulness is proved in our monomial counting and coefficient extraction results. Using these concepts we solve an interesting special case of diagonal depth-3 circuits.

An open question is to make our approach work for field characteristic smaller than the degree. Another interesting problem is to employ the cone-closed basis properties of the $\Sigma \wedge \Sigma^n$ model to devise a poly-time blackbox PIT for general n.

In our second result, we proved that after shifting the variables by a basis isolating weight assignment, a polynomial has a cone-closed basis. Basis isolating weight assignment is much weaker than the one induced by lexicographic monomial ordering (or the Kronecker map). An interesting open question is to *efficiently* design a weight assignment (or, in general, polynomial map) that ensures a cone closed basis. Till now, no known blackbox PIT algorithm for ROABPs gives a polynomial time blackbox PIT algorithm for log (or sub-log) variate ROABPs. So, achieving cone-closed basis or low-cone concentration property (in polynomial time) for log (or sub-log) variate ROABPs is also interesting; then, the counting & extraction techniques developed in our first result will give a polynomial time blackbox PIT. This will solve some open problems posed in [2, Sec.6].

References

1 Manindra Agrawal. Proving lower bounds via pseudo-random generators. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, Hyderabad, India, December 15-18, 2005, Proceedings*, pages 92–105, 2005.

2 Manindra Agrawal, Sumanta Ghosh, and Nitin Saxena. Bootstrapping variables in algebraic circuits. Technical report, https://www.cse.iitk.ac.in/users/nitin/research.html, 2017. (To appear in 50th ACM Symposium on Theory of Computing (STOC), 2018).

3 Manindra Agrawal, Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Hitting-sets for ROABP and sum of set-multilinear circuits. *SIAM Journal on Computing*, 44(3):669–697, 2015.

4 Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of mathematics*, pages 781–793, 2004.

5 Manindra Agrawal, Chandan Saha, Ramprasad Saptharishi, and Nitin Saxena. Jacobian hits circuits: hitting-sets, lower bounds for depth-d occur-k formulas & depth-3 transcendence degree-k circuits. In *STOC*, pages 599–614, 2012.

6 Manindra Agrawal, Chandan Saha, and Nitin Saxena. Quasi-polynomial hitting-set for set-depth-Δ formulas. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 321–330, 2013.

7 M. Beecken, J. Mittmann, and N. Saxena. Algebraic Independence and Blackbox Identity Testing. *Inf. Comput.*, 222:2–19, 2013. (Conference version in ICALP 2011).

8 Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 301–309, 1988.

9 Xi Chen, Neeraj Kayal, and Avi Wigderson. Partial derivatives in arithmetic complexity and beyond. *Foundations and Trends in Theoretical Computer Science*, 6(1-2):1–138, 2011.

10 Richard A. Demillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978.

11 Zeev Dvir, Rafael Mendes de Oliveira, and Amir Shpilka. Testing Equivalence of Polynomials under Shifts. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 417–428. Springer International Publishing, 2014. doi:10.1007/978-3-662-43948-7_35.

12 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-NC. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 754–763, 2016.

13 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Guest column: Parallel algorithms for perfect matching. *SIGACT News*, 48(1):102–109, 2017.

14 Michael A. Forbes. *Polynomial Identity Testing of Read-Once Oblivious Algebraic Branching Programs*. PhD thesis, Massachusetts Institute of Technology, 2014.

15 Michael A Forbes. Deterministic divisibility testing via shifted partial derivatives. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 451–465. IEEE, 2015.

16 Michael A. Forbes, Ankit Gupta, and Amir Shpilka. private communication, 2013.

17 Michael A. Forbes, Ramprasad Saptharishi, and Amir Shpilka. Pseudorandomness for multilinear read-once algebraic branching programs, in any order. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:132, 2013.

18 Michael A. Forbes, Ramprasad Saptharishi, and Amir Shpilka. Hitting sets for multilinear read-once algebraic branching programs, in any order. In *Symposium on Theory of Computing (STOC), New York, NY, USA, May 31 - June 03, 2014*, pages 867–875, 2014.

19 Michael A. Forbes and Amir Shpilka. On identity testing of tensors, low-rank recovery and compressed sensing. In *STOC*, pages 163–172, 2012.

20 Michael A Forbes and Amir Shpilka. Explicit noether normalization for simultaneous conjugation via polynomial identity testing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 527–542. Springer, 2013.

21 Ignacio García-Marco, Pascal Koiran, Timothée Pecatte, and Stéphan Thomassé. On the complexity of partial derivatives. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 37:1–37:13, 2017.

22 Ankit Garg, Leonid Gurvits, Rafael Oliveira, and Avi Wigderson. A deterministic polynomial time algorithm for non-commutative rational identity testing. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016*, pages 109–117, 2016.

23 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic circuits: A chasm at depth three. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 578–587, 2013.

24 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic circuits: A chasm at depth 3. *SIAM Journal on Computing*, 45(3):1064–1079, 2016.

25 Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Identity testing for constant-width, and any-order, read-once oblivious arithmetic branching programs. *Theory of Computing*, 13(2):1–21, 2017. (Preliminary version in CCC'16).

26 Rohit Gurjar, Arpita Korwar, Nitin Saxena, and Thomas Thierauf. Deterministic identity testing for sum of read-once oblivious arithmetic branching programs. *Computational Complexity*, pages 1–46, 2016. (Conference version in CCC 2015).

27 Rohit Gurjar and Thomas Thierauf. Linear matroid intersection is in quasi-nc. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 821–830, 2017.

28 Rohit Gurjar, Thomas Thierauf, and Nisheeth K. Vishnoi. Isolating a vertex via lattices: Polytopes with totally unimodular faces. *CoRR*, abs/1708.02222, 2017.

29 Joos Heintz and Claus-Peter Schnorr. Testing polynomials which are easy to compute (extended abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, pages 262–272, 1980.

30 Stasys Jukna. *Extremal Combinatorics: With Applications in Computer Science*. Springer Publishing Company, Incorporated, 1st edition, 2010.

31 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 355–364, 2003.

32 Neeraj Kayal. Algorithms for arithmetic circuits. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:73, 2010.

33 Neeraj Kayal and Nitin Saxena. Polynomial identity testing for depth 3 circuits. *Computational Complexity*, 16(2):115–138, 2007.

34 Adam R. Klivans and Daniel A. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 216–223, 2001.

35 Swastik Kopparty, Shubhangi Saraf, and Amir Shpilka. Equivalence of polynomial identity testing and deterministic multivariate polynomial factorization. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 169–180, 2014.

36 Mrinal Kumar and Shubhangi Saraf. Arithmetic circuits with locally low algebraic rank. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 34:1–34:27, 2016.

37 Mrinal Kumar and Shubhangi Saraf. Sums of products of polynomials in few variables: Lower bounds and polynomial identity testing. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 35:1–35:29, 2016.

38 Guillaume Lagarde, Guillaume Malod, and Sylvain Perifel. Non-commutative computations: lower bounds and polynomial identity testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:94, 2016.

39 Richard J. Lipton and Nisheeth K. Vishnoi. Deterministic identity testing for multivariate polynomials. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 756–760, 2003.

40 Ketan Mulmuley. Geometric complexity theory V: Efficient algorithms for Noether normalization. *Journal of the American Mathematical Society*, 30(1):225–309, 2017.

41 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 345–354, 1987.

42 Ketan D Mulmuley. The GCT program toward the P vs. NP problem. *Communications of the ACM*, 55(6):98–107, 2012.

43 Ketan D. Mulmuley. Geometric complexity theory V: Equivalence between blackbox derandomization of polynomial identity testing and derandomization of Noether's normalization lemma. In *FOCS*, pages 629–638, 2012.

44 Noam Nisan and Avi Wigderson. Lower bounds for arithmetic circuits via partial derivatives (preliminary version). In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*, pages 16–25, 1995.

45 Anurag Pandey, Nitin Saxena, and Amit Sinhababu. Algebraic independence over positive characteristic: New criterion and applications to locally low algebraic rank circuits. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 74:1–74:15, 2016. (In print, Computational Complexity, 2018).

46 Chandan Saha, Ramprasad Saptharishi, and Nitin Saxena. A case of depth-3 identity testing, sparse factorization and duality. *Computational Complexity*, 22(1):39–69, 2013.

47 Ramprasad Saptharishi. personal communication, 2013.

48 Ramprasad Saptharishi. *Unified Approaches to Polynomial Identity Testing and Lower Bounds*. PhD thesis, Chennai Mathematical Institute, 2013.

49 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. Technical report, https://github.com/dasarpmar/lowerbounds-survey/, 2016.

50 Nitin Saxena. Diagonal circuit identity testing and lower bounds. In *ICALP*, volume 5125 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 2008.

51 Nitin Saxena. Progress on polynomial identity testing. *Bulletin of the EATCS*, 99:49–79, 2009.

52 Nitin Saxena. Progress on polynomial identity testing- II. In *Perspectives in Computational Complexity*, volume 26 of *Progress in Computer Science and Applied Logic*, pages 131–146. Springer International Publishing, 2014.

53 Nitin Saxena and C. Seshadhri. Blackbox identity testing for bounded top-fanin depth-3 circuits: The field doesn't matter. *SIAM Journal on Computing*, 41(5):1285–1298, 2012.

54 J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

55 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010.

56 Avi Wigderson. Low-depth arithmetic circuits: technical perspective. *Communications of the ACM*, 60(6):91–92, 2017.

57 Jiang Zeng. A bijective proof of Muir's identity and the Cauchy-Binet formula. *Linear Algebra and its Applications*, 184:79–82, 1993.

58 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, EUROSAM '79, pages 216–226, 1979.

Finding Cliques in Social Networks: A New Distribution-Free Model

Jacob Fox[1]
Department of Mathematics, Stanford University, Stanford, CA 94305, USA
jacobfox@stanford.edu

Tim Roughgarden[2]
Department of Computer Science, Stanford University, Stanford, CA 94305, USA

C. Seshadhri
Department of Computer Science, University of California, Santa Cruz, CA 95064, USA

Fan Wei
Department of Mathematics, Stanford University, Stanford, CA 94305, USA

Nicole Wein[3]
EECS, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

Abstract

We propose a new distribution-free model of social networks. Our definitions are motivated by one of the most universal signatures of social networks, triadic closure – the property that pairs of vertices with common neighbors tend to be adjacent. Our most basic definition is that of a *c-closed* graph, where for every pair of vertices u, v with at least c common neighbors, u and v are adjacent. We study the classic problem of enumerating all maximal cliques, an important task in social network analysis. We prove that this problem is fixed-parameter tractable with respect to c on c-closed graphs. Our results carry over to *weakly c-closed graphs*, which only require a vertex deletion ordering that avoids pairs of non-adjacent vertices with c common neighbors. Numerical experiments show that well-studied social networks tend to be weakly c-closed for modest values of c.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Graph algorithms, social networks, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.55

Related Version A full version of the paper is available at [29], `https://arxiv.org/abs/1804.07431`.

Acknowledgements We would like to thank Christina Gilbert for writing the code to calculate the c-closure and weak c-closure of networks in the SNAP data sets. We would also like to thank Virginia Vassilevska Williams and Josh Alman for useful conversations about turning our bound into an algorithm.

[1] Supported by a Packard Fellowship, an NSF Career Award DMS-1352121, and an Alfred P. Sloan Fellowship
[2] Supported by NSF award CCF-1524062
[3] Supported by an NSF Graduate Fellowship

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 55; pp. 55:1–55:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

There has been an enormous amount of important work over the past 15 years on models for capturing the special structure of social networks. This literature is almost entirely driven by the quest for generative (i.e., probabilistic) models. Well-known examples of such models include preferential attachment [6], the copying model [38], Kronecker graphs [13, 39], and the Chung-Lu random graph model [14, 15]. There is little consensus about which generative model is the "right" one. For example, already in 2006, the survey by Chakrabarti and Faloutsos [12] compares 23 different probabilistic models of social networks, and multiple new such models are proposed every year.

Generative models articulate a hypothesis about what "real-world" social networks look like, how they are created, and how they will evolve in the future. They are directly useful for generating synthetic data and can also be used as a proxy to study the effect of random processes on a network [3, 41, 43]. However, the plethora of models presents a quandary for the design of algorithms for social networks with rigorous guarantees: which of these models should one tailor an algorithm to? One idea is to seek algorithms that are tailored to *none* of them, and to instead assume only determinstic combinatorial conditions that share the spirit of the prevailing generative models. This is the approach taken in this paper.

There is empirical evidence that many NP-hard optimization problems are often easier to solve in social networks than in worst-case graphs. For example, lightweight heuristics are unreasonably effective in practice for finding the maximum clique of a social network [52]. Similar success stories have been repeatedly reported for the problem of recovering dense subgraphs or communities in social networks [60, 54, 42, 59]. To define our notion of "social-network-like" graphs, we turn to one of the most agreed upon properties of social networks – *triadic closure*, the property that when two members of a social network have a friend in common, they are likely to be friends themselves.

1.1 Properties of social networks

There is wide consensus that social networks have relatively predictable structure and features, and accordingly are not well modeled by arbitrary graphs. From a structural viewpoint, the most well studied and empirically validated statistical properties of social networks include heavy-tailed degree distributions [6, 11, 23], a high density of triangles [64, 53, 63] and other dense subgraphs or "communities" [26, 32, 46, 47, 40], low diameter and the small world property [35, 36, 37, 45], and triadic closure [53, 63, 56].

For the problem of finding cliques in networks, it does not help to assume that the graph has small diameter (every network can be rendered small-diameter by adding one extra vertex connected to all other vertices). Similarly, merely assuming a power-law degree distribution does not seem to make the clique problem easier [24]. On the other hand, as we show, the clique problem is tractable on graphs with strong triadic closure properties.

1.2 Our model: c-closed graphs

Motivated by the empirical evidence for triadic closure in social networks, we define the class of c-closed graphs. Figure 1 shows the triadic closure of the network of email communications at Enron [1] and other social networks have been shown to behave similarly [8]. In particular, the more common neighbors two vertices have, the more likely they are to be adjacent to each other. The definition of c-closed graphs is a coarse version of this property: we assert that *every* pair of vertices with c or more common neighbors must be adjacent to each other.

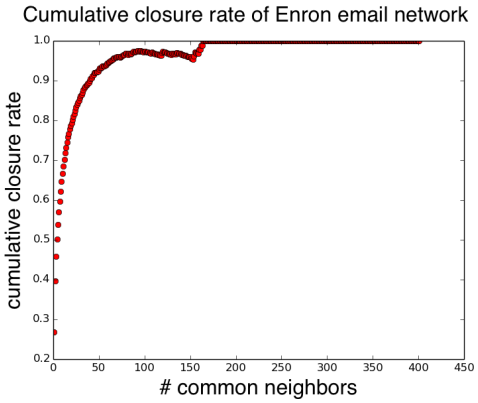


Figure 1 Triadic closure properties of the Enron email graph (36K nodes and 183K edges). Nodes of this network are Enron employees, and there is an edge connecting two employees if one sent at least one email to the other. Given an x value, the y-axis shows the cumulative closure rate: the fraction of pairs of nodes with *at least* x common neighbors that are themselves connected by an edge.

▶ **Definition 1** (c-closed graph)**.** For a positive integer c, an undirected graph $G = (V, E)$ is *c-closed* if, whenever two distinct vertices $u, v \in V$ have at least c common neighbors, (u, v) is an edge of G.

The parameter c interpolates between a disjoint unions of cliques (when $c = 1$) and all graphs (when $c = |V| - 1$). The class of 2-closed graphs is already non-trivial. These are exactly the graphs that do not contain $K_{2,2}$ or a diamond (K_4 minus an edge) as an induced subgraph. For example, graphs with girth at least 5, e.g. constant-degree expanders, are 2-closed. For every c, membership in the class of c-closed graphs can be checked by squaring the adjacency matrix in $O(n^\omega)$ time, where $\omega < 2.373$ is the matrix multiplication exponent.

While the definition of c-closed captures important aspects of triadic closure, it is fragile in the sense that a single pair of non-adjacent vertices with many common neighbors prevents the graph from being c-closed for a low value of c. To address this, we define the more robust notion of *weakly c-closed* graphs and show that our results carry over to these graphs. Well-studied social networks with thousands of vertices are typically weakly c-closed for modest values of c (see the full version [29] for experimental results).

▶ **Definition 2.** Given a graph and a value of c, a *bad pair* is a non-adjacent pair of vertices with at least c common neighbors.

▶ **Definition 3** (Weakly c-closed graph)**.** A graph is *weakly c-closed* if there exists an ordering of the vertices $\{v_1, v_2, \ldots, v_n\}$ such that for all i, v_i is in no bad pairs in the graph induced by $\{v_i, v_{i+1}, \ldots, v_n\}$.

A graph can be c-closed only for large c but weakly c-closed for much smaller c. Consider the graph G that is a clique of size k with one edge (u, v) missing. G is not c-closed for any $c < k - 2$. The only bad pair in G is (u, v). The vertex ordering that places u and v at the end demonstrates that G is weakly 1-closed. Also, the properties of being c-closed and weakly c-closed are hereditary, meaning that they are closed under taking induced subgraphs. We will use this basic fact often.

1.3 Our contributions

One can study a number of computational problems on c-closed (and weakly c-closed) graphs. We focus on the problem of enumerating all maximal cliques, an important problem in social network analysis [16, 58, 18, 21, 57]. We study *fixed-parameter tractability*[4] with respect to c. There is a rich literature on fixed-parameter tractability for other graph parameters including treewidth, arboricity, and the size of the output [17].

In a graph G, a *clique* is a subgraph of G in which there is an edge between every pair of vertices. A *maximal* clique in G is a clique that cannot be made any larger by the addition of some other vertex in G. In any graph, all maximal cliques can be listed in $O(mn)$ time per maximal clique [61]. We focus on the following two problems:

1. determining the maximum possible number of maximal cliques in a c-closed graph on n vertices.
2. finding algorithms to enumerate all maximal cliques in c-closed graphs (that run faster than $O(mn)$ time per maximal clique).

Our main result is that for constant c the number of maximal cliques in a c-closed graph on n vertices is $O(n^{2-2^{1-c}})$. More specifically, we prove the following bound.

▶ **Theorem 4.** *Any c-closed graph on n vertices has at most* $\min\{3^{(c-1)/3}n^2, 4^{(c+4)(c-1)/2}n^{2-2^{1-c}}\}$ *maximal cliques.*

For example, 3-closed, 4-closed, and 5-closed graphs have $O(n^{3/2})$, $O(n^{7/4})$, and $O(n^{15/8})$ maximal cliques respectively.

The proof of the first bound listed in Theorem 4 extends to weakly c-closed graphs, giving the following result.

▶ **Theorem 5.** *Any weakly c-closed graph on n vertices has at most $3^{(c-1)/3}n^2$ maximal cliques.*

See the full version [29] for experimental results showing that well-studied social networks are weakly c-closed for modest values of c. Note that Theorem 5 is exponential in the even smaller value of $(c-1)/3$.

Since in any graph all maximal cliques can be listed in $O(mn)$ time per maximal clique, Theorem 4 proves that listing all maximal cliques in a c-closed graph is fixed-parameter tractable (i.e. has running time $f(c)n^\alpha$ for constant α). We give an algorithm for listing all maximal cliques in a c-closed graph that runs faster than applying the $O(mn)$-per-clique algorithm as a black box. Our algorithm follows naturally from the proof of Theorem 4 and gives the following theorem, where $p(n, c)$ denotes the time to list all wedges (induced 2-paths) in a c-closed graph on n vertices. A result of Gąsieniec, Kowaluk, and Lingas [31] implies that $p(n, c) = O(n^{2+o(1)}c + c^{(3-\omega-\alpha)/(1-\alpha)}n^\omega + n^\omega \log n)$ where ω is the matrix multiplication exponent and $\alpha > 0.29$.

▶ **Theorem 6.** *In any c-closed graph, a set of cliques containing all maximal cliques can be generated in time $O(p(n, c) + 3^{c/3}n^2)$. The exact set of all maximal cliques in any c-closed graph can be generated in time $O(p(n, c) + 3^{c/3}2^c cn^2)$.*

Non-trivial lower bounds for the number of maximal cliques in a c-closed graph were previously known only for extreme values of c. A 2-closed graph can have $n^{3/2}$ maximal

[4] A problem is said to be fixed-parameter tractable with respect to a parameter k if there is an algorithm that solves it in time at most $f(k)n^\alpha$ where f can be an arbitrary function but α is a constant.

cliques [22]. The classic Moon-Moser graph (with additional isolated vertices) is $(n-2)$-closed and has $3^{\lfloor n/3 \rfloor}$ maximal cliques [44]. This graph consists of the complete multipartite graph with $\lfloor n/3 \rfloor$ parts of size 3, and possibly additional isolated vertices. By taking a disjoint union of $n/(c+2)$ Moon-Moser graphs on $(c+2)$ vertices, we can construct a c-closed graph on n vertices with $\Omega(c^{-1}3^{c/3}n)$ maximal cliques for all $n \geq c$. We give improved lower bounds for intermediate values of c.

▶ **Theorem 7.** *For any positive integer c, there are c-closed graphs with n vertices and $\Omega(c^{-3/2}2^{c/2}n^{3/2})$ maximal cliques.*

It is an open problem to determine the exact exponent of n (between $3/2$ and $2 - 2^{1-c}$) in the expression for the maximum number of maximal cliques in a c-closed graph.

1.4 Related work

There are only a few algorithmic results for graph classes motivated by social networks. Although a number of NP-hard problems remain NP-hard on graphs with a power-law degree distribution [25], several problems in P have been shown to be easier on such graphs. Brach, Cygan, Lacki, and Sankowski [10] give faster algorithms for transitive closure, maximum matching, determinant, PageRank and matrix inverse. Borassi, Crescenzi, and Trevisan [9] assume several axioms satisfied by real-world graphs, one being a power-law degree distribution, and give faster algorithms for diameter, radius, distance oracles, and computing the most "central" vertices. Motivated by triadic closure, Gupta, Roughgarden, and Seshadhri [33] define *triangle-dense* graphs and prove relevant structural results. Intuitively, they prove that if a constant fraction of two-hop paths are closed into triangles, then the graph must contain many dense clusters.

For general graphs, Moon and Moser prove that the maximum possible number of maximal cliques in a graph on n vertices is $3^{n/3}$ (realized by a complete $n/3$-partite graph) [44]. Tomita, Tanaka, and Takahashi prove that the time to generate all maximal cliques in any n-vertex graph is also $O(3^{n/3})$ [58].

The clique problem has been studied on 2-closed graphs (under a different name). Eschen, Hoang, Spinrad, and Sritharan [22] show that the maximum number of maximal cliques in a 2-closed graph is $O(n^{3/2})$. They also show a matching lower bound via a projective planes construction. Suppose $n = p^2 + p + 1$ for a positive integer p and consider a finite projective plane on n points (and hence with n lines, see e.g. [2]). Let G denote the bipartite graph representing the point-line incidence matrix. The defining properties of finite projective planes imply that no two vertices have two common neighbors, so the 2-closed condition is vacuously satisfied. Every vertex of G has degree $p+1$, so the graph has $\Theta(n^{3/2})$ edges, each a maximal clique.

The clique problem has also been studied on other special classes of graphs such as graphs embeddable on a surface [19] and graphs of bounded degeneracy [20]. Degeneracy is a measure of everywhere sparsity. More formally, the degeneracy of a graph G is the smallest value d such that every nonempty subgraph of G contains a vertex of degree at most d. Eppstein et al. show that the maximum number of maximal cliques in a graph of degeneracy d is $O(n3^{d/3})$. The degeneracy of a graph, however, can be much larger than its c-closure. For example, the degeneracy of a graph is at least the size of a maximum clique, while even in 1-closed graphs, the size of the maximum clique can be arbitrarily large.

Clique counting is a classical problem in extremal combinatorics. One fundamental question is to count the *minimum* number of cliques in graphs with fixed number of edges i.e. to show that graphs with few cliques must have few edges. This simple question turns

out to be a complex problem, and is settled for K_3 by Razborov [50] by flag algebra, K_4 by Nikiforov [48] by a combination of combinatorics and analytical arguments, and all K_t by Reiher [51] by generalizing the argument of flag algebra to all sizes of cliques.

There has also been a long line of work in combinatorics on counting (not necessarily maximal) cliques in graphs with certain excluded subgraphs, subdivisions, or minors. Most recently, Fox and Wei give an asymptotically tight bound on the maximum number of cliques in graphs with forbidden minors [28], and an upper bound on the maximum number of cliques in graphs with forbidden subdivisions or immersions [27].

Many problems in combinatorics can be phrased as counting the number of cliques or independent sets in a (hyper)graph. For example, the problems of finding the volume of the metric polytope and counting the number of n-vertex H-free graphs (for any fixed subgraph H) can be translated into clique counting problems. The recently developed "container method" [5, 55] is a powerful tool to bound the number of cliques in (hyper)graphs and can be used to tackle a great range of problems.

1.5 Organization

In Section 2 we prove the first bound listed in Theorem 4, state Theorem 5, and introduce the proof of Theorem 6. In Section 3 we prove the second bound listed in Theorem 4 (which has improved dependence on n). In Section 4 we prove Theorem 7.

See the full version [29] for the proof of Theorem 6.

1.6 Notation

All graphs $G(V, E)$ are simple, undirected and unweighted. For any $v \in V$, let $N(v)$ denote the neighborhood of v. When the current graph is ambiguous, $N_G(v)$ will denote the neighborhood of v in G. For any $S \subseteq V$, let $G[S]$ denote the subgraph of G induced by S.

2 Initial Bound and Algorithm

2.1 Bound on number of maximal cliques

In this section, we prove the following bound on the number of maximal cliques in a c-closed graph and show that this bound carries over to weakly c-closed graphs. Let $F(n, c)$ denote the maximum possible number of maximal cliques in a c-closed graph on n vertices. The following theorem uses a natural peeling process and obtain an initial upper bound on the number of maximal cliques. A more involved analysis, Theorem 11 which gives a tighter upper bound, is delayed to later.

▶ **Theorem 8** (restatement of part of Theorem 4). *For all positive integers c, n, we have $F(n, c) \leq 3^{(c-1)/3} n^2$.*

Proof. Let G be a c-closed graph on n vertices and let $v \in V(G)$ be an arbitrary vertex. Every maximal clique $K \subseteq G$ is of one of the following types:

1. The clique K does not contain vertex v; and K is maximal in $G \setminus \{v\}$.
2. The clique K contains vertex v; and $K \setminus \{v\}$ is maximal in $G \setminus \{v\}$.
3. The clique K contains vertex v; and $K \setminus \{v\}$ is not maximal in $G \setminus \{v\}$.

Bounding the number of maximal cliques of type 1 and 2 is straightforward because every such clique can be obtained by starting with a clique maximal in $G \setminus \{v\}$ and extending it to include vertex v if possible. Therefore, the number of maximal cliques of types 1 and 2 combined is at most $F(n-1, c)$.

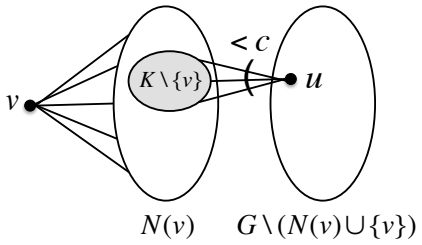


Figure 2 A maximal clique K of type 3: K contains vertex v and $K \setminus \{v\}$ is not maximal in $G \setminus \{v\}$. Property C asserts that there exists a vertex $u \notin N(v)$ whose neighborhood contains $K \setminus \{v\}$. Since G is c-closed, $|N(u) \cap N(v)| < c$.

Type 3 cliques are maximal in $N(v)$, but not in $G \setminus \{v\}$. We will prove that the number of maximal cliques of type 3 is at most $3^{(c-1)/3}n$, crucially using the c-closed property. Figure 2 shows a maximal clique K of type 3.

We claim that each type 3 maximal clique K satisfies the following three properties.

A) $K \setminus \{v\}$ is a clique in the neighborhood of v, and
B) $K \setminus \{v\}$ is not in the neighborhood of any other vertex in $N(v)$.
C) There exists a vertex $u \notin N(v)$ whose neighborhood contains $K \setminus \{v\}$.

Property A is clear since K is a clique containing v. Property B is true because if we can extend $K \setminus \{v\}$ to include some vertex $w \in N(v)$, then K can also be extended to include w which contradicts the fact that K is maximal. To see property C, note that since $K \setminus \{v\}$ is not a maximal clique in $G \setminus \{v\}$ we can extend the clique $K \setminus \{v\}$ to include some vertex in $G \setminus \{v\}$. By property B, we can extend $K \setminus \{v\}$ to include some vertex u not in $N(v)$.

Let u be as in property C. Then $K \setminus \{v\}$ must be a maximal clique in $G[N(v) \cap N(u)]$ because otherwise we could extend $K \setminus \{v\}$ to some other vertex in $N(v) \cap N(u)$, which contradicts property B.

Thus, the number of type 3 maximal cliques is at most

$$\sum_{u \in G \setminus (N(v) \cup \{v\})} F(|N(u) \cap N(v)|, c). \tag{1}$$

Since G is c-closed, $|N(u) \cap N(v)| < c$ for all vertices $u \notin N(v)$. Then since any k-vertex graph has at most $3^{k/3}$ maximal cliques [44],

$$F(|N(u) \cap N(v)|, c) \leq 3^{(c-1)/3}.$$

Thus, the number of type 3 maximal cliques in G is at most $3^{(c-1)/3}n$.

Counting all three types of maximal cliques, we have the following recursive inequality:

$$F(n, c) \leq F(n-1, c) + 3^{(c-1)/3}n.$$

By induction on n with the base case $F(1, c) = 1$, this gives

$$F(n, c) \leq 3^{(c-1)/3}\binom{n+1}{2} \leq 3^{(c-1)/3}n^2. \quad \blacktriangleleft$$

Note that v was chosen arbitrarily and the proof is valid as long as "$|N(u) \cap N(v)| < c$ for all vertices $u \notin N(v)$". Thus, in each recursive level, we only require the existence of a vertex v in no bad pairs. Equivalently, it suffices to have an ordering of the vertices $\{v_1, v_2, \ldots, v_n\}$ such that for all i, v_i is in no bad pairs in the graph induced by $\{v_i, v_{i+1}, \ldots, v_n\}$. This is exactly the definition of a weakly c-closed graph. Thus, we get the following theorem.

▶ **Theorem 9** (Restatement of Theorem 5). *For any positive integers c, n, there are at most $3^{(c-1)/3}n^2$ maximal cliques in an n-vertex weakly c-closed graph.*

2.2 Algorithm to generate all maximal cliques

Recall that $p(n, c)$ denotes the time to list all wedges (induced 2-paths) in a c-closed graph on n vertices. A result of Gąsieniec, Kowaluk, and Lingas [31] implies that $p(n, c) = O(n^{2+o(1)}c + c^{(3-\omega-\alpha)/(1-\alpha)}n^\omega + n^\omega \log n)$ where ω is the matrix multiplication exponent and $\alpha > 0.29$.

▶ **Theorem 10** (restatement of Theorem 6). *A superset of the maximal cliques in any c-closed graph can be generated in time $O(p(n, c) + 3^{c/3}n^2)$. The exact set of all maximal cliques in any c-closed graph can be generated in time $O(p(n, c) + 3^{c/3}2^c cn^2)$.*

The algorithm follows naturally from the proof of Theorem 8 with two additional ingredients:

- A preprocessing step to enumerate all wedges in the graph speeds up the later process of finding the intersection of the neighborhoods of two vertices (i.e. $N(u) \cap N(v)$ from the proof of Theorem 8).
- An algorithm of Tomita, Tanaka, and Takahashi [58] generates all maximal cliques in any n-vertex graph in time $O(3^{n/3})$. We apply this to the recursive calls on the small induced subgraphs $G[N(u) \cap N(v)]$, which have less than c vertices, that arise in handling the type 3 cliques in the proof of Theorem 8.

We defer the full algorithm description and runtime analysis to the full version [29].

3 Improved Bound

Recall that $F(n, c)$ is the maximum number of maximal cliques in a c-closed graph on n vertices.

▶ **Theorem 11** (restatement of part of Theorem 4). *For all positive integers c, n, we have $F(n, c) \leq 4^{(c+4)(c-1)/2}n^{2-2^{1-c}}$.*

The structure of the proof is similar to that of Theorem 8. We get an improved bound by a separate analysis depending on whether G has a vertex of "high" degree. This idea appears in the result of Eschen et al. [22], who prove the result for the $c = 2$ case.

We will require the following simple lemma.

▶ **Lemma 12.** *For any v, $G[N(v)]$ is a $(c-1)$-closed graph.*

Proof. Consider pair $x, y \in N(v)$ with $c-1$ common neighbors in $G[N(v)]$. Since vertex $v \notin N(v)$ is also a common neighbor of x and y, x and y have c common neighbors in G. Thus, (x, y) is an edge. ◀

Proof of Theorem 11. Let $G = (V, E)$ be a c-closed graph on n vertices with $F(n, c)$ maximal cliques. Let $\Delta(G)$ be the maximum degree of G.

Case 1: $\Delta(G) \leq n^{1/2}$.

By Lemma 12 for all $v \in V(G)$, $G[N(v)]$ is $(c-1)$-closed. Then, since the number of maximal cliques containing v is exactly the number of maximal cliques in $G[N(v)]$, we have

$$F(n, c) \leq nF(\Delta(G), c-1).$$

Case 2: $\Delta(G) > n^{1/2}$.

Let $v \in V$ be a vertex of degree $\Delta(G)$. We will count the maximal cliques containing at least one vertex in $N(v) \cup \{v\}$, delete $N(v) \cup \{v\}$, and recurse.

Since the number of maximal cliques containing v is exactly the number of maximal cliques in $G[N(v)]$, v is in at most $F(\Delta(G), c-1)$ maximal cliques. It remains to bound the number the maximal cliques that contain some vertex in $N(v)$ but not v itself. Such a clique must contain some vertex in $u \in V \setminus (N(v) \cup \{v\})$ (otherwise, it would not be maximal). Let $\mathcal{K}$ be the set of such cliques. We will bound $|\mathcal{K}|$ by grouping the maximal cliques K in $\mathcal{K}$ based on which vertices of $N(v)$ are in K. For nonempty $S \subseteq N(v)$, let $N(S)$ denote $\bigcap_{u \in S} N(u)$. Also, let $N_2(v)$ denote the set of vertices of distance *exactly* 2 from v. Let us bound the number of cliques $K \in \mathcal{K}$ such that $K \cap N(v) = S$. The other vertices in K must be in $N(S) \cap N_2(v)$. By Lemma 12, $G[N(S) \cap N_2(v)]$ is a $(c-1)$-closed graph. The number of cliques $K \in \mathcal{K}$ such that $K \cap N(v) = S$ is at most $F(|N(S) \cap N_2(v)|, c-1)$. Summing over all subsets $S \subseteq N(v)$, we have

$$|\mathcal{K}| \leq \sum_{S \subseteq N(v)} F(|N(S) \cap N_2(v)|, c-1). \tag{2}$$

For all $u \in N_2(v)$, since u and v are not adjacent, $|N(u) \cap N(v)| < c$ (because G is c-closed). Each vertex in $N_2(v)$ can be in $N(S)$ for only 2^{c-1} sets $S \subseteq N(v)$, implying

$$\sum_{S \subseteq N(v)} |N(S) \cap N_2(v)| \leq |N_2(v)| 2^{c-1} \leq \min\{(\Delta(G))^2, n\} 2^{c-1}. \tag{3}$$

We want to determine for all S the value of $|N(S) \cap N_2(v)|$ that maximizes the upper bound for $|\mathcal{K}|$ in Inequality (2) subject to the constraint in Inequality (3). Later, we prove our bound on F (from the theorem statement) by induction on n and c. In fact, we show that $F(n, c)$ is bounded by

$$F_0(n, c) = 4^{(c+4)(c-1)/2} n^{2-2^{1-c}},$$

the desired upper bound for $F(n, c)$ that we are trying to prove by induction. Since $F_0(n, c)$ is convex in n, by the inductive hypothesis we can apply Jenson's inequality on Inequality (2). Jenson's inequality implies that the upper bound on $|\mathcal{K}|$ is maximized by setting $|N(S) \cap N_2(v)|$ to be as large as possible (note that it cannot exceed $\Delta(G)$) for as many S as possible until the bound in Inequality (3) is met and setting the rest to be 0. By Inequality (3), the number of non-zero terms $|N(S) \cap N_2(v)|$ we sum over is at most $\Delta(G)^{-1} \min\{(\Delta(G))^2, n\} 2^{c-1} \leq \min\{\Delta(G), \frac{n}{\Delta(G)}\} 2^{c-1}$. Thus, we have the following continuation of Inequality (2).

$$|\mathcal{K}| \leq \sum_{S \subseteq N(v)} F(|N(S) \cap N_2(v)|, c-1) \leq \min\{\Delta(G), \frac{n}{\Delta(G)}\} 2^{c-1} F_0(\Delta(G), c-1). \tag{4}$$

Recall $|\mathcal{K}|$ is the number of maximal cliques that contain some vertex in $N(v)$ but not v itself, so we combine Inequality (4) with the observation (from the beginning of case 2) that v is in at most $F(\Delta(G), c-1)$ maximal cliques to conclude that the number of maximal cliques containing at least one vertex in $N(v) \cup \{v\}$ is at most

$$F_0(\Delta(G), c-1)(1 + \min\{\Delta(G), \frac{n}{\Delta(G)}\} 2^{c-1}) \leq F_0(\Delta(G), c-1) \min\{\Delta(G), \frac{n}{\Delta(G)}\} 2^c.$$

Then, recursing on $G \setminus (N(v) \cup \{v\})$, we have:

$$F(n, c) < F_0(\Delta(G), c-1) \min\{\Delta(G), \frac{n}{\Delta(G)}\} 2^c + F(n - \Delta(G), c).$$

Combining the low and high degree bounds on $F(n,c)$, we get the following recurrence.

$$F(n,1) = n, \quad F(1,c) = 1$$
$$F(n,c) \leq \begin{cases} nF(\Delta(G), c-1) & \Delta(G) \leq n^{1/2} \\ F_0(\Delta(G), c-1)\frac{n}{\Delta(G)}2^c + F(n-\Delta(G), c) & \Delta(G) > n^{1/2} \end{cases}$$

The remainder of the proof shows inductively that the recurrence implies the desired bound $F(n,c) \leq F_0(n,c) \leq n^{2-2^{1-c}}2^{(c+4)(c-1)/2}$. The desired bound holds in the two base cases. For the inductive case, we need to show that

$$n^{2-2^{1-c}}2^{(c+4)(c-1)/2} \geq \begin{cases} n\Delta(G)^{2-2^{2-c}}2^{(c+3)(c-2)/2} & \Delta(G) \leq n^{1/2} \\ \Delta(G)^{2-2^{2-c}}2^{(c+3)(c-2)/2}\frac{n}{\Delta(G)}2^c & \\ \quad +(n-\Delta(G))^{2-2^{1-c}}2^{(c+4)(c-1)/2} & \Delta(G) > n^{1/2}. \end{cases}$$

In the $\Delta(G) \leq n^{1/2}$ case, the expression is maximized when $\Delta(G) = n^{1/2}$. Thus,

$$n\Delta(G)^{2-2^{2-c}}2^{(c+3)(c-3)/2} \leq n^{1+\frac{1}{2}(2-2^{2-c})}2^{(c+3)(c-2)/2} < n^{2-2^{1-c}}2^{(c+4)(c-1)/2},$$

as desired.

For the $\Delta(G) > n^{1/2}$ case, the second term of the expression can be written as $(n(1-\frac{\Delta(G)}{n}))^{2-2^{1-c}}2^{(c+4)(c-1)/2}$. We use the following claim.

▶ **Claim 13.** *$(1-x)^k \leq 1 - \frac{xk}{2}$ for any $0 < x \leq 1/2$ and $1 \leq k \leq 2$*

Proof. For any $y \in (0,1)$, $(1-y) \leq e^{-y}$ and $e^{-y} \leq 1-y/2$. Thus, $(1-x)^k \leq e^{-xk} \leq 1-xk/2$. ◀

Applying the claim with $x = \frac{\Delta(G)}{n}$ and $k = 2-2^{1-c}$, it suffices to show that

$$n^{2-2^{1-c}}2^{(c+4)(c-1)/2} \geq \Delta(G)^{2-2^{2-c}}2^{(c+3)(c-2)/2}\frac{n}{\Delta(G)}2^c + n^{2-2^{1-c}}\left(1 - \frac{\Delta(G)(2-2^{1-c})}{2n}\right)2^{(c+4)(c-1)/2}.$$

or equivalently that

$$\Delta(G)^{2-2^{2-c}}2^{(c+3)(c-2)/2}\frac{n}{\Delta(G)}2^c - n^{2-2^{1-c}}\frac{\Delta(G)(2-2^{1-c})}{2n}2^{(c+4)(c-1)/2} \leq 0.$$

Simplifying the left-hand side of the above inequality and using the fact that $c \geq 1$:

$$\begin{aligned} &\Delta(G)^{2-2^{2-c}}2^{(c+3)(c-2)/2}\frac{n}{\Delta(G)}2^c - n^{2-2^{1-c}}\frac{\Delta(G)(2-2^{1-c})}{2n})2^{(c+4)(c-1)/2} \\ &= n\Delta(G)^{1-2^{2-c}}2^{(c^2+c-6)/2} - n^{1-2^{1-c}}\Delta(G)(1-2^{-c})2^{(c^2+3c-4)/2} \\ &\leq 2^{(c^2+c-6)/2}(n\Delta(G)^{1-2^{2-c}} - n^{1-2^{1-c}}\Delta(G)(1-2^{-c})2^{c+1}) \\ &= 2^{(c^2+c-6)/2}(n\Delta(G)^{1-2^{2-c}} - n^{1-2^{1-c}}\Delta(G)(2^{c+1}-2)) \\ &\leq 2^{(c^2+c-6)/2}(n\Delta(G)^{1-2^{2-c}} - n^{1-2^{1-c}}\Delta(G)) \\ &= 2^{(c^2+c-6)/2}n^{1-2^{1-c}}\Delta(G)^{1-2^{2-c}}(n^{2^{1-c}} - \Delta(G)^{2^{2-c}}) \leq 0. \end{aligned}$$

The last inequality holds because $\Delta \geq n^{1/2}$. ◀

Like the proof of the initial bound (Theorem 8), the proof of the improved bound (Theorem 11) also suggests an algorithm for generating the set of maximal cliques involving the preprocessing step of listing the set of all wedges in the graph. However, this algorithm is not asymptotically faster than the algorithm from Theorem 10 since its dependence on n still includes $p(n,c)$ and we omit it.

4 Lower bound

▶ **Theorem 14** (restatement of Theorem 7). *For any positive integer c, we can construct graphs which are c-closed and with $\Omega(c^{-3/2}2^{c/2}n^{3/2})$ maximal cliques.*

Construction

We suppose that c is even and n is a multiple of c. We can do this with only an absolute constant factor loss in the bound, which is allowable. We start with a graph H on $v = 2n/c$ vertices with girth 5 and the maximum possible number of edges, which is $\Omega(v^{3/2})$ [30].

We construct our c-closed graph G on n vertices from H in the following way. For each vertex $x \in V(H)$, we replace it with a vertex set U_x with $c/2$ vertices. Therefore, there are $|V(H)| \cdot c/2 = n$ vertices in G. The adjacency relation of G is as follows.

- Add all edges within each U_x so that U_x is a clique for all $x \in V(H)$.
- For any edge (x, y) of H, we place edges between the vertex sets U_x and U_y such that the bipartite graph between U_x and U_y consists of a complete bipartite graph minus a perfect matching.
- For any distinct and nonadjacent $x, y \in V(H)$, there are no edges between U_x and U_y.

Theorem 14 follows from the next two claims.

▶ **Claim 15.** *The graph G constructed is c-closed.*

Proof. It suffices to check that for any two non-adjacent vertices in G, they have at most $c - 1$ common neighbors. By the construction, there are only two types of non-adjacent vertices:

Case 1: The non-adjacent pair $u, v \in V(G)$ are such that $u \in U_x, v \in U_y$ and $x, y \in V(H)$ are disitinct and non-adjacent in H.

In this case, there are no edges between U_x, U_y, and the common neighbors of u, v are such that there is a vertex $z \in V(H)$ such that $(x, z), (y, z)$ are both edges in H. Since H has girth 5, there is at most one such $z \in H$ given x, y, as otherwise H would contain a C_4. Vertex $u \in V(G)$ is adjacent to exactly $|U_z| - 1 = (c/2) - 1$ vertices in U_z, so u, v can have at most $(c/2) - 1$ common neighbors.

Case 2: The non-adjacent pair $u, v \in V(G)$ is such that $u \in U_x, v \in U_y$ and $x, y \in V(H)$ are adjacent in H.

In this case, u and v are adjacent to all other vertices in $U_x \cup U_y$, so they have $c - 2$ common neighbors in $U_x \cup U_y$. Suppose for contradiction that u, v have some other common neighbor w and $w \in U_z$ for some $z \neq x, y$. This implies that $(w, x), (w, y)$ are both edges in H. However, (x, y) is already an edge in H by the assumption of this case. This implies that H contains a triangle, which contradicts the fact that H has girth 5.

Combining both cases, we know that G is c-closed. ◀

▶ **Claim 16.** *There are $\Omega(c^{-3/2}2^{c/2}n^{3/2})$ maximal cliques in G.*

Proof. For any edge (x, y) of H, picking one endpoint of each non-edge in $U_x \cup U_y$ gives a maximal clique. Thus for each edge (x, y) of H, there are exactly $2^{|U_x|} = 2^{c/2}$ maximal cliques.

There are $\Omega(|V(H)|^{3/2}) = \Omega((2n/c)^{3/2})$ edges in H. As each of the maximal cliques obtained are distinct, we obtain $\Omega(2^{c/2} \cdot (2n/c)^{3/2}) = \Omega(c^{-3/2}2^{c/2}n^{3/2})$ maximal cliques. ◀

5 Open problems and future directions

Direct improvement of our results

- Determine the exact dependence on n for the maximum possible number of of maximal cliques in a c-closed graph. We have proven (up to constant dependence on c) that this number is between $n^{3/2}$ and $n^{2-2^{1-c}}$.
- Find a faster algorithm for listing the set of all wedges (induced 2-paths) in a c-closed graph (this would improve the runtime of Algorithm from Theorem 10).

Further exploration of c-closed graphs

- Study the densest k-subgraph problem, a generalization of the clique problem, on c-closed graphs. The input to the problem is a graph G and a parameter k, and the goal is to to find the subgraph of G on k vertices with the most edges. Unlike the clique problem, densest k-subgraph is NP-hard even for 2-closed graphs (more specifically, for graphs of girth 6) [49]. For general graphs, the best-known approximation algorithm has approximation ratio roughly $O(n^{1/4})$ [7] and under certain average-case hardness assumptions (concerning the planted clique problem), constant-factor approximation algorithms do not exist [4].
- Determine which other NP-hard problems are fixed-parameter tractable with respect to c.
- Determine which problems in P have faster algorithms on c-closed graphs.

Other model-free definitions of social networks

- Explore other graph classes motivated by the well-established signatures of social networks (described in the introduction): heavy-tailed degree distributions, high triangle density, dense "communities", low diameter and the small world property, and triadic closure.
- Determine other model-free definitions of social networks, for example, those motivated by 4-vertex subgraph frequencies. Ugander et al. [62] and subsequently Seshadhri [34] computed 4-vertex subgraph counts in a variety of social networks and the frequencies observed are far different than what one would expect from a random graph. In particular, social networks tend to have far fewer induced 4-cycles than random graphs.

References

1 Enron email dataset. URL: `https://www.cs.cmu.edu/~./enron/`.

2 A. Abraham and R. Sandler. *An Introduction to Finite Projective Planes*. Dover, 2015.

3 R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406:378–382, 2000.

4 N. Alon, S. Arora, R. Manokaran, D. Moshkovitz, and O. Weinstein. Inapproximability of densest κ-subgraph from average case hardness. *Unpublished manuscript*, 2011.

5 J. Balogh, R. Morris, and W. Samotij. Independent sets in hypergraphs. *J. Amer. Math. Soc.*, 28(3):669–709, 2015.

6 A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999. `doi:10.1126/science.286.5439.509`.

7 A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph. In *Proceedings of the 2010 ACM Symposium on Theory of Computing*, pages 201–210. ACM, 2010.

8 M. Bloznelis and V. Kurauskas. Clustering function: a measure of social influence. *CoRR*, abs/1207.4941, 2012. URL: `http://arxiv.org/abs/1207.4941`, `arXiv:1207.4941`.

9 M. Borassi, P. Crescenzi, and L. Trevisan. An axiomatic and an average-case analysis of algorithms and heuristics for metric properties of graphs. *arXiv preprint arXiv:1604.01445*, 2016.

10 P. Brach, M. Cygan, J. Łącki, and P. Sankowski. Algorithmic complexity of power law networks. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1306–1325. SIAM, 2016.

11 A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer Networks*, 33:309–320, 2000.

12 D. Chakrabarti and C. Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys*, 38(1), 2006. doi:10.1145/1132952.1132954.

13 D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A recursive model for graph mining. In *SIAM Conference on Data Mining*, pages 442–446, 2004. URL: http://siam.org/proceedings/datamining/2004/dm04_043chakrabartid.pdf.

14 F. Chung and L. Lu. The average distances in random graphs with given expected degrees. *Proc. Natl. Acad. Sci. USA*, 99(25):15879–15882, 2002. doi:10.1073/pnas.252631999.

15 F. Chung and L. Lu. Connected components in random graphs with given degree sequences. *Ann. Comb.*, 6:125–145, 2002. doi:10.1007/PL00012580.

16 A. Conte, R. De Virgilio, Antonio Maccioni, M. Patrignani, and R. Torlone. Finding all maximal cliques in very large social networks. In *EDBT*, pages 173–184, 2016.

17 M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*, volume 3. Springer, 2015.

18 N. Du, B. Wu, L. Xu, B. Wang, and X. Pei. A parallel algorithm for enumerating all maximal cliques in complex network. In *Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on*, pages 320–324. IEEE, 2006.

19 V. Dujmović, G. Fijavž, G. Joret, T. Sulanke, and D. R. Wood. On the maximum number of cliques in a graph embedded in a surface. *European J. Combin.*, 32(8):1244–1252, 2011.

20 D. Eppstein, M. Löffler, and D. Strash. *Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time*, pages 403–414. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-17517-6_36.

21 D. Eppstein and D. Strash. Listing all maximal cliques in large sparse real-world graphs. *Experimental Algorithms*, pages 364–375, 2011.

22 E. M. Eschen, C. T. Hoàng, J. P. Spinrad, and R. Sritharan. On graphs without a C_4 or a diamond. *Discrete Appl. Math.*, 159(7):581–587, 2011.

23 M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of SIGCOMM*, pages 251–262, 1999.

24 A. Ferrante, G. Pandurangan, and K. Park. On the hardness of optimization in power law graphs. In *Proceedings of Conference on Computing and Combinatorics*, pages 417–427, 2006.

25 A. Ferrante, G. Pandurangan, and K. Park. On the hardness of optimization in power-law graphs. *Theoret. Comput. Sci.*, 393(1):220–230, 2008. doi:10.1016/j.tcs.2007.12.007.

26 S. Fortunato. Community detection in graphs. *Physics Reports*, 486:75–174, 2010.

27 J. Fox and F. Wei. On the number of cliques in graphs with a forbidden subdivision or immersion, 2016.

28 J. Fox and F. Wei. On the number of cliques in graphs with a forbidden minor. *J. Combin. Theory Ser. B*, 126:175–197, 2017. doi:10.1016/j.jctb.2017.04.004.

29 Jacob Fox, Tim Roughgarden, C. Seshadhri, Fan Wei, and Nicole Wein. Finding cliques in social networks: A new distribution-free model. *arXiv preprint arXiv:1804.07431*, 2018.

30 Z. Füredi and M. Simonovits. *The History of Degenerate (Bipartite) Extremal Graph Problems*, pages 169–264. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. doi:10.1007/978-3-642-39286-3_7.

31 L. Gąsieniec, M. Kowaluk, and A. Lingas. Faster multi-witnesses for boolean matrix multiplication. *Information Processing Letters*, 109(4):242–247, 2009.

32 M. Girvan and M. Newman. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA*, 99(12):7821–7826, 2002. doi:10.1073/pnas.122653799.

33 R. Gupta, T. Roughgarden, and C. Seshadhri. Decompositions of triangle-dense graphs. *SIAM J. Comput.*, 45(2):197–215, 2016.

34 M. Jha, C. Seshadhri, and A. Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *World Wide Web (WWW)*, pages 495–505, 2015.

35 J. M. Kleinberg. Navigation in a small world. *Nature*, 406:845, 2000.

36 J. M. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the Symposium on Theory of Computing*, pages 163–170, 2000.

37 J. M. Kleinberg. Small-world phenomena and the dynamics of information. In *Advances in Neural Information Processing Systems*, volume 1, pages 431–438, 2002.

38 R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *Proceedings of Foundations of Computer Science*, pages 57–65, 2000. doi:10.1109/SFCS.2000.892065.

39 J. Leskovec, D. Chakrabarti, J. M. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker graphs: An approach to modeling networks. *J. Mach. Learn. Res.*, 11:985–1042, 2010. URL: http://jmlr.csail.mit.edu/papers/v11/leskovec10a.html.

40 J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Math.*, 6(1):29–123, 2008.

41 H. Lin, C. Amanatidis, M. Sideri, R. M. Karp, and C. H. Papadimitriou. Linked decompositions of networks and the power of choice in Polya urns. In *Proceedings of the Symposium on Discrete Algorithms*, pages 993–1002, 2008.

42 M. Mitzenmacher, J. Pachocki, R. Peng, C. Tsourakakis, and S. Xu. Scalable large near-clique detection in large-scale networks via sampling. In *SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 815–824, 2015. doi:10.1145/2783258.2783385.

43 A. Montanari and A. Saberi. The spread of innovations in social networks. *Proc. Natl. Acad. Sci. USA*, 107(47):20196–20201, 2010.

44 J. Moon and L. Moser. On cliques in graphs. *Israel J. Math.*, 3(1):23–28, 1965.

45 M. E. J. Newman. The structure of scientific collaboration networks. *Proc. Natl. Acad. Sci. USA*, 98(2):404–409, 2001. doi:10.1073/pnas.98.2.404.

46 M. E. J. Newman. Properties of highly clustered networks. *Physical Review E*, 68(2):026121, 2003. doi:10.1103/PhysRevE.68.026121.

47 M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74(3):036104, 2006. doi:10.1103/PhysRevE.74.036104.

48 V. Nikiforov. The number of cliques in graphs of given order and size. *Trans. Amer. Math. Soc.*, 363(3):1599–1618, 2011.

49 V. Raman and S. Saket. Short cycles make W-hard problems hard: FPT algorithms for W-hard problems in graphs with no short cycles. *Algorithmica*, 52(2):203–225, 2008.

50 A. A. Razborov. On the minimal density of triangles in graphs. *Comb. Probab. Comput.*, 17(4):603–618, 2008. doi:10.1017/S0963548308009085.

51 C. Reiher. The clique density theorem. *Ann. of Math. (2)*, 184(3):683–707, 2016. doi:10.4007/annals.2016.184.3.1.

52 R. A. Rossi, D. F. Gleich, and A. H. Gebremedhin. Parallel maximum clique algorithms with applications to network analysis. *SIAM J. Sci. Comput.*, 37(5), 2015. doi:10.1137/14100018X.

53 A. Sala, L. Cao, C. Wilson, R. Zablit, H. Zheng, and B. Y. Zhao. Measurement-calibrated graph models for social network experiments. In *Proceedings of the World Wide Web Conference*, pages 861–870. ACM, 2010. doi:10.1145/1772690.1772778.

54 A. E. Sarıyüce, C. Seshadhri, A. Pınar, and Ü. V. Çatalyürek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 927–937, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee. URL: http://dl.acm.org/citation.cfm?id=2736277.2741640.

55 D. Saxton and A. Thomason. Hypergraph containers. *Invent. Math.*, 201(3):925–992, Sep 2015. doi:10.1007/s00222-014-0562-8.

56 C. Seshadhri, A. Pinar, and T. G. Kolda. Fast triangle counting through wedge sampling. In *Proceedings of the SIAM Conference on Data Mining*, 2013. URL: http://arxiv.org/abs/1202.5230.

57 E. Tomita and T. Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *J. Global Optim.*, 37(1):95–111, 2007.

58 E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.*, 363(1):28–42, 2006. doi:10.1016/j.tcs.2006.06.015.

59 C. Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 1122–1132, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee. URL: http://dl.acm.org/citation.cfm?id=2736277.2741098.

60 C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In *Proc. of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, 2013.

61 S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM J. Comput.*, 6(3):505–517, 1977.

62 J. Ugander, L. Backstrom, and J. Kleinberg. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *Proceedings of World Wide Web Conference*, pages 1307–1318, 2013.

63 J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*, 2011.

64 D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998. doi:10.1038/30918.

A PTAS for a Class of Stochastic Dynamic Programs

Hao Fu
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijng, China
fu-h13@mails.tsinghua.edu.cn

Jian Li
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijng, China
Corresponding author lijian83@mail.tsinghua.edu.cn

Pan Xu
Department of Computer Science, University of Maryland, College Park, USA
panxu@cs.umd.edu

Abstract

We develop a framework for obtaining polynomial time approximation schemes (PTAS) for a class of stochastic dynamic programs. Using our framework, we obtain the first PTAS for the following stochastic combinatorial optimization problems:

1. *Probemax* [19]: We are given a set of n items, each item $i \in [n]$ has a value X_i which is an independent random variable with a known (discrete) distribution π_i. We can *probe* a subset $P \subseteq [n]$ of items sequentially. Each time after probing an item i, we observe its value realization, which follows the distribution π_i. We can *adaptively* probe at most m items and each item can be probed at most once. The reward is the maximum among the m realized values. Our goal is to design an adaptive probing policy such that the expected value of the reward is maximized. To the best of our knowledge, the best known approximation ratio is $1 - 1/e$, due to Asadpour *et al.* [2]. We also obtain PTAS for some generalizations and variants of the problem.
2. *Committed Pandora's Box* [24, 22]: We are given a set of n boxes. For each box $i \in [n]$, the cost c_i is deterministic and the value X_i is an independent random variable with a known (discrete) distribution π_i. Opening a box i incurs a cost of c_i. We can adaptively choose to open the boxes (and observe their values) or stop. We want to maximize the expectation of the realized value of the last opened box minus the total opening cost.
3. *Stochastic Target* [15]: Given a predetermined target $\mathbb{T}$ and n items, we can adaptively insert the items into a knapsack and insert at most m items. Each item i has a value X_i which is an independent random variable with a known (discrete) distribution. Our goal is to design an adaptive policy such that the probability of the total values of all items inserted being larger than or equal to $\mathbb{T}$ is maximized. We provide the first bi-criteria PTAS for the problem.
4. *Stochastic Blackjack Knapsack* [16]: We are given a knapsack of capacity $\mathbb{C}$ and probability distributions of n independent random variables X_i. Each item $i \in [n]$ has a size X_i and a profit p_i. We can adaptively insert the items into a knapsack, as long as the capacity constraint is not violated. We want to maximize the expected total profit of all inserted items. If the capacity constraint is violated, we lose all the profit. We provide the first bi-criteria PTAS for the problem.

2012 ACM Subject Classification Theory of computation → Stochastic approximation

Keywords and phrases stochastic optimization, dynamic program, markov decision process, block policy, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.56

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 56; pp. 56:1–56:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version A full version of the paper is available at https://arxiv.org/abs/1805.07742.

Funding This research is supported in part by the National Basic Research Program of China Grant 2015CB358700, the National Natural Science Foundation of China Grant 61772297, 616320-16, 61761146003, and a grant from Microsoft Research Asia.

Acknowledgements We would like to thank Anupam Gupta for several helpful discussions during the various stages of the paper. Jian Li would like to thank the Simons Institute for the Theory of Computing, where part of this research was carried out. Hao Fu would like to thank Sahil Singla for useful discussions about Pandora's Box problem. Pan Xu would like to thank Aravind Srinivasan for his many useful comments.

1 Introduction

Consider an online stochastic optimization problem with a finite number of rounds. There are a set of tasks (or items, boxes, jobs or actions). In each round, we can choose a task and each task can be chosen at most once. We have an initial "state" of the system (called the value of the system). At each time period, we can select a task. Finishing the task generates some (possibly stochastic) feedback, including changing the value of the system and providing some profit for the round. Our goal is to design a strategy to maximize our total (expected) profit.

The above problem can be modeled as a class of stochastic dynamic programs which was introduced by Bellman [3]. There are many problems in stochastic combinatorial optimization which fit in this model, *e.g.,* the stochastic knapsack problem [9], the Probemax problem [19]. Formally, the problem is specified by a 5-tuple $(\mathcal{V}, \mathcal{A}, f, g, h, T)$. Here, $\mathcal{V}$ is the set of all possible values of the system. $\mathcal{A}$ is a finite set of items or tasks which can be selected and each item can be chosen at most once. This model proceeds for at most T rounds. At each round $t \in [T]$, we use $I_t \in \mathcal{V}$ to denote the current value of the system and $\mathcal{A}_t \subseteq \mathcal{A}$ the set of remaining available items. If we select an item $a_t \in \mathcal{A}_t$, the value of the system changes to $f(I_t, a_t)$. Here f may be stochastic and is assumed to be independent for each item $a_t \in \mathcal{A}$. Using the terminology from Markov decision processes, the state at time t is $s_t = (I_t, \mathcal{A}_t) \in \mathcal{V} \times 2^{\mathcal{A}}$. [1] Hence, if we select an item $a_t \in \mathcal{A}_t$, the evolution of the state is determined by the state transition function f:

$$s_{t+1} = (I_{t+1}, \mathcal{A}_{t+1}) = (f(I_t, a_t), \mathcal{A}_t \setminus a_t) \quad t = 1, \ldots, T. \tag{1}$$

Meanwhile the system yields a random profit $g(I_t, a_t)$. The function $h(I_{T+1})$ is the terminal profit function at the end of the process.

We begin with the initial state $s_1 = (I_1, \mathcal{A})$. We choose an item $a_1 \in \mathcal{A}$. Then the system yields a profit $g(I_1, a_1)$, and moves to the next state $s_2 = (I_2, \mathcal{A}_2)$ where I_2 follows the distribution $f(I_1, a_1)$ and $\mathcal{A}_2 = \mathcal{A} \setminus a_1$. This process is iterated yielding a random sequence

$$s_1, a_1, s_2, a_2, s_3, \ldots, a_T, s_{T+1}.$$

The profits are accumulated over T steps. [2] The goal is to find a policy that maximizes the

[1] This is why we do not call I_t the state of the system.

[2] If less than T steps, we can use some special items to fill which satisfy that $f(I, a) = I$ and $g(I, a) = 0$ for any value $I \in \mathcal{V}$.

expectation of the total profits $\mathbb{E}\Big[\sum_{t=1}^{T} g(I_t, a_t) + h(I_{T+1})\Big]$. Formally, we want to determine:

$$\mathrm{DP}^*(s_1) = \max_{\{a_1,\ldots,a_T\}\subseteq\mathcal{A}} \mathbb{E}\Big[\sum_{t=1}^{T} g(I_t, a_t) + h(I_{T+1})\Big] \qquad \text{(DP)}$$
$$\text{subject to: } I_{t+1} = f(I_t, a_t), \qquad t = 1, \ldots, T.$$

By Bellman's equation [3], for every initial state $s_1 = (I_1, \mathcal{A})$, the optimal value $\mathrm{DP}^*(s_1)$ is given by $\mathrm{DP}_1(I_1, \mathcal{A})$. Here DP_1 is the function defined by $\mathrm{DP}_{T+1}(I_{T+1}) = h(I_{T+1})$ together with the recursion:

$$\mathrm{DP}_t(I_t, \mathcal{A}_t) = \max_{a_t\in\mathcal{A}_t} \mathbb{E}\Big[\mathrm{DP}_{t+1}(f(I_t, a_t), \mathcal{A}_t \setminus a_t) + g(I_t, a_t)\Big], \quad t = 1, \ldots, T. \tag{2}$$

When the value and the item spaces are finite, and the expectations can be computed, this recursion yields an algorithm to compute the optimal value. However, since the state space $\mathcal{S} = \mathcal{V} \times 2^{\mathcal{A}}$ is exponentially large, this exact algorithm requires exponential time. Since this model can capture several stochastic optimization problems which are known (or believed) be #P-hard or even PSPACE-hard, we are interested in obtaining polynomial-time approximation algorithms with provable performance guarantees.

1.1 Our Results

In order to obtain a polynomial time approximation scheme (PTAS) for the stochastic dynamic program, we need the following assumptions.

▶ **Assumption 1.** *In this paper, we make the following assumptions.*

1. *The value space $\mathcal{V}$ is discrete and ordered, and its size $|\mathcal{V}|$ is a constant. W.l.o.g., we assume $\mathcal{V} = (0, 1, \ldots, |\mathcal{V}| - 1)$.*
2. *The function f satisfies that $f(I_t, a_t) \geq I_t$, which means the value is nondecreasing.*
3. *The function $h : \mathcal{V} \to R^{\geq 0}$ is a nonnegative function. The expected profit $\mathbb{E}[g(I_t, a_t)]$ is nonnegative (although the function $g(I_t, a_t)$ may be negative with nonzero probability).*

Assumption (1) seems to be quite restrictive. However, for several concrete problems where the value space is not of constant size (*e.g.,* Probemax in Section 1.2), we can discretize the value space and reduce its size to a constant, without losing much profit. Assumption (2) and (3) are quite natural for many problems. Now, we state our main result.

▶ **Theorem 1.** *For any fixed $\varepsilon > 0$, if Assumption 1 holds, we can find an adaptive policy in polynomial time $n^{2^{O(\varepsilon^{-3})}}$ with expected profit at least $\mathsf{OPT} - O(\varepsilon) \cdot \mathsf{MAX}$ where $\mathsf{MAX} = \max_{I\in\mathcal{V}} \mathrm{DP}_1(I, \mathcal{A})$ and OPT denotes the expected profit of the optimal adaptive policy.*

Our Approach: For the stochastic dynamic program, an optimal adaptive policy σ can be represented as a decision tree $\mathcal{T}$ (see Section 2 for more details). The decision tree corresponding to the optimal policy may be exponentially large and arbitrarily complicated. Hence, it is unlikely that one can even represent an optimal decision for the stochastic dynamic program in polynomial space. In order to reduce the space, we focus a special class of policies, called *block adaptive policy*. The idea of *block adptive policy* was first introduced by Bhalgat*et al.* [6] and further generalized in [17] to the context of the stochastic knapsack. To the best of our knowledge, the idea has not been extended to other applications. In this paper, we make use of the notion of block policy as well, but we target at the development

of a general framework. For this sake we provide a general model of block policy (see Section 3). Since we need to work with the more abstract dynamic program, our construction of block adaptive policy is somewhat different from that in [6, 17].

Roughly speaking, in a block adaptive policy, we take a batch of items simultaneously instead of a single one each time. This can significantly reduce the size of the decision tree. Moreover, we show that there exists a block-adaptive policy that approximates the optimal adaptive policy and has only a constant number of blocks on the decision tree (the constant depends on ε). Since the decision tree corresponding to a block adaptive policy has a constant number of nodes, the number of all topologies of the block decision tree is a constant. Fixing the topology of the decision tree corresponding to the block adaptive policy, we still need to decide the subset of items to place in each block. Again, there is exponential number of possible choices. For each block, we can define a *signature* for it, which allows us to represent a block using polynomially many possible signatures. The signatures are so defined such that two subsets with the same signature have approximately the same reward distribution. Finally, we show that we can enumerate the signatures of all blocks in polynomial time using dynamic programming and find a nearly optimal block-adaptive policy. The high level idea is somewhat similar to that in [17], but the details are again quite different.

1.2 Applications

Our framework can be used to obtain the first PTAS for the following problems.

1.2.1 The Probemax Problem

In the Probemax problem, we are given a set of n items. Each item $i \in [n]$ has a value X_i which is an independent random variable following a known (discrete) distribution π_i. We can *probe* a subset $P \subseteq [n]$ of items sequentially. Each time after *probing* an item i, we observe its value realization, which is an independent sample from the distribution π_i. We can *adaptively* probe at most m items and each item can be probed at most once. The reward is the maximum among the m realized values. Our goal is to design an adaptive probing policy such that the expected value of the reward is maximized.

Despite being a very basic stochastic optimization problem, we still do not have a complete understanding of the approximability of the Probemax problem. It is not even known whether it is intractable to obtain the optimal policy. For the non-adaptive Probemax problem (*i.e.,* the probed set P is just a priori fixed set), it is easy to obtain a $1 - 1/e$ approximation by noticing that $f(P) = \mathbb{E}[\max_{i \in P} X_i]$ is a submodular function (see e.g., Chen *et al.* [8]). Chen *et al.* [8] obtained the first PTAS. When considering the adaptive policies, Munagala [19] provided a $\frac{1}{8}$-approximation ratio algorithm by LP relaxation. His policy is essentially a non-adaptive policy (it is related to the contention resolution schemes [23, 10]). They also showed that the *adaptivity gap* (the gap between the optimal adaptive policy and optimal non-adaptive policy) is at most 3. For the Probemax problem, the best-known approximation ratio is $1 - \frac{1}{e}$. Indeed, this can be obtained using the algorithm for stochastic monotone submodular maximization in Asadpour *et al.* [2]. This is also a non-adaptive policy, which implies the adaptivity gap is at most $\frac{e}{e-1}$. In this paper, we provide the first PTAS, among all adaptive policies. Note that our policy is indeed adaptive.

▶ **Theorem 2.** *There exists a PTAS for the Probemax problem. In other words, for any fixed constant $\varepsilon > 0$, there is a polynomial-time approximation algorithm for the Probemax problem that finds a policy with the expected profit at least $(1-\varepsilon)\mathsf{OPT}$, where OPT denotes the expected profit of the optimal adaptive policy.*

Let the value I_t be the maximum among the realized values of the probed items at the time period t. Using our framework, we have the following system dynamics for Probemax:

$$I_{t+1} = f(I_t, i) = \max\{I_t, X_i\}, \quad g(I_t, i) = 0, \text{ and } h(I_{T+1}) = I_{T+1} \tag{3}$$

$t = 1, 2, \ldots, T$. Clearly, Assumption 1 (2) and (3) are satisfied. But Assumption 1 (1) is not satisfied because the value space $\mathcal{V}$ is not of constant size. We can discretize the value space and reduce its size to a constant. See full version for more details. If the reward is the summation of top-k values ($k = O(1)$) among the m realized values, we obtain the ProbeTop-k problem. Our techniques also allow us to derive the following result.

▶ **Theorem 3.** *For the ProbeTop-k problem where k is a constant, there is a polynomial time algorithm that finds an adaptive policy with the expected profit at least $(1-\varepsilon)\mathsf{OPT}$, where OPT denotes the expected profit of the optimal adaptive policy.*

1.2.2 Committed ProbeTop-k Problem

We are given a set of n items. Each item $i \in [n]$ has a value X_i which is an independent random variable with a known (discrete) distribution π_i. We can *adaptively* probe at most m items and choose k values in the committed model, where k is a constant. In the *committed* model, once we probe an item and observe its value realization, we must make an irrevocable decision whether to choose it or not, *i.e.,* we must either add it to the final chosen set C immediately or discard it forever. [3] If we add the item to the final chosen set C, the realized profit is collected. Otherwise, no profit is collected and we are going the probe the next item. Our goal is to design an adaptive probing policy such that the expected value $\mathbb{E}[\sum_{i\in C} X_i]$ is maximized, where C is the final chosen set.

▶ **Theorem 4.** *There is a polynomial time algorithm that finds a committed policy with the expected profit at least $(1-\varepsilon)\mathsf{OPT}$ for the committed ProbeTop-k problem, where OPT is the expected total profit obtained by the optimal policy.*

Let b_i^θ represent the action that we probe item i with the threshold θ (*i.e.,* we choose item i if X_i realizes to a value s such that $s \geq \theta$). Let I_t be the the number of items that have been chosen at the period time t. Using our framework, we have following transition dynamics for the ProbeTop-k problem.

$$I_{t+1} = f(I_t, b_i^\theta) = \begin{cases} I_t + 1 & \text{if } X_i \geq \theta, I_t < k, \\ I_t & \text{otherwise;} \end{cases} \quad g(I_t, b_i^\theta) = \begin{cases} X_i & \text{if } X_i \geq \theta, I_t < k, \\ 0 & \text{otherwise;} \end{cases} \tag{4}$$

for $t = 1, 2, \ldots, T$, and $h(I_{T+1}) = 0$. Since k is a constant, Assumption 1 is immediately satisfied. There is one extra requirement for the problem: in any realization path, we can choose at most one action b_i^θ from the set $\mathcal{B}_i = \{b_i^\theta\}_\theta$.

1.2.3 Committed Pandora's Box Problem

For Weitzman's "Pandora's box" problem [24], we are given n boxes. For each box $i \in [n]$, the probing cost c_i is deterministic and the value X_i is an independent random variable with a known (discrete) distribution π_i. Opening a box i incurs a cost of c_i. When we open the box i, its value is realized, which is a sample from the distribution π_i. The goal is to adaptively open

[3] In [10, 11], it is called the online decision model.

a subset $P \subseteq [n]$ to maximize the expected profit: $\mathbb{E}\left[\max_{i\in P}\{X_i\} - \sum_{i\in P} c_i\right]$. Weitzman provided an elegant optimal adaptive strategy, which can be computed in polynomial time. Recently, Singla [22] generalized this model to other combinatorial optimization problems such as matching, set cover and so on.

In this paper, we focus on the committed model, which is mentioned in Section 1.2.2. Again, we can *adaptively* open the boxes and choose at most k values in the committed way, where k is a constant. Our goal is to design an adaptive policy such that the expected value $\mathbb{E}\left[\sum_{i\in C} X_i - \sum_{i\in P} c_i\right]$ is maximized, where $C \subseteq P$ is the final chosen set and P is the set of opened boxes. Although the problem looks like a slight variant of Weitzman's original problem, it is quite unlikely that we can adapt Weitzman's argument (or any argument at all) to obtain an optimal policy in polynomial time. When $k = O(1)$, we provide the first PTAS for this problem. Note that a PTAS is not known previously even for $k = 1$.

▶ **Theorem 5.** *When $k = O(1)$, there is a polynomial time algorithm that finds a committed policy with the expected value at least $(1-\varepsilon)\mathsf{OPT}$ for the committed Pandora's Box problem.*

Similar to the committed ProbeTop-k problem, let b_i^θ represent the action that we open the box i with threshold θ. Let I_t be the number of boxes that have been chosen at the time period t. Using our framework, we have following system dynamics for the committed Pandora's Box problem:

$$I_{t+1} = f(I_t, b_i^\theta) = \begin{cases} I_t + 1 & \text{if } X_i \geq \theta, I_t < k, \\ I_t & \text{otherwise;} \end{cases} \quad g(I_t, b_i^\theta) = \begin{cases} X_i - c_i & \text{if } X_i \geq \theta, I_t < k, \\ -c_i & \text{otherwise;} \end{cases} \tag{5}$$

for $t = 1, 2, \cdots, T$, and $h(I_{T+1}) = 0$. Notice that we never take an action b_i^θ for a value $I_t < k$ if $\mathbb{E}[g(I_t, b_i^\theta)] = \Pr[X_t \geq \theta] \cdot \mathbb{E}[X_i \mid X_i \geq \theta] - c_i < 0$. Then Assumption 1 is immediately satisfied.

1.2.4 Stochastic Target Problem

İlhan *et al.* [15] introduced the following stochastic target problem. [4] In this problem, we are given a predetermined target $\mathbb{T}$ and a set of n items. Each item $i \in [n]$ has a value X_i which is an independent random variable with a known (discrete) distribution π_i. Once we decide to insert an item i into a knapsack, we observe a reward realization X_i which follows the distribution π_i. We can insert at most m items into the knapsack and our goal is to design an adaptive policy such that $\Pr[\sum_{i\in P} X_i \geq \mathbb{T}]$ is maximized, where $P \subseteq [n]$ is the set of inserted items. For the stochastic target problem, İlhan *et al.* [15] provided some heuristic based on dynamic programming for the special case where the random profit of each item follows a known normal distribution. In this paper, we provide an additive PTAS for the stochastic target problem when the target is relaxed to $(1-\varepsilon)\mathbb{T}$.

▶ **Theorem 6.** *There exists an additive PTAS for stochastic target problem if we relax the target to $(1-\varepsilon)\mathbb{T}$. In other words, for any given constant $\varepsilon > 0$, there is a polynomial-time approximation algorithm that finds a policy such that the probability of the total rewards exceeding $(1-\varepsilon)\mathbb{T}$ is at least $\mathsf{OPT} - \varepsilon$, where OPT is the resulting probability of an optimal adaptive policy.*

[4] [15] called the problem the adaptive stochastic knapsack instead. However, their problem is quite different from the stochastic knapsack problem studied in the theoretical computer science literature. So we use a different name.

Let the value I_t be the total profits of the items in the knapsack at time period t. Using our framework, we have following system dynamics for the stochastic target problem:

$$I_{t+1} = f(I_t, i) = I_t + X_i, \quad g(I_t, i) = 0, \text{ and } h(I_{T+1}) = \begin{cases} 1 & \text{if } I_{T+1} \geq \mathbb{T}, \\ 0 & \text{otherwise;} \end{cases} \tag{6}$$

for $t = 1, 2, \cdots, T$. Then Assumption 1 (2,3) is immediately satisfied. But Assumption 1 (1) is not satisfied for that the value space $\mathcal{V}$ is not of constant size. We can discretize the value space and reduce its size to a constant.

1.2.5 Stochastic Blackjack Knapsack

Levin *et al.* [16] introduced the *stochastic blackjack knapsack*. In this problem, we are given a capacity $\mathbb{C}$ and a set of n items, each item $i \in [n]$ has a size X_i which is an independent random variable with a known distribution π_i and a profit p_i. We can adaptively insert the items into a knapsack, as long as the capacity constraint is not violated. Our goal is to design an adaptive policy such that the expected total profits of all items inserted is maximized. The key feature here different from classic stochastic knapsack is that we gain zero if overflow, *i.e.,* we will lose the profits of all items inserted already if the total size is larger than the capacity. This extra restriction might induce us to take more conservative policies. Levin *et al.* [16] presented a non-adaptive policy with expected value that is at least $(\sqrt{2}-1)^2/2 \approx 1/11.66$ times the expected value of the optimal adaptive policy. Chen *et al.* [7] assumed each size X_i follows a known exponential distribution and gave an optimal policy for $n = 2$ based on dynamic programming. In this paper, we provide the first bi-criteria PTAS for the problem.

▶ **Theorem 7.** *For any fixed constant $\varepsilon > 0$, there is a polynomial-time approximation algorithm for stochastic blackjack knapsack that finds a policy with the expected profit at least $(1-\varepsilon)\mathsf{OPT}$, when the capacity is relaxed to $(1+\varepsilon)\mathbb{C}$, where OPT is the expected profit of the optimal adaptive policy.*

Denote $I_t = (I_{t,1}, I_{t,2})$ and let $I_{t,1}, I_{t,2}$ be the total sizes and total profits of the items in the knapsack at the time period t respectively. When we insert an item i into the knapsack and observe its size realization, say s_i, we define the system dynamics function to be

$$I_{t+1} = f(I_t, i) = (I_{t,1} + s_i, I_{t,2} + p_i), \quad h(I_{T+1}) = \begin{cases} I_{T+1,2} & \text{if } I_{T+1,1} \leq \mathbb{C}, \\ 0 & \text{otherwise;} \end{cases} \tag{7}$$

and $g(I_t, i) = 0$ for $t = 1, 2, \cdots, T$. Then Assumption 1 (2,3) is immediately satisfied. But Assumption 1 (1) is not satisfied for that the value space $\mathcal{V}$ is not of constant size. We can discretize the value space and reduce its size to a constant.

1.3 Related Work

Stochastic dynamic program has been widely studied in computer science and operation research (see, for example, [4, 20]) and has many applications in different fields. It is a natural model for decision making under uncertainty. In 1950s, Richard Bellman [3] introduced the "principle of optimality" which leads to dynamic programming algorithms for solving sequential stochastic optimization problems. However, Bellman's principle does not immediate lead to efficient algorithms for many problems due to "curse of dimensionality" and the large state space.

There are some constructive frameworks that provide approximation schemes for certain classes of stochastic dynamic programs. Shmoys *et al.* [21] dealt with stochastic linear programs. Halman *et al.* [12, 13, 14] studied stochastic discrete DPs with scalar state and action spaces and designed an FPTAS for their framework. As one of the applications, they used it to solve the stochastic ordered adaptive knapsack problem. As a comparison, in our model, the state space $\mathcal{S} = \mathcal{V} \times 2^{\mathcal{A}}$ is exponentially large and hence cannot be solved by previous framework.

Stochastic knapsack problem (SKP) is one of the most well-studied stochastic combinatorial optimization problem. We are given a knapsack of capacity $\mathbb{C}$. Each item $i \in [n]$ has a random value X_i with a known distribution π_i and a profit p_i. We can adaptively insert the items to the knapsack, as long as the capacity constraint is not violated. The goal is to maximize the expected total profit of all items inserted. For SKP, Dean *et al.* [9] first provide a constant factor approximation algorithm. Later, Bhalgat *et al.* [6] improved that ratio to $\frac{3}{8} - \varepsilon$ and gave an algorithm with ratio of $(1 - \varepsilon)$ by using ε extra budget for any given constant $\varepsilon \geq 0$. In that paper, the authors first introduced the notion of block adaptive policies, which is crucial for this paper. The best known single-criterion approximation factor is 2 [5, 17, 18].

The Probemax problem and ProbeTop-k problem are special cases of the general stochastic probing framework formulated by Gupta *et al.* [11]. They showed that the adaptivity gap of any stochastic probing problem where the outer constraint is prefix-closed and the inner constraint is an intersection of p matroids is at most $O(p^3 \log(np))$, where n is the number of items. The Bernoulli version of stochastic probing was introduced in [10], where each item $i \in U$ has a fixed value w_i and is "active" with an independent probability p_i. Gupta *et al.* [10] presented a framework which yields a $\frac{1}{4(k^{in}+k^{out})}$-approximation algorithm for the case when $\mathcal{I}_{in}$ and $\mathcal{I}_{out}$ are respectively an intersection of k^{in} and k^{out} matroids. This ratio was improved to $\frac{1}{(k^{in}+k^{out})}$ by Adamczyk *et al.* [1] using the iterative randomized rounding approach. Weitzman's Pandora's Box is a classical example in which the goal is to find out a single random variable to maximize the utility minus the probing cost. Singla [22] generalized this model to other combinatorial optimization problems such as matching, set cover, facility location, and obtained approximation algorithms.

2 Policies and Decision Trees

An instance of stochastic dynamic program is given by $\mathcal{J} = (\mathcal{V}, \mathcal{A}, f, g, h, T)$. For each item $a \in \mathcal{A}$ and values $I, J \in \mathcal{V}$, we denote $\Phi_a(I, J) := \Pr[f(I, a) = J]$ and $\mathcal{G}_a(I) := \mathbb{E}[g(I, a)]$. The process of applying a feasible adaptive *policy* σ can be represented as a decision tree $\mathcal{T}_\sigma$. Each node v on $\mathcal{T}_\sigma$ is labeled by a unique item $a_v \in \mathcal{A}$. Before selecting the item a_v, we denote the corresponding time index, the current value and the set of the remaining available items by t_v, I_v and $\mathcal{A}(v)$ respectively. Each node has several children, each corresponding to a different value realization (one possible $f(I_v, a_v)$). Let $e = (v, u)$ be the s-th edge emanating from $s \in \mathcal{V}$ where s is the realized value. We call u the s-child of v. Thus e has probability $\pi_e := \pi_{v,s} = \Phi_{a_v}(I_v, s)$ and weight $w_e := s$.

We use $\mathbb{P}(\sigma)$ to denote the expected profit that the policy σ can obtain. For each node v on $\mathcal{T}_\sigma$, we define $\mathcal{G}_v := \mathcal{G}_{a_v}(I_v)$. In order to clearly illustrate the tree structure, we add a dummy node at the end of each root-to-leaf path and set $\mathcal{G}_v = h(I_v)$ if v is a dummy node. Then, we recursively define the expected profit of the subtree $\mathcal{T}_v$ rooted at v to be

$$\mathbb{P}(v) = \mathcal{G}_v + \sum_{e=(v,u)} \pi_e \cdot \mathbb{P}(u), \tag{8}$$

if v is an internal node and $\mathbb{P}(v) = \mathcal{G}_v = h(I_v)$ if v is a leaf (*i.e.,* the dummy node). The expected profit $\mathbb{P}(\sigma)$ of the policy σ is simply $\mathbb{P}(\text{the root of } \mathcal{T}_\sigma)$. Then, according to Equation (2), we have

$$\mathbb{P}(v) \leq \mathrm{DP}_{t_v}(I_v, \mathcal{A}(v)) \leq \mathrm{DP}_1(I_v, \mathcal{A}) \leq \max_{I \in \mathcal{V}} \mathrm{DP}_1(I, \mathcal{A}) = \mathsf{MAX}$$

for each node v. For a node v, we say the path from the root to it on $\mathcal{T}_\sigma$ as the *realization path* of v, and denote it by $\mathcal{R}(v)$. We denote the probability of reaching v as $\Phi(v) = \Phi(\mathcal{R}(v)) = \prod_{e \in \mathcal{R}(v)} \pi_e$. Then, we have

$$\mathbb{P}(\sigma) = \sum_{v \in \mathcal{T}_\sigma} \Phi(v) \cdot \mathcal{G}_v. \tag{9}$$

We use OPT to denote the expected profit of the optimal adaptive policy. For each node v on the tree $\mathcal{T}_\sigma$, by Assumption 1 (2) that $f(I_v, a_v) \geq I_v$, we define $\mu_v := \Pr[f(I_v, a_v) > I_v] = 1 - \Phi_{a_v}(I_v, I_v)$. For a set of nodes P, we define $\mu(P) := \sum_{v \in P} \mu_v$.

▶ **Lemma 8.** *Given an policy σ, there is a policy σ' with profit at least $\mathsf{OPT} - O(\varepsilon) \cdot \mathsf{MAX}$ which satisfies that for any realization path $\mathcal{R}$, $\mu(\mathcal{R}) \leq O(1/\varepsilon)$, where $\mathsf{MAX} = \max_{I \in \mathcal{V}} \mathrm{DP}_1(I, \mathcal{A})$.*

W.l.o.g, we assume that all (optimal or near optimal) policies σ considered in this paper satisfy that for any realization $\mathcal{R}$, $\mu(\mathcal{R}) \leq O(1/\varepsilon)$.

3 Block Adaptive Policies

The decision tree corresponding to the optimal policy may be exponentially large and arbitrarily complicated. Now we consider a restrict class of policies, called block-adaptive policy. The concept was first introduced by Bhalgat *et al.* [6] in the context of stochastic knapsack. Our construction is somewhat different from that in [6, 17]. Here, we need to define an order for each block and introduce the notion of approximate block policy.

Formally, a block-adaptive policy $\hat{\sigma}$ can be thought as a decision tree $\mathcal{T}_{\hat{\sigma}}$. Each node on the tree is labeled by a *block* which is a set of items. For a block M, we choose an arbitrary order φ for the items in the block. According to the order φ, we take the items one by one, until we get a bigger value or all items in the block are taken but the value does not change (recall from Assumption 1 that the value is nondecreasing). Then we visit the child block which corresponds to the realized value. We use I_M to denote the current value right before taking the items in the block M. Then for each edge $e = (M, N)$, it has probability

$$\pi_e^\varphi = \sum_{a \in M} \left[\left(\prod_{\varphi_b < \varphi_a} \Phi_b(I_M, I_M) \right) \cdot \Phi_a(I_M, I_N) \right]$$

if $I_N > I_M$ and $\pi_e^\varphi = \prod_{a \in M} \Phi_a(I_M, I_M)$ if $I_N = I_M$.

Similar to Equation (8), for each block M and an arbitrary order φ for M, we recursively define the expected profit of the subtree $\mathcal{T}_M$ rooted at M to be

$$\mathbb{P}(M) = \mathcal{G}_M^\varphi + \sum_{e=(M,N)} \pi_e^\varphi \cdot \mathbb{P}(N) \tag{10}$$

if M is an internal block and $\mathbb{P}(M) = h(I_M)$ if M is a leaf (*i.e.,* the dummy node). Here $\mathcal{G}_M^\varphi$ is the expected profit we can get from the block which is equal to

$$\mathcal{G}_M^\varphi = \sum_{a \in M} \left[\left(\prod_{\varphi_b < \varphi_a} \Phi_b(I_M, I_M) \right) \cdot \mathcal{G}_a(I_M) \right].$$

Since the profit $\mathcal{G}_M^{\varphi}$ and the probability π_e^{φ} are dependent on the order φ and thus difficult to deal with, we define the approximate block profit and the approximate probability which do not depend on the choice of the specific order φ:

$$\widetilde{\mathcal{G}}_M = \sum_{a\in M} \mathcal{G}_a(I_M) \quad \text{and} \quad \widetilde{\pi}_e = \sum_{a\in M} \left[\left(\prod_{b\in M\setminus a} \Phi_b(I_M, I_M) \right) \cdot \Phi_a(I_M, I_N) \right] \tag{11}$$

if $I_N > I_M$ and $\widetilde{\pi}_e = \prod_{a\in M} \Phi_a(I_M, I_M)$ if $I_N = I_M$. Then we recursively define the approximate profit

$$\widetilde{\mathbb{P}}(M) = \widetilde{\mathcal{G}}_M + \sum_{e=(M,N)} \widetilde{\pi}_e \cdot \widetilde{\mathbb{P}}(N), \tag{12}$$

if M is an internal block and $\widetilde{\mathbb{P}}(M) = \mathbb{P}(M) = h(I_M)$ if M is a leaf. For each block M, we define $\mu(M) := \sum_{a\in M} [1 - \Phi_a(I_M, I_M)]$. Lemma 9 below can be used to bound the gap between the approximate profit and the original profit if the policy satisfies the following property.Then it suffices to consider the approximate profit for a block adaptive policy $\hat{\sigma}$ in this paper.

(P1) Each block M with more than one item satisfies that $\mu(M) \le \varepsilon^2$.

▶ **Lemma 9.** *For any block-adaptive policy $\hat{\sigma}$ satisfying Property (P1), we have*

$$\left(1 + O(\varepsilon^2)\right) \cdot \widetilde{\mathbb{P}}(\hat{\sigma}) \ge \mathbb{P}(\hat{\sigma}) \ge \left(1 - \varepsilon^2\right) \cdot \widetilde{\mathbb{P}}(\hat{\sigma}).$$

3.1 Constructing a Block Adaptive Policy

In this section, we show that there exists a block-adaptive policy that approximates the optimal adaptive policy. In order to prove this, from an optimal (or nearly optimal) adaptive policy σ, we construct a block adaptive policy $\hat{\sigma}$ which satisfies certain nice properties and can obtain almost as much profit as σ does. Thus it is sufficient to restrict our search to the block-adaptive policies. The construction is similar to that in [17].

▶ **Lemma 10.** *An optimal policy σ can be transformed into a block adaptive policy $\hat{\sigma}$ with approximate expected profit $\widetilde{\mathbb{P}}(\hat{\sigma})$ at least $\mathsf{OPT} - O(\varepsilon) \cdot \mathsf{MAX}$. Moreover, the block-adaptive policy $\hat{\sigma}$ satisfies Property (P1) and (P2):*

(P1) *Each block M with more than one item satisfies that $\mu(M) \le \varepsilon^2$.*

(P2) *There are at most $O(\varepsilon^{-3})$ blocks on any root-to-leaf path on the decision tree.*

Proof (sketch). For a node v on the decision tree $\mathcal{T}_\sigma$ and a value $s \in \mathcal{V}$, we use v_s to denote the s-child of v, which is the child of v corresponding to the realized value s (see Figure 1). We say an edge $e_{v,u}$ is *non-increasing* if $I_v = I_u$ and define the *leftmost path* of v to be the realization path which starts at v, ends at a leaf, and consists of only the non-increasing edges.

We say a node v is a *starting node* if v is the root or v corresponds to an increasing value of its parent v' (*i.e.,* $I_v > I_{v'}$). For each staring node v, we greedily partition the leftmost path of v into several segments such that for any two nodes u, w in the same segment M and for any value $s \in \mathcal{V}$, we have

$$|\mathbb{P}(u_s) - \mathbb{P}(w_s)| \le \varepsilon^2 \cdot \mathsf{MAX} \text{ and } \mu(M) \le \varepsilon^2. \tag{13}$$

For each root-to-leaf path $\mathcal{R}$, Equation (13) can yield at most $O(\varepsilon^{-3})$ blocks. Now, we are ready to describe the algorithm, which takes a policy σ as input and outputs a block adaptive

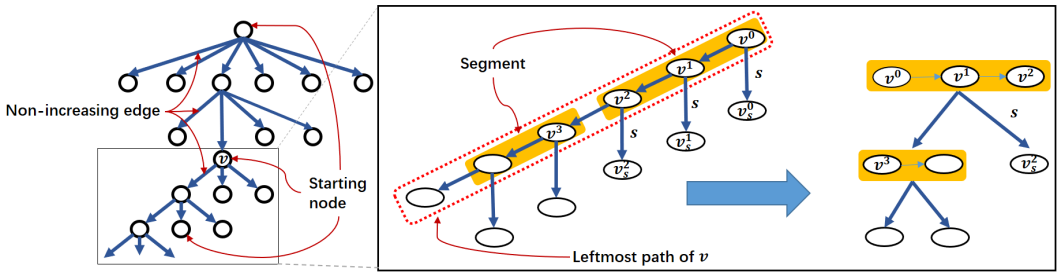


Figure 1 Decision tree and block policy.

Algorithm 1 A policy $\hat{\sigma}$.

Input: A policy σ.

1: We start at the root of $\mathcal{T}_\sigma$.
2: **repeat**
3: Suppose we are at node v on $\mathcal{T}_\sigma$. Take the items in $\mathsf{seg}(v)$ one by one in the original order (the order of items in policy σ) until some node u makes a transition to an increasing value, say s.
4: Visit the node $l(v)_s$, the s-child of $l(v)$ (*i.e.,* the last node of $\mathsf{seg}(v)$).
5: If all items in $\mathsf{seg}(v)$ have been taken and the value does not change, visit $l(v)_{I_v}$.
6: **until** A leaf on $\mathcal{T}_\sigma$ is reached.

policy $\hat{\sigma}$. For each node v, we denote its segment $\mathsf{seg}(v)$ and use $l(v)$ to denote the last node in $\mathsf{seg}(v)$. In Algorithm 1, we can see that the set of items which the policy $\hat{\sigma}$ attempts to take always corresponds to some realization path in the original policy σ. Property (P1) and (P2) hold immediately following from the partition argument. Then we can show that the expected profit $\mathbb{P}(\hat{\sigma})$ that the new policy $\hat{\sigma}$ can obtain is at least $\mathsf{OPT} - O(\varepsilon^2) \cdot \mathsf{MAX}$. ◀

3.2 Enumerating Signatures

To search for the (nearly) optimal block-adaptive policy, we want to enumerate all possible structures of the block decision tree. Fixing the topology of the decision tree, we need to decide the subset of items to place in each block. To do this, we define the *signature* such that two subsets with the same signature have approximately the same profit distribution. Then, we can enumerate the signatures of all blocks in polynomial time and find a nearly optimal block-adaptive policy. Formally, for an item $a \in \mathcal{A}$ and a value $I \in \mathcal{V} = (0, 1, \ldots, |\mathcal{V}| - 1)$, we define the *signature* of a on I to be the following vector

$$\mathsf{Sg}_I(a) = \left(\bar{\Phi}_a(I,0), \bar{\Phi}_a(I,1), \ldots, \bar{\Phi}_a(I,|\mathcal{V}|-1), \bar{\mathcal{G}}_a(I)\right),$$

where $\bar{\Phi}_a(I,J) = \left\lfloor \Phi_a(I,J) \cdot \frac{n}{\varepsilon^4} \right\rfloor \cdot \frac{\varepsilon^4}{n}$ and $\bar{\mathcal{G}}_a(I) = \left\lfloor \mathcal{G}_a(I) \cdot \frac{n}{\varepsilon^4 \mathsf{MAX}} \right\rfloor \cdot \frac{\varepsilon^4 \mathsf{MAX}}{n}$ for any $J \in \mathcal{V}$. [5] For a block M of items, we define the *signature* of M on I to be $\mathsf{Sg}_I(M) = \sum_{a \in M} \mathsf{Sg}_I(a)$.

▶ **Lemma 11.** *Consider two decision trees $\mathcal{T}_1, \mathcal{T}_2$ corresponding to block-adaptive policies with the same topology* (i.e., *$\mathcal{T}_1$ and $\mathcal{T}_2$ are isomorphic) and the two block adaptive policies*

[5] If $\mathsf{MAX} = \max_{I \in \mathcal{V}} \mathrm{DP}_1(I, \mathcal{A})$ is unknown, for some several concrete problems (*e.g.,* Probemax), we can get a constant approximation result for MAX, which is sufficient for our purpose. In general, we can guess a constant approximation result for MAX using binary search.

satisfy Property (P1) and (P2). If for each block M_1 on $\mathcal{T}_1$, the block M_2 at the corresponding position on $\mathcal{T}_2$ satisfies that $\mathsf{Sg}_I(M_1) = \mathsf{Sg}_I(M_2)$ where $I = I_{M_1} = I_{M_2}$, then $|\widetilde{\mathbb{P}}(\mathcal{T}_1) - \widetilde{\mathbb{P}}(\mathcal{T}_2)| \leq O(\varepsilon) \cdot \mathsf{MAX}$.

Since $|V| = O(1)$, the number of possible signatures for a block is $O\left((n/\varepsilon^4)^{|\mathcal{V}|}\right) = n^{O(1)}$, which is a polynomial of n. By Lemma 10, for any block decision tree $\mathcal{T}$, there are at most $(|\mathcal{V}|)^{O(\varepsilon^{-3})} = 2^{O(\varepsilon^{-3})}$ blocks on the tree which is a constant.

3.3 Finding a Nearly Optimal Block-adaptive Policy

In this section, we find a nearly optimal block-adaptive policy and prove Theorem 1. To do this, we enumerate over all topologies of the decision trees along with all possible signatures for each block. This can be done by a standard dynamic programming.

Consider a given tree topology $\mathcal{T}$. A configuration C is a set of signatures each corresponding to a block. Let t_1 and t_2 be the number of paths and blocks on $\mathcal{T}$ respectively. We define a vector $\mathsf{CA} = (u_1, u_2, \ldots, u_{t_1})$ where u_j is the upper bound of the number of items on the jth path. For each given $i \in [n], \mathsf{C}$ and CA, let $\mathcal{M}(i, \mathsf{C}, \mathsf{CA}) = 1$ indicate that we can reach the configuration C using a subset of items $\{a_1, \ldots, a_i\}$ such that the total number of items on each path j is no more than u_j and 0 otherwise. Set $\mathcal{M}(0, \mathbf{0}, \mathbf{0}) = 1$ and we compute $\mathcal{M}(i, \mathsf{C}, \mathsf{CA})$ in an lexicographically increasing order of $(i, \mathsf{C}, \mathsf{CA})$ as follows:

$$\mathcal{M}(i, \mathsf{C}, \mathsf{CA}) = \max\left\{\mathcal{M}(i-1, \mathsf{C}, \mathsf{CA}), \mathcal{M}(i-1, \mathsf{C}', \mathsf{CA}')\right\} \tag{14}$$

Now, we explain the above recursion as follows. In each step, we should decide how to place the item a_i on the tree $\mathcal{T}$. Notice that there are at most $t_2 = (|\mathcal{V}|)^{O(\varepsilon^{-3})} = 2^{O(\varepsilon^{-3})}$ blocks and therefore at most 2^{t_2} possible placements of item a_i and each placement is called *feasible* if there are no two blocks on which we place the item a_i have an ancestor-descendant relation. For a feasible placement of a_i, we subtract $\mathsf{Sg}(a_i)$ from each entry in C corresponding to the block we place a_i and subtract 1 from CA on each entry corresponding to a path including a_i, and in this way we get the resultant configuration C' and CA' respectively. Hence, the max is over all possible such $\mathsf{C}', \mathsf{CA}'$.

We have shown that the total number of all possible configurations on $\mathcal{T}$ is n^{t_2}. The total number of vectors CA is $T^{t_1} \leq n^{t_1} \leq n^{t_2} = n^{t_2}$ where T is the number of rounds. For each given $(i, \mathsf{C}, \mathsf{CA})$, the computation takes a constant time $O(2^{t_2})$. Thus we claim for a given tree topology, finding the optimal configuration can be done within $O(n^{2^{O(\varepsilon^{-3})}})$ time .

The proof of Theorem 1. Suppose σ^* is the optimal policy with expected profit $\mathbb{P}(\sigma^*) = \mathsf{OPT}$. We use the above dynamic programming to find a nearly optimal block adaptive policy σ. By Lemma 10, there exists a block adaptive policy $\hat{\sigma}$ such that

$$\widetilde{\mathbb{P}}(\hat{\sigma}) \geq \mathsf{OPT} - O(\varepsilon)\mathsf{MAX}.$$

Since the configuration of $\hat{\sigma}$ is enumerated at some step of the algorithm, our dynamic programming is able to find a block adaptive policy σ with the same configuration (the same tree topology and the same signatures for corresponding blocks). By Lemma 11, we have

$$\widetilde{\mathbb{P}}(\sigma) \geq \widetilde{\mathbb{P}}(\hat{\sigma}) - O(\varepsilon)\mathsf{MAX} \geq \mathsf{OPT} - O(\varepsilon)\mathsf{MAX}.$$

By Lemma 9, we have $\mathbb{P}(\sigma) \geq \left(1 - \varepsilon^2\right) \cdot \widetilde{\mathbb{P}}(\sigma) \geq \mathsf{OPT} - O(\varepsilon)\mathsf{MAX}$. Hence, the proof of Theorem 1 is completed. ◀

References

1 Marek Adamczyk, Maxim Sviridenko, and Justin Ward. Submodular stochastic probing on matroids. *Mathematics of Operations Research*, 41(3):1022–1038, 2016.

2 Arash Asadpour and Hamid Nazerzadeh. Maximizing stochastic monotone submodular functions. *Management Science*, 62(8):2374–2391, 2015.

3 Richard Bellman. Dynamic programming. In *Princeton University Press*, 1957.

4 Dimitri P. Bertsekas. *Dynamic programming and optimal control*. Athena scientific Belmont, MA, 1995.

5 Anand Bhalgat. A $(2 + \varepsilon)$-approximation algorithm for the stochastic knapsack problem. *Unpublished Manuscript*, 2011.

6 Anand Bhalgat, Ashish Goel, and Sanjeev Khanna. Improved approximation results for stochastic knapsack problems. *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1647–1665, 2011.

7 Kai Chen and Sheldon M. Ross. An adaptive stochastic knapsack problem. *European Journal of Operational Research*, 239(3):625–635, 2014. doi:10.1016/j.ejor.2014.06.027.

8 Wei Chen, Wei Hu, Fu Li, Jian Li, Yu Liu, and Pinyan Lu. Combinatorial multi-armed bandit with general reward functions. *Advances in Neural Information Processing Systems*, 2016.

9 Brian C Dean, Michel X Goemans, and Jan Vondrák. Adaptivity and approximation for stochastic packing problems. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 395–404. Society for Industrial and Applied Mathematics, 2005.

10 Anupam Gupta and Viswanath Nagarajan. A stochastic probing problem with applications. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 205–216. Springer, 2013.

11 Anupam Gupta, Viswanath Nagarajan, and Sahil Singla. Algorithms and adaptivity gaps for stochastic probing. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1731–1747. Society for Industrial and Applied Mathematics, 2016.

12 Nir Halman, Diego Klabjan, Chung Lun Li, James Orlin, and David Simchi-Levi. Fully polynomial time approximation schemes for stochastic dynamic programs. In *Nineteenth Acm-Siam Symposium on Discrete Algorithms*, pages 700–709, 2008.

13 Nir Halman, Diego Klabjan, Chung-Lun Li, James Orlin, and David Simchi-Levi. Fully polynomial time approximation schemes for stochastic dynamic programs. *SIAM Journal on Discrete Mathematics*, 28(4):1725–1796, 2014.

14 Nir Halman, Giacomo Nannicini, and James Orlin. A computationally efficient fptas for convex stochastic dynamic programs. *SIAM Journal on Optimization*, 25(1):317–350, 2015.

15 Taylan İlhan, Seyed MR Iravani, and Mark S Daskin. The adaptive knapsack problem with stochastic rewards. *Operations Research*, 59(1):242–248, 2011.

16 Asaf Levin and Aleksander Vainer. Adaptivity in the stochastic blackjack knapsack problem. *Theoretical Computer Science*, 516:121–126, 2014.

17 Jian Li and Wen Yuan. Stochastic combinatorial optimization via poisson approximation. *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 971–980, 2013.

18 Will Ma. Improvements and generalizations of stochastic knapsack and markovian bandits approximation algorithms. *Mathematics of Operations Research*, 2017. doi:10.1287/moor.2017.0884.

19 Kamesh Munagala. Approximation algorithms for stochastic optimization. `https://simons.berkeley.edu/talks/kamesh-munagala-08-22-2016-1`, Simons Institue for the Theory of Computing, 2016.

20 Warren B Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, volume 842. John Wiley & Sons, 2011.

21 David B Shmoys and Chaitanya Swamy. An approximation scheme for stochastic linear programming and its application to stochastic integer programs. *Journal of the ACM (JACM)*, 53(6):978–1012, 2006.

22 Sahil Singla. The price of information in combinatorial optimization. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2523–2532. SIAM, 2018.

23 Jan Vondrák, Chandra Chekuri, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 783–792. ACM, 2011.

24 Martin L. Weitzman. Optimal search for the best alternative. *Econometrica*, 47(3):641–654, 1979.

Semi-Supervised Algorithms for Approximately Optimal and Accurate Clustering

Buddhima Gamlath
École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
buddhima.gamlath@epfl.ch

Sangxia Huang
Sony Mobile Communications, Lund, Sweden
huang.sangxia@gmail.com

Ola Svensson
École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
ola.svensson@epfl.ch

Abstract

We study k-means clustering in a semi-supervised setting. Given an oracle that returns whether two given points belong to the same cluster in a fixed optimal clustering, we investigate the following question: how many oracle queries are sufficient to efficiently recover a clustering that, with probability at least $(1-\delta)$, simultaneously has a cost of at most $(1+\epsilon)$ times the optimal cost and an accuracy of at least $(1-\epsilon)$?

We show how to achieve such a clustering on n points with $O((k^2 \log n) \cdot m(Q, \epsilon^4, \delta/(k \log n)))$ oracle queries, when the k clusters can be learned with an ϵ' error and a failure probability δ' using $m(Q, \epsilon', \delta')$ labeled samples in the supervised setting, where Q is the set of candidate cluster centers. We show that $m(Q, \epsilon', \delta')$ is small both for k-means instances in Euclidean space and for those in finite metric spaces. We further show that, for the Euclidean k-means instances, we can avoid the dependency on n in the query complexity at the expense of an increased dependency on k: specifically, we give a slightly more involved algorithm that uses $O(k^4/(\epsilon^2\delta) + (k^9/\epsilon^4)\log(1/\delta) + k \cdot m(\mathbb{R}^r, \epsilon^4/k, \delta))$ oracle queries.

We also show that the number of queries needed for $(1-\epsilon)$-accuracy in Euclidean k-means must linearly depend on the dimension of the underlying Euclidean space, and for finite metric space k-means, we show that it must at least be logarithmic in the number of candidate centers. This shows that our query complexities capture the right dependencies on the respective parameters.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Clustering, Semi-supervised Learning, Approximation Algorithms, k-Means, k-Median

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.57

Related Version The full version of this work is available at `https://arxiv.org/abs/1803.00926`.

Funding This research was supported by ERC Starting Grant 335288-OptApprox.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 57; pp. 57:1–57:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Clustering is a fundamental problem that arises in many learning tasks. Given a set P of data points, the goal is to output a k-partition $C_1 \dot\cup \ldots \dot\cup C_k$ of P according to some optimization criteria. In unsupervised clustering, the data points are unlabeled. The classic k-means problem and other well-studied clustering problems such as k-median fall into this category.

In a general k-means clustering problem, the input comprises a finite set of n points P that is to be clustered, a set of candidate centers Q, and a distance metric d giving the distances between each pair of points in $P \cup Q$. The goal is to find k cluster centers $c_1, \ldots, c_k \in Q$ that minimizes the cost, which is the sum of squared distances between each point in P and its closest cluster center. In this case, the clustering $\mathcal{C}$ is defined by setting $C_i = \{x \in P : c_i \text{ is the closest center to } x\}$ for all $i = 1, \ldots, k$ and breaking ties arbitrarily. Two widely studied special cases are the k-means problem in Euclidean space (where $P \subset \mathbb{R}^r, Q = \mathbb{R}^r$, and d is the Euclidean distance function) and the k-means problem in finite metric spaces (where $(P \cup Q, \mathrm{d})$ forms a finite metric space).

Despite its popularity and success in many settings, there are two known drawbacks of the unsupervised k-means problem:

1. Finding the centers that satisfy the clustering goal is computationally hard. For example, even the special case of 2-means problem in Euclidean space is NP-hard [8].
2. There could be multiple possible sets of centers that minimize the cost. However, in practical instances, not all such sets are equally meaningful, and we would like our algorithm to find one that corresponds to the concerns of the application.

Since k-means is NP-hard, it is natural to seek approximation algorithms. For the general k-means problem in Euclidean space, notable approximation results include the local search by Kanungo et al. [13] with an approximation guarantee of $(9 + \epsilon)$ and the recent LP-based 6.357-approximation algorithm by Ahmadian et al. [1]. On the negative side, Lee et al. [14] ruled out arbitrarily good approximation algorithms for the k-means problem on general instances. For several special cases, however, there exist PTASes. For example, in the case where k is constant, Har-Peled and Mazumdar [11] and Feldman et al. [9] showed how to get a PTAS using weak coresets, and in the case where the dimension d is constant, Cohen-Addad et al. [7] and Friggstad et al. [10] gave PTASes based on a basic local search algorithm. In addition, Awasthi et al. [4] presented a PTAS for k-means, assuming that the input is "clusterable" (satisfies a certain stability criterion).

Even if we leave aside the computational issues with unsupervised k-means, we still have the problem that there can be multiple different clusterings that minimize the cost. To see this, consider the 2-means problem on the set of vertices of an equilateral triangle. In this case, we have three different clusterings that give the same minimum cost, but only one of the clusterings might be meaningful. One way to avoid this issue is to have strong assumptions on the input. For example, Balcan et al. [5] considered the problem in a restricted setting where any c-approximation to the problem also classifies at least a $(1 - \epsilon)$ fraction of the points correctly.

Ashtiani et al. [3] recently proposed a different approach for addressing the aforementioned drawbacks. They introduced a semi-supervised active clustering framework where the algorithm is allowed to make queries of the form *same-cluster(x, y)* to a domain expert, and the expert replies whether the points x and y belong to the same cluster in some fixed optimal clustering. Under the additional assumptions that the clusters are contained inside k balls in $\mathbb{R}^r$ that are sufficiently far away from each other, they presented an algorithm that makes $O(k^2 \log k + k(\log n + \log(1/\delta)))$ same-cluster queries, runs in $O(kn \log n + k^2 \log(1/\delta))$ time,

and recovers the clusters with probability at least $(1-\delta)$. Their algorithm finds approximate cluster centers, orders all points by their distances to the cluster centers, and performs binary searches to determine the radii of the balls. Although it recovers the exact clusters, this approach works only when the clusters are contained inside well-separated balls. When the clusters are determined by a general Voronoi partitioning, and thus distances to the cluster boundaries can differ in different directions, this approach fails.

A natural question arising from the work of Ashtiani et al. [3] is whether such strong assumptions on the input structure are necessary. Ailon et al. [2] addressed this concern and considered the problem without any assumptions on the structure of the underlying true clusters. Their main result was a polynomial-time $(1+\epsilon)$-approximation scheme for k-means in the same semi-supervised framework as in Ashtiani et al. [3]. However, in contrast to Ashtiani et al. [3], their work gives no assurance on the accuracy of the recovered clustering compared to the true clustering. To achieve their goal, the authors utilized importance sampling to uniformly sample points from small clusters that significantly contribute to the cost. Their algorithm makes $O(k^9/\epsilon^4)$ same-cluster queries, runs in $O(nr(k^9/\epsilon^4))$ time, and succeeds with a constant probability.

In this work, we investigate the k-means problem in the same semi-supervised setting as Ailon et al. [2], but in addition to approximating the cost, we seek a solution that is also accurate with respect to the true clustering. We assume that the underlying true clustering minimizes the cost, and that there are no points on cluster boundaries (i.e., the margin between each pair of clusters can be arbitrarily small but not zero). This last assumption is what differentiates our setup from that of Ailon et al. [2]. It is reasonable to assume that no point lies on the boundary of two clusters, as otherwise, to achieve constant accuracy, we would have to query at least a constant fraction of the boundary points. Without querying each boundary point, we have no way of determining to which cluster it belongs.

Observe that if we label all the points correctly with respect to the true clustering, the resulting clustering automatically achieves the optimal cost. However, such perfect accuracy is difficult to achieve as there may be points that are arbitrarily close to each other but belong to different clusters. Using only a reasonable number of samples, the best we can hope for is to recover an approximately accurate solution. PAC (Probably Approximately Correct) learning helps us achieve this goal and provides a trade-off between the desired accuracy and the required number of samples.

Suppose that we have a solution where only a small fraction of the input points is incorrectly classified. In this case, one would hope that the cost is also close to the optimal cost. Unfortunately, the extra cost incurred by the incorrectly classified points can be very high depending on their positions, true labels, and the labels assigned to them. Our main concern in this paper is controlling this additional cost.

We show that if we start with a constant-factor approximation for the cost, we can refine the clustering using a PAC learning algorithm. This yields a simple polynomial-time algorithm that, given a k-means instance and $(\epsilon, \delta) \in (0,1)^2$ as parameters, with probability at least $(1-\delta)$ outputs a clustering that has a cost of at most $(1+\epsilon)$ times the optimal cost and that classifies at least a $(1-\epsilon)$ fraction of the points correctly with respect to the underlying true clustering. To do so, the algorithm makes $O((k^2 \log n) \cdot m(Q, \epsilon^4, \delta/(k \log n)))$ same-cluster queries. Here, $m(Q, \epsilon', \delta')$ is the sufficient number of labeled samples for a PAC learning algorithm to learn k clusters in a k-means instance with an ϵ' error and a failure probability δ' in the supervised setting (recall that Q is the set of candidate centers). We further show that our algorithm can be easily adapted to k-median and other similar problems that use the ℓ'th power of distances in place of squared distances for some fixed

$\ell > 0$. We formally present this result as Theorem 6 in Section 3. In Theorem 1 below, we give an informal statement for the case of k-means.

▶ **Theorem 1** (An informal version of Theorem 6). *There exists a semi-supervised learning algorithm that, given a k-means instance, oracle access to same-cluster queries that are consistent with some fixed optimal clustering, and parameters $(\epsilon, \delta) \in (0,1)^2$, outputs a clustering that, with probability at least $(1-\delta)$, correctly labels (up to a permutation of the labels) at least a $(1-\epsilon)$ fraction of the points and, simultaneously, has a cost of at most $(1+\epsilon)$ times the optimal cost. In doing so, the algorithm makes $O((k^2 \log n) \cdot m(Q, \epsilon^4, \delta/(k \log n)))$ same-cluster queries.*

Our algorithm is general and applicable to any family of k-means, k-median, or similar distance based clustering instances that can be efficiently learned with PAC learning. As discussed later in this work, these include Euclidean and general finite metric space clustering instances. In contrast, both Ashtiani et al. [3] and Ailon et al. [2], considered only the Euclidean k-means problem. To the best of our knowledge, ours is the first such result applicable to finite metric space k-means and both Euclidean and finite metric space k-median problems.

Ideally, we want $m(Q, \epsilon, \delta)$ to be small. Additionally, the analysis of our algorithm relies on two natural properties of learning algorithms. Firstly, we require PAC learning to always correctly label all the sampled points. Secondly, we also require it to not 'invent' new labels and only output labels that it has seen on the samples. We show that such learning algorithms with small $m(Q, \epsilon, \delta)$ exist both for k-means instances in Euclidean space and for those in finite metric spaces with no points on the boundaries of the optimal clusters. For r-dimensional Euclidean k-means, $m(Q = \mathbb{R}^r, \epsilon, \delta)$ has a linear dependency on r. For the case of finite metric spaces, $m(Q, \epsilon, \delta)$ has a logarithmic dependency on $|Q|$, which is the size of the set of candidate centers. In fact, these learning algorithms are applicable not only to k-means instances but also to instances of other similar center-based clustering problems (where clusters are defined by assigning points to their closest cluster centers). We discuss our learning algorithms in detail in the full version of this paper.

Our semi-supervised learning algorithm is inspired by the work of Feldman et al. [9] on weak coresets. Their construction of the weak coresets first obtains an intermediate clustering using a constant-factor approximation algorithm and refines each intermediate cluster by taking random samples. In order to get a good guarantee for the cost, their algorithm partitions each cluster into an inner ball that contains the majority of the points, and an outer region that contains the remaining points. We proceed similarly to this construction; however, we further partition the outer region into $O(\log n)$ concentric rings and use PAC learning to label the points in the inner ball and in each of the outer rings separately. For Euclidean k-means instances, the number of same-cluster queries needed by the algorithm has a logarithmic dependency on the number n of points, which is similar (up to a $\text{poly}(\log \log n)$ factor) to that of the algorithm by Ashtiani et al. [3]. The advantage of our algorithm is that it works for a much broader range of k-means instances whereas the applicability of the algorithm of Ashtiani et al. [3] is restricted to those instances whose clusters are contained in well-separated balls in Euclidean space.

This algorithm is effective in many natural scenarios where the number of clusters k is larger than $\log n$. However, as the size of the k-means instance (i.e., the number of points) becomes large, the $\log n$ factor becomes undesirable. In Euclidean k-means, the number of samples needed by the learning algorithm for an ϵ error and a failure probability δ does not depend on n. The $\log n$ dependency in the final query complexity is exclusively due to repeating the PAC learning step on $\Omega(k \log n)$ different partitions of P. To overcome

this problem, we present a second algorithm, which is applicable only to Euclidean k-means instances, inspired by the work of Ailon et al. [2]. This time, we start with a $(1+\epsilon)$-approximation for the cost and refine it using PAC learning. Unlike our first algorithm, we only run the PAC learning once on the whole input, and thus we completely eliminate the dependency on n. The disadvantages of this algorithm compared to our first algorithm are the slightly more involved nature of the algorithm and the increased dependency on k in its query complexity. Theorem 2 below formally states this result. We present our algorithm in Section 4 and discuss the key ideas that lead to its construction. The complete proof of Theorem 2 is given in the full version.

▶ **Theorem 2.** *There exists a polynomial-time algorithm that, given a k-means instance in r-dimensional Euclidean space, oracle access to same-cluster queries that are consistent with some fixed optimal clustering, and parameters $(\epsilon, \delta) \in (0,1)^2$, outputs a clustering that, with probability at least $(1-\delta)$, correctly labels (up to a permutation of the labels) at least a $(1-\epsilon)$ fraction of the points and, simultaneously, has a cost of at most $(1+\epsilon)$ times the optimal cost. The algorithm makes $O(k^4/(\epsilon^2\delta) + (k^9/\epsilon^4)\log(1/\delta) + k \cdot m(\mathbb{R}^r, \epsilon^4/k, \delta))$ same-cluster queries.*

For the Euclidean setting, the query complexities of both our algorithms have a linear dependency on the dimension of the Euclidean space. The algorithm of Ashtiani et al. [3] does not have such a dependency due to their strong assumption on the cluster structure, whereas the one by Ailon et al. [2] does not have that as it only approximates the cost. We show that, in our scenario, such a dependency is necessary to achieve the accuracy guarantees of our algorithms. For the finite metric space k-means, the query complexity of our general algorithm has an $O(\text{poly}(\log|P|, \log|Q|))$ dependency. The dependency on $|P|$ comes from the repeated application of the learning algorithm on $\Omega(k \log|P|)$ different partitions, and whether we can avoid this is an open problem. However, we show that an $\Omega(\log|Q|)$ query complexity is necessary for the accuracy. We formalize these results in Theorem 3 below (the proof is in the full version).

▶ **Theorem 3.** *Let K be a family of k-means instances. Let $\mathcal{A}$ be an algorithm that, given a k-means instance in K, oracle access to same-cluster queries for some fixed optimal clustering, and parameters $(\epsilon, \delta) \in (0,1)^2$, outputs a clustering that, with probability at least $(1-\delta)$, correctly labels (up to a permutation of the cluster labels) at least a $(1-\epsilon)$ fraction of the points. Then, the following statements hold:*

1. *If K is the family of k-means instances in r-dimensional Euclidean space that have no points on the boundaries of optimal clusters, $\mathcal{A}$ must make $\Omega(r)$ same-cluster queries.*
2. *If K is the family of finite metric space k-means instances that have no points on the boundaries of optimal clusters, $\mathcal{A}$ must make $\Omega(\log|Q|)$ same-cluster queries.*

The outline of this extended abstract is as follows. In Section 2 we introduce the notation, formulate the problem and present the learning theorems that we use in the subsequent sections. In Section 3 we present our first algorithm, which is simple and applicable to general k-means instances that admit efficient learning algorithms, but has a dependency of $\log n$ in its query complexity. Finally, in Section 4 we discuss how to remove the $\log n$ dependency in the query complexity for the special case of Euclidean k-means instances and present our second algorithm.

In the full version, we present formal proofs of all the stated results, where we also introduce the basic concepts and tools of PAC learning and explain how to design learning algorithms for Euclidean and finite metric space k-means instances.

2 Preliminaries

In this section, we introduce the basic notation and two common families of k-means instances, and formally define the k-means problem that we address in this work. We also introduce the notion of *learnability* for families of k-means instances and state two learning theorems that will be used in the later sections.

2.1 k-Means Problem in a Semi-supervised Setting

Let P and Q be two sets of points where $|P| = n$, and let $\mathrm{d} : (P \cup Q) \times (P \cup Q) \to \mathbb{R}_+$ be a distance metric. We denote a k-means instance by the triple (P, Q, d). Two common families of k-means instances we consider in this work are:

1. k-means instances in Euclidean space, where $P \subset \mathbb{R}^r$, $Q = \mathbb{R}^r$, and $\mathrm{d}(x_1, x_2) = \|x_1 - x_2\|$ is the Euclidean distance between x_1 and x_2, and
2. k-means instances in finite metric spaces, where $(P \cup Q, \mathrm{d})$ forms a finite metric space.

Let $[k] := \{1, \ldots, k\}$. We identify a k-clustering $\mathcal{C}$ of (P, Q, d) by a labeling function $f_{\mathcal{C}} : P \to [k]$, and a set of k centers, $c_1, \ldots, c_k \in Q$, associated with each label, $1, \ldots, k$. For each label $i \in [k]$ of a clustering $\mathcal{C}$, let $C_i := \{p \in P : f_{\mathcal{C}}(p) = i\}$ be the set of points whose label is i. For convenience, we may use the labeling function $f_{\mathcal{C}}$ or the set of clusters $\{C_1, \ldots, C_k\}$ interchangeably to denote a clustering $\mathcal{C}$.

For a subset $C \subseteq P$ and a point $q \in Q$, define $\mathrm{cost}(C, q) := \sum_{p \in C} \mathrm{d}^2(p, q)$. For each i, define center $c_i := \mathrm{argmin}_{q \in Q} \mathrm{cost}(C_i, q)$, i.e., each center is a point in Q that minimizes the sum of squared distances between itself and each of the points assigned to it. For a k-clustering $\mathcal{C}$, we define its *k-means cost* as $\mathrm{cost}(\mathcal{C}) := \sum_{i \in [k]} \mathrm{cost}(C_i, c_i)$. Let $\mathcal{C}^*$ be the set of all k-clusterings of (P, Q, d). Then, the optimal k-means cost of (P, Q, d) is defined as $OPT := \min_{\mathcal{C} \in \mathcal{C}^*} \mathrm{cost}(\mathcal{C})$. We say that a k-clustering $\mathcal{C}$ α-approximates the k-means cost if $\mathrm{cost}(\mathcal{C}) \leq \alpha OPT$.

Let $\mathcal{O}$ be a fixed k-clustering of (P, Q, d) that achieves the optimal k-means cost, and let $\mathcal{C}$ be any k-clustering of P. Let $f_{\mathcal{O}}$ and $f_{\mathcal{C}}$ be the labeling functions that correspond to $\mathcal{O}$ and $\mathcal{C}$ respectively. We assume that we have oracle access to the labeling function $f_{\mathcal{O}}$ of the optimal target clustering up to a permutation of the labels. We can simulate a single query to such an oracle with $O(k)$ queries to a same-cluster oracle (see the full version for details). A same-cluster oracle is an oracle that answers *same-cluster*(p_1, p_2) queries with 'yes' or 'no' based on whether p_1 and p_2 belong to the same cluster in the fixed optimal clustering $\mathcal{O}$.

The error of a clustering $\mathcal{C}$ with respect to the clustering $\mathcal{O}$ for a k-means instance (P, Q, d) is now defined as $\mathrm{error}(\mathcal{C}, \mathcal{O}) := \min_{\sigma} |\{p \in P : f_{\mathcal{O}}(p) \neq \sigma(f_{\mathcal{C}}(p))\}|$, where the minimization is over all permutations $\sigma : [k] \to [k]$. In other words, $\mathrm{error}(\mathcal{C}, \mathcal{O})$ is the minimum number of points incorrectly labeled by the clustering $\mathcal{C}$ with respect to the optimal clustering $\mathcal{O}$, considering all possible permutations of the cluster labels. The reason for defining error in this manner is because we use a simulated version of $f_{\mathcal{O}}$ (which is only accurate up to a permutation of the cluster labels) instead of the true $f_{\mathcal{O}}$ to learn cluster labels. We say that a k-clustering $\mathcal{C}$ is $(1 - \alpha)$-accurate with respect to $\mathcal{O}$ if $\mathrm{error}(\mathcal{C}, \mathcal{O}) \leq \alpha n$.

Given (P, Q, d), parameters k and $(\epsilon, \delta) \in (0, 1)^2$, and oracle access to $f_{\mathcal{O}}$, our goal is to output a k-clustering $\hat{\mathcal{O}}$ of (P, Q, d) that, with probability at least $(1 - \delta)$, satisfies $\mathrm{error}(\hat{\mathcal{O}}, \mathcal{O}) \leq \epsilon n$ and $\mathrm{cost}(\hat{\mathcal{O}}) \leq (1 + \epsilon) OPT$.

2.2 PAC Learning for k-Means

Let K be a family of k-means instances, and let $m(Q, \epsilon, \delta)$ be a positive integer-valued function. We say such a family K is *learnable* with *sample complexity* m if there exists a learning algorithm $\mathcal{A}_L$ such that the following holds: Let $\epsilon \in (0,1)$ be an error parameter and let $\delta \in (0,1)$ be a probability parameter. Let (P, Q, d) be a k-means instance that belongs to K. Let $\mathcal{O}$ be a fixed optimal k-means clustering and let $f_{\mathcal{O}}$ be the associated labeling function. Let T be a fixed subset of P, and let S be a multiset of at least $m(Q, \epsilon, \delta)$ independently and uniformly distributed samples from T. The algorithm $\mathcal{A}_L$, given input (P, Q, d) and $(s, f_{\mathcal{O}}(s))$ for all $s \in S$, outputs a function $h : P \to [k]$. Moreover, with probability at least $(1-\delta)$ over the choice of S, the output h agrees with $f_{\mathcal{O}}$ on at least a $(1-\epsilon)$ fraction of the points in T (i.e., $|\{p \in T : h(p) = f_{\mathcal{O}}(p)\}| \geq (1-\epsilon)|T|$). This simpler notion of learnability is sufficient for the purpose of this work although it deviates from that of the general PAC learnability, which concerns with samples drawn from arbitrary distributions.

We say that such a learning algorithm $\mathcal{A}_L$ has the *zero sample error* property if the output h of $\mathcal{A}_L$ assigns the correct label to all the sampled points (i.e., $h(s) = f_{\mathcal{O}}(s)$ for all $s \in S$). Furthermore, we say that such a learning algorithm $\mathcal{A}_L$ is *non-inventive* if it does not 'invent' labels that it has not seen. This means that the output h of $\mathcal{A}_L$ does not assign labels that were not present in the input (sample, label) pairs (i.e., if $h(x) = c$ for some $x \in P$, then for some sample point $s \in S$, $f_{\mathcal{O}}(s) = c$).

In Section 3, we present a simple algorithm for $(1+\epsilon)$-approximate and $(1-\epsilon)$-accurate k-means clustering for a family K of k-means instances, assuming that K is learnable with a zero sample error, non-inventive learning algorithm. In the analysis, zero sample error and non-inventive properties play a key role in the crucial step of bounding the cost of incorrectly labeled points in terms of that of correctly labeled nearby points.

We now present two learning theorems for the Euclidean setting and the finite metric space setting (see the full version for the proofs). Assuming no point lies on cluster boundaries, the theorems state that the labeling function $f_{\mathcal{O}}$ of the optimal clustering is learnable with a zero sample error, non-inventive learning algorithm in both settings. We say that a k-means instance (P, Q, d) has *no boundary points* if in any optimal clustering $\mathcal{O}$ with clusters $O_1, \ldots, O_k$ and respective centers $o_1, \ldots, o_k$, the closest center to any given point $p \in P$ is unique (i.e., if $p \in O_i$, $\mathrm{d}(p, o_i) < \mathrm{d}(p, o_j)$ for all $j \neq i$).

▶ **Theorem 4** (Learning k-Means in Euclidean Space). *Let* $\mathrm{d}(p_1, p_2) = \|p_1 - p_2\|$ *be the Euclidean distance function. Let* $K = \{(P, \mathbb{R}^r, \mathrm{d}) : P \subset \mathbb{R}, |P| < \infty, (P, \mathbb{R}^r, \mathrm{d})$ *has no boundary points*$\}$ *be the family of* k*-means instances that are in* r*-dimensional Euclidean space and that have no boundary points. The family* K *is learnable with sample-complexity*[1] $m(\mathbb{R}^r, \epsilon, \delta) = \tilde{O}((k^2 r \log(k^2 r)(\log(k^3 r/\epsilon)) + \log(1/\delta))/\epsilon)$.

▶ **Theorem 5** (Learning k-Means in Finite Metric Spaces). *Let* $K = \{(P, Q, \mathrm{d}) : (P \cup Q, \mathrm{d})$ *is a finite metric space, and* (P, Q, d) *has no boundary points*$\}$ *be the family of finite metric space* k*-means instances that have no boundary points. The family* K *is learnable with sample-complexity*[2] $m(Q, \epsilon, \delta) = \tilde{O}((k^2(\log k)(\log |Q|)(\log k + \log 1/\epsilon) + \log(1/\delta))/\epsilon)$.

[1] $\tilde{O}$ hides $\mathrm{poly}(\log\log k, \log\log r)$ factors.
[2] $\tilde{O}$ hides $\mathrm{poly}(\log\log k, \log\log |Q|)$ factors.

3 A Simple Algorithm for $(1+\epsilon)$ Cost and $(1-\epsilon)$ Accuracy

Let K be a family of k-means instances that is learnable with sample complexity m using a zero sample error, non-inventive learning algorithm $\mathcal{A}_L$. Let $\mathcal{A}_\alpha$ be a constant-factor approximation algorithm (in terms of cost) for k-means, and let $\mathcal{A}_1$ be a polynomial-time algorithm for the 1-means problem (i.e., given $(P, Q, \mathrm{d}) \in K$, $\mathcal{A}_1$ finds $\operatorname{argmin}_{q \in Q} \operatorname{cost}(P, q)$ in polynomial time). We present a simple semi-supervised learning algorithm that, given a k-means instance (P, Q, d) of class K and oracle access to the labeling function $f_{\mathcal{O}}$ of a fixed optimal clustering $\mathcal{O}$ of (P, Q, d), outputs a clustering $\hat{\mathcal{O}}$ that, with probability at least $(1-\delta)$, satisfies $\operatorname{cost}(\hat{\mathcal{O}}) \leq (1+\epsilon)OPT$ and $\operatorname{error}(\hat{\mathcal{O}}, \mathcal{O}) \leq \epsilon|P|$. Our algorithm uses $\mathcal{A}_\alpha$, $\mathcal{A}_1$, and $\mathcal{A}_L$ as subroutines and makes $O((k \log |P|) \cdot m(Q, \epsilon^4, \delta/(k \log |P|)))$ oracle queries. We show that our algorithm can be easily modified for $(1+\epsilon)$-approximate and $(1-\epsilon)$-accurate k-median and other similar distance-based clustering problems. Towards the end of this section, we discuss several applications of this result, namely, for Euclidean and finite metric space k-means and k-median problems.

Let us start by applying the learning algorithm $\mathcal{A}_L$ to learn all the cluster labels. If we get perfect accuracy, the cost will be optimal. A natural question to ask in this case is: what happens to the cost if the learning output has ϵ error? In general, even a single misclassified point can incur an arbitrarily large additional cost. To better understand this, consider the following: Let $O_i, O_j \subseteq P$ be two distinct optimal clusters in the target clustering, and let o_i, o_j be their respective cluster centers. Let $p \in O_i$ be a point that is incorrectly classified and assigned label $j \neq i$ by $\mathcal{A}_L$. Also assume that the number of misclassified points is small enough so that the centers of the clusters output by the learning algorithm are close to those of the optimal clustering. Thus, in the optimal clustering, p incurs a cost of $\mathrm{d}^2(p, o_i)$, whereas according to the learning outcome, p incurs a cost that is close to $\mathrm{d}^2(p, o_j)$. In the worst case, $\mathrm{d}(p, o_j)$ can be arbitrarily larger than $\mathrm{d}(p, o_i)$.

Now suppose that, within distance ρ from p, there exists some point $q \in O_j$. In this case, we can bound the cost incurred due to the erroneous label of p using the true cost of p in the target clustering. To be more specific, using the triangle inequality, we get the following bound for any metric space: $\mathrm{d}(p, o_j) \leq \mathrm{d}(p, q) + \mathrm{d}(q, o_j) \leq \rho + \mathrm{d}(q, o_j)$. Furthermore, due to the optimality, $\mathrm{d}(q, o_j) \leq \mathrm{d}(q, o_i) \leq \mathrm{d}(q, p) + \mathrm{d}(p, o_i) \leq \rho + \mathrm{d}(p, o_i)$. Hence, it follows that $\mathrm{d}(p, o_j) \leq 2\rho + \mathrm{d}(p, o_i)$. To utilize this observation in an algorithmic setting, we need to make sure that, for every point that is misclassified into cluster j, there exists a correctly classified nearby point q that belongs to the optimal cluster O_j. Luckily, this is ensured by the combination of zero sample error and non-inventive properties of $\mathcal{A}_L$. If a point is misclassified into cluster j, the non-inventive property says that $\mathcal{A}_L$ must have seen a sample point q from cluster j. The zero sample error property ensures that q is labeled correctly by $\mathcal{A}_L$. To make sure that such correctly labeled points are sufficiently close to their incorrectly labeled counterparts, we run $\mathcal{A}_L$ separately on certain suitably bounded partitions of P.

The formal description of our algorithm is given in Algorithm 1. The outline is as follows: First, we run $\mathcal{A}_\alpha$ on (P, Q, d) and obtain an intermediate clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$. For each C_i, we run $\mathcal{A}_1$ to find a suitable center c_i. Next, we partition each intermediate cluster C_i into an inner ball and $O(\log |P|)$ outer rings centered around c_i. We run the learning algorithm $\mathcal{A}_L$ separately on each of these partitions. We choose the inner and outer radii of the rings so that, in each partition, the points that are incorrectly classified by the learning algorithm only incur a small additional cost compared to that of the correctly classified points. The final output is a clustering $\hat{\mathcal{O}}$ that is consistent with the learning outputs on each of the partitions. For each cluster $\hat{O}_i$, we associate the output of running $\mathcal{A}_1$ on $(\hat{O}_i, Q, \mathrm{d})$ as

Algorithm 1: A simple algorithm for $(1+\epsilon)$-approximate $(1-\epsilon)$-accurate k-means.

Input : k-Means instance (P, Q, d), oracle access to $f_{\mathcal{O}}$, constant-factor approximation algorithm $\mathcal{A}_\alpha$ for k-means, 1-means algorithm $\mathcal{A}_1$, learning algorithm $\mathcal{A}_L$ with sample complexity m, accuracy parameter $0 < \epsilon < 1$, and failure probability $0 < \delta < 1$.

Output : The clustering $\hat{\mathcal{O}} = \{\hat{O}_1, \ldots, \hat{O}_k\}$ defined by the labeling $f_{\hat{\mathcal{O}}} : P \to [k]$ computed below. The respective cluster centers are $\hat{o}_i = \operatorname{argmin}_{q \in Q} \operatorname{cost}(\hat{O}_i, q)$, which can be found by running $\mathcal{A}_1$ on $(\hat{O}_i, Q, \mathrm{d})$.

1 Let $n = |P|$, and let $\gamma = \epsilon^2/(288\alpha)$.
2 Run $\mathcal{A}_\alpha$ and obtain an α-approximate k-means clustering $\mathcal{C} = \{C_1, \ldots, C_k\}$. For each $i \in [k]$, run $\mathcal{A}_1$ on (C_i, Q, d) and find centers $c_i = \operatorname{argmin}_{q \in Q} \operatorname{cost}(C_i, q)$.
3 **for** $C_i \in \mathcal{C}$ **do**
4 Let $r_i = \sqrt{\operatorname{cost}(C_i, c_i)/(\gamma |C_i|)}$.
5 Let $C_{i,0}$ be all points in C_i that are at most r_i away from c_i.
6 Let $C_{i,j}$ be the points in C_i that are between $2^{j-1} r_i$ and $2^j r_i$ away from c_i for $j = 1, \ldots, (\log n)/2$.
7 Let $m' = m(Q, \gamma^2, \delta/(k \log n))$.
8 **for** *each non-empty* $C_{i,j}$ **do**
9 Sample m' points $x_1, \ldots, x_{m'} \in C_{i,j}$ independently and uniformly at random.
10 Query the oracle on $x_1, \ldots, x_{m'}$ and let $S_{i,j} = \{(x_i, f_{\mathcal{O}}(x_i)) : i = 1, \ldots, m'\}$.
11 Run $\mathcal{A}_L$ on input (P, Q, d) and $S_{i,j}$, and obtain a labeling $h_{i,j} : C_{i,j} \to [k]$.
12 Output the clustering $\hat{\mathcal{O}}$ defined by the following labeling function:
13 **for** *each* i, j, $x \in C_{i,j}$ **do**
14 Set $f_{\hat{\mathcal{O}}}(x) = h_{i,j}(x)$.

its center. Note that, due to the accuracy requirements, the cluster center to which a point is assigned in the output may not be the cluster center closest to that point in the output. It remains an interesting problem to find an accurate clustering in which every point is always assigned to its nearest cluster center.

With probability at least $(1-\delta)$, Algorithm 1 outputs a $(1+\epsilon)$-approximately optimal, $(1-\epsilon)$-accurate k-means clustering (the complete analysis is in the full version). In Algorithm 1, instead of an exact algorithm $\mathcal{A}_1$ for the 1-means problem, we can also use a PTAS. Using a PTAS to approximate 1-means up to a $(1+\epsilon)$ factor will only cost an additional $(1+\epsilon)$ factor in our cost analysis. As a result, we get the same approximation and accuracy guarantees if we replace ϵ with $\epsilon/3$.

Algorithm 1 makes $O((k \log n) \cdot m(Q, \epsilon^4, \delta/(k \log n)))$ queries to the oracle $f_{\mathcal{O}}$ in total. Recall that simulating an oracle query to $f_{\mathcal{O}}$ takes $O(k)$ same-cluster queries. Therefore, the total number of same-cluster queries is $O((k^2 \log n) \cdot m(Q, \epsilon^4, \delta/(k \log n)))$.

Our definition of a learning algorithm in Section 2.2 has nothing to do with whether the input is a k-means instance or a k-median instance, which is similar to k-means except that the cost of a cluster C with respect to a center q is defined as $\operatorname{cost}(C, q) := \sum_{p \in C} \mathrm{d}(p, q)$. In fact, it applies to any similar clustering scenario where the cost is defined in terms of the ℓ'th power ($\ell > 0$) of distances instead of squared distances. The analysis of Algorithm 1 can be adapted to any fixed ℓ once we have a suitable triangle inequality. (For example, in k-means (i.e., when $\ell = 2$), we use $(a+b)^2 \le (1+\epsilon)a^2 + (1+1/\epsilon)b^2$. When $\ell \le 1$, we can simply use the trivial triangle inequality $(a+b)^\ell \le a^\ell + b^\ell$.) Thus, for such clustering problems, Algorithm 1, with a slight modification on choice of radii in Step 4 and a little adjustment to the parameter γ, will give the same guarantees. Hence, we have the following theorem which is the formal version of Theorem 1. The proof follows from the analysis of Algorithm 1.

▶ **Theorem 6.** *Let K be a family of k-means (k-median) instances. Suppose that K is learnable with sample complexity $m(Q, \epsilon, \delta)$ using a zero sample error, non-inventive learning*

algorithm $\mathcal{A}_L$. Let $\mathcal{A}_\alpha$ be a constant-factor approximation algorithm, and let $\mathcal{A}_1$ be a PTAS for the 1-means (1-median) problem. There exists a polynomial-time algorithm that, given an instance $(P, Q, \mathrm{d}) \in K$, oracle access to same-cluster queries for some fixed optimal clustering $\mathcal{O}$, and parameters $(\epsilon, \delta) \in (0,1)^2$, outputs a clustering that, with probability at least $(1-\delta)$, is $(1-\epsilon)$-accurate with respect to $\mathcal{O}$, and simultaneously has a cost of at most $(1+\epsilon)OPT$. The algorithm uses $\mathcal{A}_L$, $\mathcal{A}_\alpha$, and $\mathcal{A}_1$ as subroutines. The number of same-cluster queries made by the algorithm is

1. $O((k^2 \log |P|) \cdot m(Q, \epsilon^4, \delta/(k \log |P|)))$ *for the k-means setting and*
2. $O((k^2 \log |P|) \cdot m(Q, \epsilon^2, \delta/(k \log |P|)))$ *for the k-median setting.*

For k-means and k-median instances in Euclidean space and those in finite metric spaces, there exist several constant-factor approximation algorithms (for example, Ahmadian et al. [1] and Kanungo et al. [13]). Solving the 1-means problem in Euclidean space is straightforward: The solution to $\operatorname{argmin}_{q \in \mathbb{R}^r} \operatorname{cost}(P, q)$ is simply $q = (\sum_{p \in P} p)/|P|$. For the k-median problem in Euclidean space, the problem of 1-median does not have an exact algorithm but several PTASes exist (for example, Cohen et al. [6]). In a finite metric space, to solve $\operatorname{argmin}_{q \in Q} \operatorname{cost}(P, q)$, we can simply try all possible $q \in Q$ in polynomial time, and this holds for the k-median setting as well. Thus, for Euclidean and finite metric space k-means and k-median instances that have no boundary points, Theorem 6, together with Theorem 4 and Theorem 5, gives efficient algorithms for $(1+\epsilon)$-approximate, $(1-\epsilon)$-accurate semi-supervised clustering.

4 Removing the Dependency on Problem Size in the Query Complexity for Euclidean k-Means

For the family of Euclidean k-means instances, the query complexity of Algorithm 1 suffers from a $\tilde{O}(\log n)$ dependency (where n is the number of points in the input k-means instance, and $\tilde{O}$ hides $\operatorname{poly}(\log \log n)$ factors) due to the repeated use of the learning algorithm $\mathcal{A}_L$. Specifically, we run $\mathcal{A}_L$ with a failure probability of $\delta/(k \log n)$, $O(\log n)$ times per cluster. Note that the sample complexity of $\mathcal{A}_L$ itself, in the case of Euclidean k-means instances, does not have this dependency.

In this section, we show that we can avoid this dependency on n using a slightly more involved algorithm at the cost of increasing the query complexity by an extra $\operatorname{poly}(k)$ factor. Nevertheless, this algorithm has superior performance when the size of the input instance (i.e., the number of points) is very large (when $\log n = \Omega(k^{10})$ for example).

Recall that, for a set $C \subset \mathbb{R}^r$, $\operatorname{cost}(C, y)$ is minimized when y is the centroid of C, denoted by $\mu(C) = (\sum_{x \in C} x)/|C|$. Define the fractional size of an optimal cluster O_i as the fraction of points that belong to O_i, i.e., the ratio $|O_i|/n$. Suppose we only want to get a good approximation for the cost, and that we know that all the clusters in the target solution have sufficiently large fractional sizes. In this case, naive uniform sampling will likely pick a large number of samples from each of the clusters. This observation, together with Lemma 7, allows us to approximate the centroid and the cost of each cluster to any given accuracy.

▶ **Lemma 7** (Lemma 1 and Lemma 2 of Inaba et al. [12]). *Let $(\epsilon, \delta) \in (0,1)^2$, let $m \geq 1/(\epsilon\delta)$ be a positive integer, and let $S = \{p_1, \ldots, p_m\}$ be a multiset of m i.i.d. samples from the uniform distribution over some finite set $C \subset \mathbb{R}^r$. With probability at least $(1-\delta)$, $\mathrm{d}^2(\mu(S), \mu(C)) \leq \epsilon \cdot \operatorname{cost}(C, \mu(C))/|C|$ and $\operatorname{cost}(C, \mu(S)) \leq (1+\epsilon) \operatorname{cost}(C, \mu(C))$.*

However, the above approach fails when some clusters in the optimal target solution contribute significantly to the cost, but have small fractional sizes (that is because uniform

sampling is not guaranteed to pick sufficient numbers of samples from the small clusters). Ailon et al. [2] circumvented this issue with an algorithm that iteratively approximates the centers of the clusters using a distance-based probability distribution (D^2-sampling). We will refer to their algorithm as $\mathcal{A}^*$.

Note that when it comes to accuracy, we can totally disregard clusters with small fractional sizes; we only have to correctly label a sufficiently large fraction of the points in large clusters. With this intuition, we present the outline of our algorithm.

Let $(P, \mathbb{R}^r, \mathrm{d})$ be a k-means instance in Euclidean space that has no boundary points. For simplicity, we refer to the instance $(P, \mathbb{R}^r, \mathrm{d})$ by just P where possible, as for Euclidean k-means, the other two parameters are fixed. We start with a naive uniform sampling step that gives a good approximation for the centers of large clusters. Starting with these centers, we run a slightly modified version of algorithm $\mathcal{A}^*$ to approximate the centers of the remaining small clusters. Thus, at this point, we have a clustering with a good cost and we know which clusters are large. We now run the learning algorithm $\mathcal{A}_L$ on input P and obtain a labeling of the points. For each point, we assign its final label based on (1) the label assigned to it by the learning algorithm $\mathcal{A}_L$, and (2) its proximity to large cluster centers. In particular, if the output of $\mathcal{A}_L$ decides that a point p should be in some large cluster i, and if p is sufficiently close to the approximate center for cluster i, we label it according to the learning output; otherwise, we label it according to its nearest approximate center. We show that this approach retains a cost that is close to the cost of the clustering output by $\mathcal{A}^*$. The accuracy guarantee comes from the facts that a large fraction of the points are sufficiently close to the centers of large clusters, and that $\mathcal{A}_L$ labels most of them correctly with a good probability.

We now review the key properties of algorithm $\mathcal{A}^*$ (the algorithm of Ailon et al. [2]). Let $0 < \epsilon < 1$. We say a k-means instance P is (k, ϵ)-irreducible if no $(k-1)$-means clustering gives an $(1+\epsilon)$-approximation for the k-means problem, i.e., if OPT^k denotes the optimal k-means cost of P, then P is (k, ϵ)-irreducible if $OPT^{k-1} > (1+\epsilon)OPT^k$. Suppose that P is (k, ϵ)-irreducible. Let $\mathcal{O} = \{O_1, \ldots, O_k\}$ be the target optimal clustering, and let $o_1, \ldots, o_k$ be the respective centers. Let $C_i = \{c_1, \ldots, c_i\}$ denote a set of i centers and let $Z(i)$ denote the following statement: There exists a set of i distinct indices $j_1, \ldots, j_i$ such that, for all $r \in [i]$, $\mathrm{cost}(O_{j_r}, c_r) \leq (1 + \epsilon/16)\,\mathrm{cost}(O_{j_r}, o_{j_r})$. To put it differently, $Z(i)$ says that C_i is a set of good candidate centers for i-many distinct clusters in the target optimal solution. Assuming P is (k, ϵ)-irreducible, the algorithm $\mathcal{A}^*$ yields a method to incrementally construct sets $C_1, \ldots, C_k$ (i.e., $C_{i+1} = C_i \cup \{c_{i+1}\}$) such that, conditioned on $Z(i)$ being true, $Z(i+1)$ is true with probability at least $(1 - 1/k)$. Now suppose that P is $(k, \epsilon/(4k))$-irreducible. Then $\mathcal{A}^*$ gives a $(1+\epsilon/(4 \cdot 16k))$-approximation for k-means with probability at least $(1-1/k)^k \geq 1/4$. Otherwise, $\mathcal{A}^*$ gives a $(1+\epsilon/(4 \cdot 16k))$-approximation for the i-means problem for some $i < k$, where i is the largest integer such that P is $(i, \epsilon/4k)$-irreducible. In the latter case, it will give a $(1 + \epsilon/(4 \cdot 16k))(1 + \epsilon/(4k))^{k-i}$-approximation with probability at least $1/4$. In either case, the output of $\mathcal{A}^*$ is a $(1+\epsilon)$-approximation.

In our algorithm, we first find the centers of large clusters using uniform sampling, and then run $\mathcal{A}^*$ to find the remaining centers. This allows us to know which clusters are large, which is a crucial information needed for the final labeling. Suppose that in the target optimal solution we have $k_0 \leq k$ clusters whose fractional sizes are at least ϵ/k. Note that k_0 is at least 1 due to the Pigeonhole Principle, since at least one cluster should have a fractional size of at least $1/k > \epsilon/k$. By Lemma 7, using uniform sampling, we can approximate the centroid of each of these large clusters with a good accuracy. Hence, we can have a set C_{k_0} of k_0 centers such that $Z(k_0)$ is true with probability $(1 - \delta)$. Afterwards, we use $\mathcal{A}^*$ to

Algorithm 2: Algorithm whose query complexity is independent of n

Input : Point set $P \subset \mathbb{R}^r$, the oracle access to $f_{\mathcal{O}}$, parameter k, accuracy parameter ϵ, failure probability δ, and algorithms $\mathcal{A}_{cost}$ and $\mathcal{A}_L$.

Output : The clustering $\hat{\mathcal{O}} = \{\hat{O}_1, \ldots, \hat{O}_k\}$ defined by the labeling $f_{\hat{\mathcal{O}}} : P \to [k]$ computed below. For each $i \in [k]$, the respective cluster center $\hat{o}_i$ is the centroid of $\hat{O}_i$.

1 Draw $Q_1(k, \epsilon, \delta)$ samples from P independently and uniformly at random, and query $f_{\mathcal{O}}$ to get their true cluster labels in $\mathcal{O}$. Denote the set of sampled points by S, and for all $i \in [k]$, denote the set of sampled points that belong to class i by S_i.

2 Let k' be the number of distinct cluster labels with more than $(\epsilon/(2k))Q_1(k, \epsilon, \delta)$ samples. Let $C_{k'} := \{\mu(S_i) : |S_i| > (\epsilon/(2k))Q_1(k, \epsilon, \delta)\}$. Without loss of generality, assume that the class labels for centers in $C_{k'}$ are $1, \ldots, k'$.

3 Run the algorithm $\mathcal{A}_{cost}$, starting from $C_{k'}$ as the partial set of centers. This takes $Q_2(k, \epsilon, \delta)$ more queries. Let $C_k = \{c_1, \ldots, c_k\}$ be the output, and let OPT^* be the cost of the clustering obtained by assigning each point to its nearest c_i.

4 Use the PAC learning algorithm $\mathcal{A}_L$ on $Q_3(k, r, \epsilon^4/k, \delta)$ uniform i.i.d. samples from P to learn a classifier for the k classes that is $(1 - \epsilon^4/k)$-accurate with probability at least $(1 - \delta)$. Let $H_1, \ldots, H_k$ be the sets of points that are labeled $1, \ldots, k$ respectively by the classifier.

5 Output the clustering $\hat{\mathcal{O}}$ defined by the following labeling function: for each $i \in [k']$ and $p \in H_i$ such that $\mathrm{d}^2(p, c_i) \leq kOPT^*/(n\epsilon^3)$, set $f_{\hat{\mathcal{O}}}(p) = i$. For any other point p, set $f_{\hat{\mathcal{O}}}(p) = i$ if the nearest cluster center to p in C_k is c_i.

incrementally construct $C_{k_0+1}, \ldots, C_k$. Conditioned on $Z(k_0)$ being true, the output C_k will be a $(1 + \epsilon)$-approximation with probability $(1 - 1/k)^{k-k_0} \geq (1 - 1/k)^k \geq 1/4$ for $k \geq 2$. However, by independently running this incremental construction $O(\log(1/\delta))$ times and choosing the set of centers with the minimum total cost, we can boost this probability to $(1 - \delta)$. This observation gives the following generalization of Theorem 10 of Ailon et al. [2].

▶ **Theorem 8.** *Consider a Euclidean k-means instance $(P, \mathbb{R}^r, \mathrm{d})$. Let $O_1, \ldots, O_k$ be a fixed optimal clustering with respective centers $o_1, \ldots, o_k$. Let $k_0 \leq k$ and let $C_{k_0} = \{c_1, \ldots, c_{k_0}\}$ be a set of points such that, with probability at least p_0, $\mathrm{cost}(O_i, c_i) \leq (1 + \epsilon/(64k))\,\mathrm{cost}(O_i, o_i)$ for all $i \in [k_0]$. There exists an algorithm $\mathcal{A}_{cost}$ that, given P, C_{k_0}, and parameters $(\epsilon, \delta) \in (0, 1)^2$ as input, outputs a set of centers $C_k = C_{k_0} \cup \{c_{k_0+1}, \ldots, c_k\}$ such that $\sum_{i\in[k]} \mathrm{cost}(O_i, c_i) \leq (1 + \epsilon) \sum_{i\in[k]} \mathrm{cost}(O_i, o_i)$ with probability at least $p_0(1 - \delta)$. Moreover, $\mathcal{A}_{cost}$ uses $O((k^9/\epsilon^4)\log(1/\delta))$ same-cluster queries and runs in time $O(nr(k^9/\epsilon^4)\log(1/\delta))$.*

Theorem 8 implies a method to get a good approximation for the cost that also reveals which clusters are large. Specifically, we first perform uniform sampling over the whole set P and approximate the centers of the large clusters. If we get a sufficient number of samples, the approximate centers will satisfy the precondition of Theorem 8 with a good probability. Thus, using algorithm $\mathcal{A}_{cost}$ from Theorem 8, we get the desired approximation for the cost. What remains now is to use PAC learning and to appropriately label the points according to the learning outcome.

We present the pseudo-code of our algorithm in Algorithm 2, where we use the algorithm $\mathcal{A}_{cost}$ from Theorem 8 and the learning algorithm $\mathcal{A}_L$ guaranteed by Theorem 4. In Algorithm 2, $Q_1(k, \epsilon, \delta) = 256k^3/(\epsilon^2\delta)$ is the number of samples needed to ensure that we pick a sufficient number of samples from each of the clusters with fractional sizes of at least ϵ/k, $Q_2(k, \epsilon, \delta)$ is the sample complexity of the algorithm $\mathcal{A}_{cost}$, and $Q_3(k, r, \epsilon, \delta) = m(\mathbb{R}^r, \epsilon, \delta)$ is the sample complexity of the learning algorithm $\mathcal{A}_L$ for an error ϵ and a failure probability δ. As with Algorithm 1, the center that a point is assigned to in the final output may not be the closest center to that point.

With probability at least $(1 - \delta)$, the output of Algorithm 2 is $(1 + \epsilon)$-approximate and $(1 - \epsilon)$-accurate (refer to the full version for the complete analysis). As for the claim on the

query complexity, recall that we only need $O(k)$ same-cluster queries per single $f_{\mathcal{O}}$ query, and Algorithm 2 makes a total number of $Q_1(k,\epsilon,\delta) + Q_2(k,\epsilon,\delta) + Q_3(k,r,\epsilon^4/k,\delta)$ queries to $f_{\mathcal{O}}$. This observation together with the analysis of Algorithm 2 proves Theorem 2. We remark that the query complexity $Q_3(k,r,\epsilon^4/k,\delta)$ for learning Euclidean k-means instances is independent of P.

References

1 Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 61–72, 2017.

2 Nir Ailon, Anup Bhattacharya, Ragesh Jaiswal, and Amit Kumar. Approximate clustering with same-cluster queries. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 40:1–40:21, 2018. doi:10.4230/LIPIcs.ITCS.2018.40.

3 Hassan Ashtiani, Shrinu Kushagra, and Shai Ben-David. Clustering with same-cluster queries. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 3224–3232, USA, 2016. Curran Associates Inc. URL: http://dl.acm.org/citation.cfm?id=3157382.3157458.

4 P. Awasthi, A. Blum, and O. Sheffet. Stability yields a PTAS for k-median and k-means clustering. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 309–318, Oct 2010. doi:10.1109/FOCS.2010.36.

5 Maria-Florina Balcan, Avrim Blum, and Anupam Gupta. Clustering under approximation stability. *J. ACM*, 60(2):8:1–8:34, 2013. doi:10.1145/2450142.2450144.

6 Michael B. Cohen, Yin Tat Lee, Gary Miller, Jakub Pachocki, and Aaron Sidford. Geometric median in nearly linear time. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 9–21, New York, NY, USA, 2016. ACM. doi:10.1145/2897518.2897647.

7 Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in euclidean and minor-free metrics. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 353–364, 2016.

8 S. Dasgupta. *The Hardness of K-means Clustering.* Technical report (University of California, San Diego. Department of Computer Science and Engineering). Department of Computer Science and Engineering, University of California, San Diego, 2008. URL: https://books.google.ch/books?id=riJuAQAACAAJ.

9 Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k-means clustering based on weak coresets. In *Proceedings of the Twenty-third Annual Symposium on Computational Geometry*, SCG '07, pages 11–18, New York, NY, USA, 2007. ACM. doi:10.1145/1247069.1247072.

10 Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Local search yields a PTAS for k-means in doubling metrics. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 365–374, 2016.

11 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 291–300. ACM, 2004.

12 Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering: (extended abstract). In *Proceedings of*

the Tenth Annual Symposium on Computational Geometry, SCG '94, pages 332–339, New York, NY, USA, 1994. ACM. doi:10.1145/177424.178042.

13 Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of the Eighteenth Annual Symposium on Computational Geometry*, SCG '02, pages 10–18, New York, NY, USA, 2002. ACM. doi:10.1145/513400.513402.

14 Euiwoong Lee, Melanie Schmidt, and John Wright. Improved and simplified inapproximability for k-means. *Inf. Process. Lett.*, 120:40–43, 2017.

High Probability Frequency Moment Sketches

Sumit Ganguly
Indian Institute of Technology, Kanpur, India
sganguly@cse.iitk.ac.in

David P. Woodruff
Carnegie Mellon University, School of Computing, Pittsburg, USA
dwoodruf@cs.cmu.edu

Abstract

We consider the problem of sketching the p-th frequency moment of a vector, $p > 2$, with multiplicative error at most $1 \pm \epsilon$ and *with high confidence* $1 - \delta$. Despite the long sequence of work on this problem, tight bounds on this quantity are only known for constant δ. While one can obtain an upper bound with error probability δ by repeating a sketching algorithm with constant error probability $O(\log(1/\delta))$ times in parallel, and taking the median of the outputs, we show this is a suboptimal algorithm! Namely, we show *optimal* upper and lower bounds of $\Theta(n^{1-2/p}\log(1/\delta) + n^{1-2/p}\log^{2/p}(1/\delta)\log n)$ on the sketching dimension, for any constant approximation. Our result should be contrasted with results for estimating frequency moments for $1 \leq p \leq 2$, for which we show the optimal algorithm for general δ is obtained by repeating the optimal algorithm for constant error probability $O(\log(1/\delta))$ times and taking the median output. We also obtain a matching lower bound for this problem, up to constant factors.

2012 ACM Subject Classification Theory of computation → Lower bounds and information complexity

Keywords and phrases Data Streams, Frequency Moments, High Confidence

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.58

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.10885`.

1 Introduction

The frequency moments problem is a very well-studied and foundational problem in the data stream literature. In the data stream model, an algorithm may use only sub-linear memory and a single pass over the data to summarize a data stream that appears as a sequence of incremental updates. A data stream may be viewed as a sequence of m records of the form $((i_1, v_1), (i_2, v_2), \ldots, (i_m, v_m))$, where, $i_j \in [n] = \{1, 2, \ldots, n\}$ and $v_j \in \mathbb{R}$. The record (i_j, v_j) changes the i_jth coordinate x_{i_j} of an underlying n-dimensional vector x to $x_{i_j} + v_j$. Equivalently, for $i \in [n]$, $x_i = \sum_{j:i_j=i} v_j$. Note that v_j may be positive or negative, which corresponds to the so-called turnstile model in data streams. Also, the i-th coordinate of x is sometimes referred to as the *frequency* of item i, though note that it can be negative in the turnstile model. The p-th moment of x is defined to be $F_p = \sum_{i\in[n]}|x_i|^p$, for a real number $p \geq 0$, which for $p \geq 1$ corresponds to the p-th power of the ℓ_p-norm $\|x\|_p^p$ of x.

The F_p estimation problem with approximation parameter ϵ and failure probability δ is: design an algorithm that makes one pass over the input stream and returns $\hat{F}_p$ such that $\Pr\left[|\hat{F}_p - F_p| \leq \epsilon F_p\right] \geq 1 - \delta$. Such an algorithm is also referred to as an (ϵ, δ)-approximation of F_p. This is a problem that is among the ones that has received the most attention

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 58; pp. 58:1–58:15


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Here, $g(p,n) = \min_{c \text{ constant}} g_c(n)$, where $g_1(n) = \log n$, $g_c(n) = \log(g_{c-1}(n))/(1-2/p)$. We start the upper bound timeline with [19], since that is the first work which achieved an exponent of $1-2/p$ for n. For earlier work which achieved worse exponents for n, see [1, 12, 14, 15].

F_p Algorithm	Sketching Dimension
[19]	$O(n^{1-2/p}\epsilon^{-O(1)}\log^{O(1)} n\log(1/\delta))$
[7]	$O(n^{1-2/p}\epsilon^{-2-4/p}\log n\log(M)\log(1/\delta))$
[31]	$O(n^{1-2/p}\epsilon^{-O(1)}\log^{O(1)} n\log(1/\delta))$
[3]	$O(n^{1-2/p}\epsilon^{-2-6/p}\log n\log(1/\delta))$
[8]	$O(n^{1-2/p}\epsilon^{-2-4/p}\log n\cdot g(p,n)\log(1/\delta))$
[2]	$O(n^{1-2/p}\log n\epsilon^{-O(1)}\log(1/\delta))$
[16], **Best upper bound**	$O(n^{1-2/p}\epsilon^{-2}\log(1/\delta)+n^{1-2/p}\epsilon^{-4/p}\log n\log(1/\delta))$

in the data stream literature, and we only give a partial list of work on this problem [1, 2, 3, 4, 6, 7, 8, 10, 12, 14, 15, 16, 20, 19, 25, 26, 27, 30, 31, 34].

We study the class of algorithms based on linear sketches, which store only a sketch $S \cdot x$ of the input vector x and a (possibly randomized) matrix A. This model is well-studied, both for the problem of estimating norms and frequency moments [4, 18, 30, 32], and for other problems such as estimating matrix norms [29], and matching size [5, 28]. The efficiency is measured in terms of the *sketching dimension* which is the maximum number of rows of a matrix S used by the algorithm. Since the algorithm is randomized, it may choose different S based on its randomness, so the maximum is taken over its randomness. Linear sketches are particularly useful for data streams since given an update (i_j, v_j), one can update Sx as $S(x + v_j e_{i_j}) = Sx + Sv_j e_{i_j}$, where e_{i_j} is the standard unit vector in the i_j-th direction. They are also used in distributed environments, since given $S \cdot x$ and618 $S \cdot y$, one can add these to obtain $S \cdot (x + y)$, the sketch of $x + y$.

When $0 < p \leq 2$, one can achieve a sketching dimension of $O(\epsilon^{-2}\log(1/\delta))$ independent of n [1, 25, 27], while for $p = 0$ the sketching dimension is $O(\epsilon^{-2}(\log(1/\epsilon) + \log\log n)\log(1/\delta))$ [26]. For $p = 2$ there is a sketching lower bound of $\Omega(\epsilon^{-2}\log(1/\delta))$ [24], which implies an optimal algorithm for general δ is to run an optimal algorithm with error probability $1/3$ and take the median of $O(\log(1/\delta))$ independent repetitions. As a side result, we show in the full version a lower bound of $\Omega(\epsilon^{-2}\log(1/\delta))$ for any $1 \leq p < 2$, which shows this strategy of amplifying the success probability by $O(\log 1/\delta)$ independent repetitions is also optimal for any $1 \leq p < 2$.

Perhaps surprisingly, for $p > 2$, the sketching dimension needs to be polynomial in n, as first shown in [32], with the best known lower bounds being $\Omega(n^{1-2/p}\log n)$ [4] for constant ϵ and δ, and $\Omega(n^{1-2/p}\epsilon^{-2})$ for constant δ [30]. Regarding upper bounds, we present the long list of bounds in Table 1. The best known upper bound is $O(n^{1-2/p}\epsilon^{-2}\log(1/\delta) + n^{1-2/p}\epsilon^{-4/p}\log n\log(1/\delta))$ [16]. This is tight only when ϵ and δ are constant, in which case it matches [4], or when δ is constant and $\epsilon < 1/\text{poly}(\log n)$, since it matches [30].

1.1 Our Contributions

In this work, we show *optimal* upper and lower bounds of $\Theta(n^{1-2/p}\log(1/\delta) + n^{1-2/p}\log^{2/p}(1/\delta)\log n)$ on the sketching dimension for F_p-estimation, for any $p > 2$, and for any constant ϵ. Our upper bound shows, perhaps surprisingly, that the optimal bound is *not* to run $O(\log(1/\delta))$ independent repetitions of a constant success probability algorithm and report the median of the outputs. Indeed, such an algorithm would give a worse $O(n^{1-2/p}\log(1/\delta)\log n)$ sketching dimension.

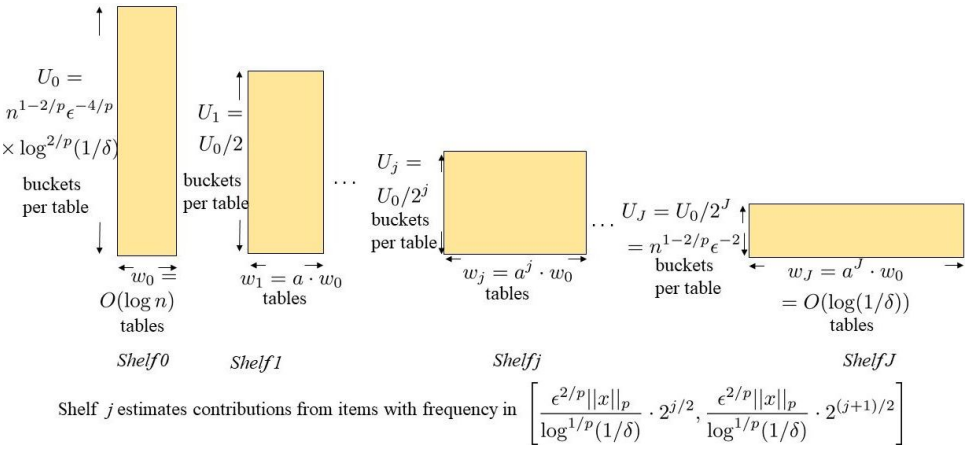


Figure 1 Shelf structure and level sets for each shelf index j whose contribution to F_p is estimated accurately.

For general ϵ, our upper bound is $O(n^{1-2/p}\epsilon^{-2}\log(1/\delta) + n^{1-2/p}\epsilon^{-4/p}\log^{2/p}(1/\delta)\log n)$ and our lower bound is $\Omega(n^{1-2/p}\epsilon^{-2}\log(1/\delta) + n^{1-2/p}\epsilon^{-2/p}\log^{2/p}(1/\delta)\log n)$, which differ by at most an $\epsilon^{-2/p}$ factor. Our results thus come close to resolving the complexity for general ϵ as well.

Our results should be contrasted to $1 \le p \le 2$, for which the optimal sketching dimension for such p is $\Theta(\epsilon^{-2}\log(1/\delta))$, and so for these p it is optimal to run $O(\log(1/\delta))$ independent repetitions of a constant probability algorithm. Here we strengthen the $\Omega(\epsilon^{-2}\log(1/\delta))$ bound for $p = 2$ of [24] by showing the same bound for $1 \le p \le 2$.

1.1.1 Overview of Upper Bound

In order to obtain a confidence of $1 - \delta$, we use the $d = \lceil \log(1/\delta) \rceil$th moment of an estimate $\hat{F}_p$ of F_p. Since we are unable to use the dth moment of the Taylor polynomial estimator of [17], we employ a different estimator X_i for estimating individual coordinates $|x_i|$ and use it as X_i^p to estimate $|x_i|^p$. This estimator is based on (a) using random qth roots of unity for sketches instead of standard Rademacher variables, and (b) taking the *average* of the estimates from those tables where the item does not collide with the set of top-k estimated heavy hitters.

The Shelf Structure. The algorithm uses two structures, namely, a GHSS-like structure from [17] and a new shelf structure , which is our main algorithmic novelty (both formally defined later). The shelf structure is necessary when the failure probability is $\delta = n^{-\omega(1)}$; otherwise, for $\delta = n^{-\Theta(1)}$, somewhat surprisingly the GHSS structure of [17] alone suffices with parameter $C = n^{1-2/p}(\epsilon^{-2}\log(1/\delta)/\log(n) + \epsilon^{-4/p}\log^{2/p}(1/\delta))$ and number of measurements $O(C\log n)$, which requires an intricate d-th moment analysis of the GHSS structure.

The shelf structure is partitioned into shelves, indexed from $j = 0, \ldots, J$, for a value J which is specified below. Each shelf consists of a pair of CountSketch like structures, HH_j and AvgEst_j. The number of buckets in the tables of the jth shelf is H_j and the number of tables in the jth shelf of the HH_j structure is w_j and of the AvgEst_j structure is $2w_j$. We set $H_J = \Theta(n^{1-2/p}\epsilon^{-2})$ and $w_J = \Theta(\log(1/\delta))$, while $H_0 = \Theta(n^{1-2/p}\epsilon^{-4/p}\log^{2/p}(1/\delta))$ and $w_0 = s = \Theta(\log n)$.

The input vector x is provided as input to all the shelves' structures. The table height $H_j = H_0 b^j$ decays geometrically with parameter $0 < b < 1$ and the table width $w_j = w_0 a^j$ increases geometrically with parameter $a > 1$. Note that the parameters a and b determine J. By requiring that $|1 - ab| = \Omega(1)$, we ensure that the total number of measurements of the shelf structure is $\sum_{j=0}^{J} H_j w_j = O(H_0 w_0 + H_J w_J)$, no matter which value of J we choose. For the shelf structure, frequency-wise thresholds are defined as $U_j = O(\hat{F}_2/H_j)^{1/2}$, for $j = 0, 1, \ldots, J$. The shelf frequency group corresponding to shelf j is $S_j = [U_j, U_{j+1})$, where, $U_{J+1} = \infty$ and $U_0 = T_0$. We sometimes conflate S_j with the set of items whose frequency belongs to S_j. The frequency group G_0 is defined as $[T_0, U_1]$ and coincides with S_0. See Figure 1.

So why a shelf structure? Suppose for simplicity that ϵ is a constant. Consider a vector x which has a constant number of "large" coordinates of value $\Theta(n^{1/p})$, and $\Theta(n)$ remaining "small" coordinates of absolute value $O(1)$. Then we need to find all the large coordinates to accurately estimate F_p up to a small constant factor. This is well-known to be possible with $\Theta(n^{1-2/p})$ buckets in the J-th shelf, since with probability $1 - \delta$, each of the large coordinates will not collide with any other large coordinate in more than a small constant fraction of tables. Note that in each table, in each bucket containing a large coordinate, the "noise" in the bucket from small coordinates will be $Cn^{1/p}$ for an arbitrarily small constant $C > 0$ with constant probability, and so this will happen in most buckets containing a large coordinate in most tables with probability $1 - \delta$.

However, now consider a vector x which has $\Theta(\log(1/\delta))$ "large-ish" coordinates of value $\Theta(n^{1/p}/\log^{1/p}(1/\delta))$, and $\Theta(n)$ remaining "small" coordinates of absolute value $O(1)$, as before. Then we again need to find most of the "large-ish" coordinates to accurately estimate F_p up to a constant factor. We also *cannot* subsample and try to estimate how many large-ish coordinates there are from a subsample. Indeed, since there are only $O(\log(1/\delta))$ total large-ish coordinates, sub-sampling would not accurately estimate this total with probability at least $1 - \delta$. However, to find these "large-ish" coordinates, we need to increase the number of buckets from $\Theta(n^{1-2/p})$ to $\Theta(n^{1-2/p} \cdot \log^{2/p}(1/\delta))$ just so that in a bucket containing one of these coordinates, with constant probability the noise will not be too large. But if we then want this to happen for a $1 - \delta$ fraction of tables, we still need $\Theta(\log(1/\delta))$ tables, which gives overall $\Theta(n^{1-2/p} \cdot \log^{1+2/p}(1/\delta))$ measurements, which is above our desired total of $O(n^{1-2/p}(\log(1/\delta) + \log(n)\log^{2/p}(1/\delta)))$ measurements.

So what went wrong? The key idea in our analysis is to relax the requirement of trying to recover all the larg-ish coordinates with probability $1 - \delta$. Suppose instead of $\Theta(\log(1/\delta))$ tables we just use $\Theta(\log n)$ tables. Then with probability $1 - 1/n$, there may be two large-ish coordinates which collide and cancel with each other in every single table, and we have no way of recovering them. However, we are able to show that with probability $1 - \delta$, only $O(\log(1/\delta)/\log n)$ large-ish coordinates will fall into this category, and neglecting this roughly $(1 - 1/\log n)$ fraction of the large-ish coordinates will not affect our estimate of F_p by more than a constant factor. And indeed, our 0-th shelf has exactly $\Theta(n^{1-2/p} \cdot \log^{2/p}(1/\delta))$ buckets and $\Theta(\log n)$ tables, so is exactly suited for finding these large-ish coordinates. In general, we can show that one of our shelves will be able to handle every vector with coordinates of magnitude between the large and large-ish coordinates. Again, by choosing the shelf structure carefully, the total number of measurements is dominated by that in the zero-th plus the J-th shelf, giving us $O(n^{1-2/p}(\log(1/\delta) + \log(n)\log^{2/p}(1/\delta)))$ total measurements, and explaining where the $\log^{2/p}(1/\delta)$ in the upper bound comes from.

The Non-Large-ish Coordinates. Our shelves are designed to estimate the contribution to F_p from all coordinates of absolute value at least $\Theta(n^{1/p}/\log^{1/p}(1/\delta))$. For coordinates of smaller value, we can now afford to sub-sample and apply the same 0-th shelf structure to estimate their contribution to F_p. We apply the GHSS structure, which is analogous to the structure presented in [17] and has $L+1$ levels corresponding to $l = 0, \ldots, L$, and consists of a pair of CountSketch like structures HH_l and AvgEst_l at each level. The sub-sampling technique and the associated frequency-wise thresholds and frequency groups are defined analogously (with new parameters) to [17].

A notable difference with [17] is that the AvgEst structures in the GHSS and shelf structures use complex qth roots of unity and return the average of table estimates instead of the median of table estimates used by CountSketch, which are novelties in this context, though have been used for other data stream problems [23]. We have that $\mathbf{E}[X_i^p] = |x_i|^p(1 \pm n^{-\Omega(1)})$ for our estimator X_i of $|x_i|$, and thus X_i^p provides a nearly unbiased estimator of $|x_i|^p$. Additionally, we use averaging in the definition of X_i instead of the median to allow for a tractable, though intricate calculation of the d-th moment of the sum of the p-th powers of X_i.

1.1.2 Overview of Lower Bounds

We give an overview for the case of constant ϵ. In both cases we start by applying Yao's minimax principle for which we fix S and then design a pair of distributions α and β which must be distinguished by an (ϵ, δ)-approximation algorithm for F_p. We can also assume the rows of S are orthonormal, since a change of basis to the row space of S can always be applied in post-processing.

Our $\Omega(n^{1-2/p}\epsilon^{-2/p}(\log^{2/p} 1/\delta)\log n)$ bound. This is our technically more involved lower bound. We first upper bound the variation distance using the χ^2-divergence as in [4] and work only with the latter. We let $\alpha = N(0, I_n)$ be an n-dimensional isotropic Gaussian distribution, while β is a distribution formed by sampling an $x \sim N(0, I_n)$, together with a random subset $T \subset [n]$ of size $O(\log(1/\delta))$, and outputting $z = x + \sum_{i \in T}(Cn^{1/p}/t^{1/p})e_i$, where e_i is the i-th standard unit vector and $C > 0$ is a constant. For $y \sim \alpha$ and $z \sim \beta$, one can show that with probability $1 - O(\delta)$, one has that $\|z\|_p^p$ is a constant factor larger than $\|y\|_p^p$, since $\|y\|_p^p$ and $\|x\|_p^p$ are concentrated at $\Theta(n)$, while $\sum_{i \in T} C^p n/t = \Theta(n)$.

A common technique in upper bounds, including our own, is the notion of subsampling, whereby a random fraction of roughly $1/2^i$ of the n coordinates are sampled, for each value of $i \in O(\log n)$, and information is then gathered for each i and combined into an overall estimate of F_p. We choose our hard distributions so that *subsampling does not help.* Indeed, if one subsamples half of the coordinates of $z \sim \beta$, with probability $\Omega(\delta)$ all of the coordinates in T will be removed, at which point z is indistinguishable from $y \sim \alpha$. Therefore, our pair of distributions suggests itself as being hard for $(\Theta(1), \delta)$-approximate F_p algorithms.

What drives our analysis is conditioning our distributions on an event $\mathcal{G}$ which only happens with probability $\Omega(\delta)$. Note that for any algorithm which can distinguish samples from α from those from β with probability at least $1 - \delta$, it must still have probability $9/10$, say, of distinguishing the distributions given an event $\mathcal{G}$ which occurs for samples drawn from β. The event $\mathcal{G}$ corresponds to every $i \in T$ having the property that the corresponding column S_i of our sketching matrix S has squared length at most $2r/n$, where r is the number of rows of S. By a Markov bound, half of the columns of S have this property, and since T has size $O(\log 1/\delta)$, with probability $\Omega(\delta)$, event $\mathcal{G}$ occurs.

We analyze the χ^2-divergence of the distributions α and β conditioned on $\mathcal{G}$. One technique helpful for this is an equality that we show in the full version, which states that for

p a distribution on $\mathbb{R}^n$, that $\chi^2(N(0,I_n)*p, N(0,I_n)) = \mathbf{E}[e^{\langle X,X'\rangle}]-1$, where X and X' are independently drawn from p. This equality was used in [4, 29, 35] among other places. In our case, the inner product of X and X' corresponds to an inner product P of two independent random sums of t columns of S, restricted to only those columns with squared length at most $2r/n$. Let the t columns forming X be denoted by T and the t columns forming X' be denoted by U.

Critical to our analysis is bounding $\mathbf{E}[P^j]$ for large powers of j, as shown in the lemma the full version. One can think of indexing the rows of S^TS by T and the columns of S^TS by U, where S^TS is an $n\times n$ matrix. Let M denote the resulting submatrix. The inner product of interest is then $e_T^TMe_U$, where $e_T=\sum_{i\in T}e_i$ and $e_U=\sum_{i\in U}e_i$.

Our bound, given in the above-referred to lemma in the full version, is very sensitive to minor changes. Indeed, if instead of showing $\mathbf{E}[P^j]\leq\left(\frac{t^2}{r^{1/2}}\right)\cdot\left(\frac{16r}{n}\right)^j$, we had shown $\mathbf{E}[P^j]\leq\left(\frac{t^2}{r^{1/2}}\right)\cdot\left(\frac{16rt}{n}\right)^j$ or $\mathbf{E}[P^j]\leq\left(\frac{t^2}{r^{1/2}}\right)\cdot\left(\frac{16r\log n}{n}\right)^j$, our resulting bound for the χ^2-divergence would be larger than 1. For instance, a natural approach is to instead consider $e_T=\sum_{i\in T}\sigma_ie_i$ and $e_U=\sum_{i\in U}\sigma_ie_i$ where the σ_i are independent random signs (i.e., $\Pr[\sigma_i=1]=\Pr[\sigma_i=-1]=1/2$), which would correspond to redefining the distribution β above to sample $z=x+\sum_{i\in T}(Cn^{1/p}/t^{1/p})\sigma_ie_i$. Without further conditioning the σ_i variables, the χ^2-divergence can be as large as $n^{\Theta(\log(1/\delta))}$. This is because with probability roughly 2^{-2t}, over the choice of the σ_i, one has $\sum_{i\in T}\sigma_ie_i$ and $\sum_{i\in U}\sigma_ie_i$ both being very well aligned with the top singular vector of M (if say, S were a random matrix with orthonormal rows), at which point our desired inner product is too large. Instead, by setting all $\sigma_i=1$, that is, by considering $e_T=\sum_{i\in T}e_i$ and $e_U=\sum_{i\in U}e_i$ as we do, we rule out this possibility.

We prove our lemma by expanding $\mathbf{E}[P^j]$ into a sum of products, each having the form $\prod_{w=1}^j|\langle S_{a_w},S_{b_w}\rangle|$ where the S_{a_w},S_{b_w} are columns of S. One thing that matters in such products is the multiplicities of duplicate columns that appear in a product. We split the summation by what we call *y-patterns*. We can think of a y-pattern as a partition of $\{1,2,\ldots,j\}$ into y non-empty pieces. We can also define a z-pattern as a partition of $\{1,2,\ldots,j\}$ into z non-empty pieces. We analyze the expectation for a particular pair P,Q, where P is a y-pattern and Q is a z-pattern for some $y,z\in\{1,2,\ldots,j\}$, that is, we only sum over pairs of j-tuples $a_1,\ldots,a_j$ and $b_1,\ldots,b_j$ for which for each non-empty piece $\{d_1,\ldots,d_\ell\}$ in P, where $d_i\in\{1,2,\ldots,j\}$ for all i and $\ell\leq j$, we have $a_{d_1}=a_{d_2}=\cdots=a_{d_\ell}$. Similarly for each $\{e_1,\ldots,e_m\}$ in Q, where $e_i\in\{1,2,\ldots,j\}$ for all i and $m\leq j$, we have $b_{e_1}=b_{e_2}=\cdots=b_{e_m}$. We also require if $d,d'\in\{1,2,\ldots,j\}$ are in different pieces of P, then $a_d\neq a_{d'}$. Similarly, if $e,e'\in\{1,2,\ldots,j\}$ are in different pieces of Q, then $b_e\neq b_{e'}$. Thus, each pair of j-tuples is valid for exactly one pair P,Q of patterns.

The valid pairs of j-tuples for P and Q define a bipartite multi-graph as follows. In the left partition we create a node for each non-empty piece of P, and in the right partition we create a node for each non-empty piece of Q. We include an edge from a node a in the left to a node b in the right if $i\in a$ and $i\in b$ for some $i\in\{1,2,\ldots,j\}$. If there is more than one such i, we include an edge with multiplicity corresponding to the number of such i. This bipartite graph only depends on P and Q. We consider a maximum matching in this multi-graph, and we upper bound the contribution of valid pairs for P and Q based on that matching. By summing over all pairs P,Q, we obtain our bound on $\mathbf{E}[P^j]$.

Our $\Omega(n^{1-2/p}\epsilon^{-2}\log(1/\delta))$ bound. This bound uses the same distributions α and β as in [30], where an $\Omega(n^{1-2/p}\epsilon^{-2})$ bound was shown, but we strengthen it to hold for general δ. To do so, we use an exact characterization of the variation distance between multi-variate

Gaussians with shifted mean by relating it to the univariate case (given in the full version), and a strong concentration of bounded Lipshitz functions with respect to the Euclidean norm (given in the full version). These enable us to show with probability $1 - O(\delta)$, vectors sampled from α and β have $\l_p$-norm differing by a $1 + \epsilon$ factor. By the definition of α and β, we can then reduce the problem to distinguishing an isotropic Gaussian from an isotropic Gaussian plus a small multiple of a fixed column of S, which typically has small norm since S has orthonormal rows. We then apply a bound as derived above (see full version).

Our $\Omega(\epsilon^{-2}\log(1/\delta))$ bound for $1 \leq p < 2$. This lower bound uses similar techniques to our lower bound of $\Omega(n^{1-2/p}\epsilon^{-2}\log(1/\delta))$, but considers distinguishing an isotropic Gaussian $N(0, I_n)$ from an $N(0, (1+\epsilon)I_n)$ random variable. Here we set $n = \Theta(\epsilon^{-2}\log(1/\delta))$, and show the p-norms of samples from the two distributions differ by a $(1+\epsilon)$-factor with probability $1 - \delta$. Using that S has orthonormal rows, the images of the two distributions under our sketching matrix S correspond to $N(0, I_r)$ and $N(0, (1+\epsilon)I_r)$, where r is the number of rows of S. The result then follows by using the product structure of Hellinger distance.

2 Our Lower Bounds

We first describe our lower bounds in a little more detail. Due to space constraints, we present a highly abridged version without proofs here (see full version). We defer both our $\Omega(n^{1-2/p}\epsilon^{-2}\log(1/\delta))$ lower bound for $p > 2$ and our $\Omega(\epsilon^{-2}\log(1/\delta))$ lower bound for $1 \leq p < 2$ entirely to the full version. Here we focus on our lower bound of $\Omega(n^{1-2/p}\epsilon^{-2/p}(\log^{2/p}(1/\delta))\log n)$ for $p > 2$. See also Section 1 for an overview of all of our lower bounds.

We assume δ-**Bound4**, which is that $\log(1/\delta) \leq (n^{1-2/p}\varepsilon^{-2/p}(\log^{2/p} 1/\delta)\log n)^{1/4}n^{-c'}$, for a sufficiently small constant $c' > 0$. Since $p > 2$ is an absolute constant, independent of n, this just states that $\delta \geq 2^{-n^{c''}}$ for a sufficiently small constant $c'' > 0$. There are other bounds - δ-**Bound1**, δ-**Bound2**, and δ-**Bound3** - see the full version, but these are not assumptions but rather implied by relations between the various parameters (e.g., otherwise the $\Omega(n^{1-2/p}\epsilon^{-2}\log(1/\delta))$ lower bound is stronger).

Let p and q be probability density functions of continuous distributions. The χ^2-divergence from p to q is $\chi^2(p, q) = \int_x \left(\frac{p(x)}{q(x)} - 1\right)^2 q(x)dx$.

▶ **Fact 1.** *([33], p.90) For any two distributions p and q, we have $D_{TV}(p, q) \leq \sqrt{\chi^2(p, q)}$.*

We need a fact about the distance between a Gaussian location mixture to a Gaussian distribution.

▶ **Fact 2.** *(p.97 of [21]) Let p be a distribution on $\mathbb{R}^n$. Then $\chi^2(N(0, I_n) * p, N(0, I_n)) = \mathbf{E}[e^{\langle X, X'\rangle}] - 1$, where X and X' are independently drawn from p.*

Let T be a sample of $t \stackrel{\text{def}}{=} \log_3(1/\sqrt{\delta})$ coordinates $i \in [n]$ without replacement.
Case 1: Suppose $y \sim N(0, I_n)$, and let α' be the distribution of y.
Case 2: Let $z = x + \sum_{i \in T} \frac{C'\epsilon^{1/p}E_{n-t}}{t^{1/p}}e_i$, where $x \sim N(0, I_n)$ and $E_{n-t} = \mathbf{E}_{x \sim N(0, I_{n-t})}[\|x\|_p]$. Note that x and T are independent. Also, $C' > 0$ is a sufficiently large constant. Let β' be the distribution of z.

In the full version we show that for the sketching algorithm to be correct, $D_{TV}(\bar{\alpha'}, \bar{\beta'}) \geq 1 - 2\delta$, where $\bar{\alpha'}$ is the distribution of $S \cdot y$ for $y \sim \alpha'$ and $\bar{\beta'}$ is the distribution of $S \cdot z$ for $z \sim \beta'$.

Fix an $r \times n$ matrix S with orthonormal rows. Important to our proof will be the existence of a subset W of $n/2$ of the columns for which $\|S_i\|^2 \leq 2r/n$ for all $i \in W$. To see that W exists, consider a uniformly random column S_i for $i \in [n]$. Then $\mathbf{E}[\|S_i\|^2] = r/n$ and so by Markov's inequality, at least a 1/2-fraction of columns S_i satisfy $\|S_i\|^2 \leq 2r/n$. We fix W to be an arbitrary subset of $n/2$ of these columns.

Suppose we sample t columns of S without replacement, indexed by $T \subset [n]$. Let $\mathcal{G}$ be the event that the set T of sampled columns belongs to the set W.

▶ **Lemma 3.** $\Pr[\mathcal{G}] \geq \sqrt{\delta}$.

Let $\alpha_G = \bar{\alpha}' \mid \mathcal{G}$ and $\beta_G = \bar{\beta}' \mid \mathcal{G}$. By the triangle inequality, $1 - 2\delta \leq D_{TV}(\bar{\alpha}', \bar{\beta}') \leq \Pr[\mathcal{G}] D_{TV}(\alpha_g, \beta_G) + 1 - \Pr[\mathcal{G}] \leq \frac{\sqrt{\delta}}{2} D_{TV}(\alpha_G, \beta_G) + 1 - \frac{\sqrt{\delta}}{2}$, which implies that $1 - 4\sqrt{\delta} \leq D_{TV}(\alpha_G, \beta_G)$. We can assume δ is less than a sufficiently small positive constant, and so it suffices to show for sketching dimension $r = o(n^{1-2/p}\varepsilon^{-2/p}(\log^{2/p} 1/\delta)\log n)$, that $D_{TV}(\alpha_G, \beta_G) \leq 1/2$. By Fact 1, it suffices to show $\chi^2(\alpha_G, \beta_G) \leq 1/4$.

Since S has orthonormal rows, $\bar{\alpha}'$ is distributed as $N(0, I_r)$. Note that, by definition of α, we in fact have $\bar{\alpha}' = \alpha_G$ since conditioning on $\mathcal{G}$ does not affect this distribution. On the other hand, β_G is a Gaussian location mixture, that is, it has the form $N(0, I_r) * p$, where p is the distribution of a random variable chosen by sampling a set T subject to event $\mathcal{G}$ occurring and outputting $\sum_{i \in T} \frac{C' \epsilon^{1/p} E_{n-t} S_i}{t^{1/p}}$. We can thus apply Fact 2 and it suffices to show for $r = o(n^{1-2/p}\varepsilon^{-2/p}(\log^{2/p} 1/\delta)\log n)$ that $\mathbf{E}[e^{\frac{(C')^2\epsilon^{2/p}E_{n-t}^2}{t^{2/p}}\langle \sum_{i\in T} S_i, \sum_{j \in U} S_j\rangle}] - 1 \leq \frac{1}{4}$, where the expectation is over independent samples T and U conditioned on $\mathcal{G}$. Note that under this conditioning T and U are uniformly random subsets of W.

To bound the χ^2-divergence, we define variables $x_{T,U}$, where $x_{T,U} = \frac{(C')^2\epsilon^{2/p}E_{n-t}^2}{t^{2/p}}\langle \sum_{i\in T} S_i, \sum_{j\in U} S_j\rangle$. Consider the following, where the expectation is over independent samples T and U conditioned on $\mathcal{G}$:

$$\begin{aligned}
\mathbf{E}\left[\exp\left\{\frac{(C')^2\epsilon^{2/p}E_{n-t}^2}{t^{2/p}}\langle \sum_{i\in T} S_i, \sum_{j\in U} S_j\rangle\right\}\right] &= \mathbf{E}\left[e^{x_{T,U}}\right] = \sum_{0 \leq j < \infty} \mathbf{E}\left[\frac{x_{T,U}^j}{j!}\right] \\
&= 1 + \sum_{j \geq 1} \frac{(C')^{2j}\varepsilon^{2j/p}E_{n-t}^{2j}}{t^{2j/p}j!}\mathbf{E}\left[\langle \sum_{i\in T} S_i, \sum_{j\in U} S_j\rangle^j\right] \\
&= 1 + \sum_{j \geq 1} \frac{O(1)^{2j}\varepsilon^{2j/p}n^{2j/p}}{t^{2j/p}j!}\mathbf{E}\left[\langle \sum_{i\in T} S_i, \sum_{j\in U} S_j\rangle^j\right].
\end{aligned}$$

The final equality uses that $E_{n-t} = \Theta(n^{1/p})$ and here $O(1)^{2j}$ denotes an absolute constant raised to the $2j$-th power. We can think of T as indexing a subset of rows of $S^T S$ and U indexing a subset of columns. Let M denote the resulting $t \times t$ submatrix of $S^T S$. Then $\langle \sum_{i\in T} S_i, \sum_{j\in U} S_j\rangle = \sum_{i,j\in[t]} M_{i,j} \leq \sum_{i,j\in[t]} |M_{i,j}| \stackrel{\text{def}}{=} P$, and we seek to understand the value of $\mathbf{E}[P^j]$ for integers $j \geq 1$.

The following lemma is the key to the argument; its proof is described in Section 1. The proof is based on defining y-patterns and looking at matchings in an associated bipartite multi-graph.

▶ **Lemma 4.** *For integers $j \geq 1$, $\mathbf{E}[P^j] \leq \left(\frac{t^2}{r^{1/2}}\right) \cdot \left(\frac{16r}{n}\right)^j$.*

Given the previous lemma, by δ-**Bound4**, we have $\frac{t^2}{r^{1/2}} = \frac{1}{n^{\Omega(1)}}$, and therefore Lemma 4 establishes that $\mathbf{E}[P^j] \leq \frac{1}{n^{\Omega(1)}} \cdot \left(\frac{16r}{n}\right)^j$. We thus have, $\mathbf{E}\left[\exp\left\{\frac{(C')^2\epsilon^{2/p}E_{n-t}^2}{t^{2/p}}\langle \sum_{i\in T} S_i, \sum_{j\in U} S_j\rangle\right\}\right] =$

$\mathbf{E}[e^{x_{T,U}}] = 1 + \frac{1}{n^{\Omega(1)}} \cdot \sum_{j \geq 1} \frac{O(1)^{2j} \epsilon^{2j/p} n^{2j/p}}{j! t^{2j/p}} \cdot \left(\frac{r}{n}\right)^j = 1 + \frac{1}{n^{\Omega(1)}} \cdot \sum_{j \geq 1} \frac{(c \log n)^j}{j!} \leq 1 + \frac{1}{n^{\Omega(1)}} \cdot e^{c(\log n)} \leq 1 + \frac{1}{4}$, since $c > 0$ is an arbitrarily small constant independent of the constant in the $n^{\Omega(1)}$. The proof is complete.

For $1 \leq p < 2$, we now show that the sketching dimension is $\Omega(\epsilon^{-2} \log(1/\delta))$, which as discussed in Section 1, matches known upper bounds up to a constant factor.

▶ **Theorem 5.** *The sketching dimension for (ϵ, δ)-approximating F_p for $1 \leq p < 2$ is $\Omega(\epsilon^{-2} \log(1/\delta))$.*

3 Algorithm

As outlined earlier, the algorithm uses two level-based structures, namely, GHSS, which is similar to the GHSS structure presented in [17], and the shelf structure. The shelf structure is needed only when $\delta = n^{-\omega(1)}$, otherwise, the GHSS structure suffices. The GHSS has $L + 1$ levels, corresponding to $l = 0, 1, \ldots, L$, and the shelf structure has J shelves numbered $0, 1, \ldots, J$. In particular, shelf 0 is identical to GHSS level 0.

3.1 Estimating F_p

GHSS *structure.* Corresponding to each GHSS level $l \in \{0, 1, \ldots, L-1\}$, a pair of CountSketch like structures named $\mathsf{HH}_l = \mathsf{HH}(C_l, s)$ (denoting that the number of buckets per table is $16C_l$ and number of independent repetitions is s) and $\mathsf{AvgEst}_l = \mathsf{AvgEst}(C_l, 2s)$ are kept. Here, $s = \Theta(\log n)$, recall $C = n^{1-2/p}(\epsilon^{-2} \log(1/\delta)/\log(n) + \epsilon^{-4/p} \log^{2/p}(1/\delta))$, $C = C_0 = \Theta(p^2 n^{1-2/p} \epsilon^{-4/p} \log^{2/p}(1/\delta))$ and $C_l = C_0 \alpha^l$, for $l = 0, 1, 2, \ldots, L-1$, where, $\alpha = 1 - (1 - 2/p)\nu$ and ν is a constant. The number of levels is $L = \lceil \log_{2\alpha}(n/C) \rceil$. The final level L of the GHSS structure uses an ℓ_2/ℓ_1 deterministic sparse-recovery algorithm [9, 13]. We will show that the number of items that are subsampled into level L is $O(C_L)$ with probability $1 - O(\delta)$ and therefore from [9, 13], by using $O(C_L \log(n/C_L))$ measurements, these item frequencies are recovered deterministically. Following [17], the GHSS structure subsamples the stream hierarchically using independent random hash functions $g_1, \ldots, g_L : [n] \to \{0, 1\}$. All items are mapped to level 0; an item is mapped to each of levels 1 through l iff $g_1(i) = \ldots = g_l(i) = 1$, where, the g_l's are $O(\log(1/\delta) + \log n)$-wise independent.

HH and AvgEst structures. The $\mathsf{HH}(C_l, s)$ is a CountSketch structure [11]. The $\mathsf{AvgEst}(C_l, 2s)$ structure is similar, except that instead of Rademacher sketches, it uses random qth roots of unity sketches, where, $q = O(\log(1/\delta) + \log n)$. At level l and for table indexed $r \in [2s]$, the corresponding hash function is $h_{lr} : [n] \to [16C_l]$, and the sketch for bucket index b is given by $T_{lr}[b] = \sum_{h_{lr}(i)=b} x_i \omega_{lr}(i)$, where, $\{\omega_{lr}(i)\}_{i \in [n]}$ is a random family of qth roots of unity that is $O(\log(1/\delta) + \log n)$-wise independent. The hash functions across the tables and distinct levels, and the seeds of the family of the random roots of unity, are independent.

Shelf structure. The shelves, indexed from $j = 0, \ldots, J$, each also consist of an analogous pair of structures, namely, $\mathsf{HH}(H_j, w_j)$ and $\mathsf{AvgEst}(H_j, 2w_j)$, where, $O(H_j)$ is the number of buckets per hash table in these structures, and there are $O(w_j)$ independent repetitions per structure. The AvgEst structures of the shelves also use sketches using qth roots of unity, instead of Rademacher sketches. In particular, $H_0 = C_0$ and $w_0 = s$, ensuring that shelf 0 coincides with level 0 of GHSS. Further, $H_J = \Theta(n^{1-2/p} \epsilon^{-2})$ and $w_J = O(\log(1/\delta))$. There are two cases, namely, (1) $H_J = \Omega(H_0)$, or, (2) $H_J = o(H_0)$. In the first case, $J = 1$ and there are only two shelves, considerably simplifying the analysis. The other case $H_J = o(H_0)$ is more interesting. Here, we let $H_j = H_0 b^j$, for a parameter $b < 1$ and $b = \Omega(1)$. The table

widths increase geometrically as $w_j = w_0 a^j$, for a parameter $a > 1$. The total measurements used by the shelf structure is $\sum_{j=0}^{J} H_j w_j = H_0 w_0 \sum_{j=0}^{J} (ab)^j = O(\max(H_0 w_0, H_J w_J))$, provided, $|1 - ab| = \Omega(1)$, or, $|\ln(ab)| = \Omega(1)$. The entire stream $\mathcal{S}$ is provided as input to each of the shelves $j = 0, 1, \ldots, J$.

Frequency groups, thresholds, estimates and samples. Let $B = \Theta(C)$ and $\bar{\epsilon} = (B/C)^{1/2} = \Theta(1/p)$. Let $\hat{F}_2$ be an estimate for $F_2 = \|x\|_2^2$ satisfying $F_2 \leq \hat{F}_2 \leq (1 + O(1/p))F_2$ with probability $1 - O(\delta)$. Define frequency thresholds for GHSS levels as follows: $T_0 = (\hat{F}_2/B)^{1/2}$, $T_l = (2\alpha)^{-l/2} T_0$ and $Q_l = T_l(1 - \bar{\epsilon})$, for $l \in [L - 1]$. Let $Q_L, T_L = 0^+$ (i.e., $a \geq T_L$ iff $a > 0$). For shelf $j = 0, \ldots, J$, let $E_j = \bar{\epsilon}^2 H_j$. For shelf j, define the frequency threshold $U_j = (\hat{F}_2/E_j)^{1/2}$ and let $U_{J+1} = \infty$. For GHSS level indices $l = 0, \ldots, L-1$, let $\hat{x}_{il}$ denote the estimate for x_i obtained using HH_l, and (overloading notation), for shelf indices, $j = 0, \ldots, J$, let $\hat{x}_{ij}$ denote the estimate for x_i obtained from the HH structure of shelf j. $\hat{x}_{iL}$ denotes the estimate returned from the ℓ_2/ℓ_1 sparse recovery structure at level L.

Discovering Items. We say that i is *discovered* at shelf $j \in [J]$, provided, $(1 - \bar{\epsilon})U_j \leq |\hat{x}_{ij}| \leq (1 + \bar{\epsilon})U_{j+1}$ and $j \in [J]$ is the *highest* numbered shelf with this property. We say that i is discovered at GHSS level $l \in \{0, \ldots, L\}$, if i is not discovered at any shelf indexed $j \in [J]$, and l is the *smallest* level such that $T_l(1 - \bar{\epsilon}) < \hat{x}_{il} \leq T_{l-1}(1 + \bar{\epsilon})$. If i is discovered at shelf j, then, i is included in the shelf sample $\bar{S}_j$. If i is discovered at level $l \in [0, 1, \ldots, L]$ and $|\hat{x}_{il}| \geq T_l$, then, i is included in the level sample $\bar{G}_l$. If i is discovered at level l and $T_l(1 - \bar{\epsilon}) < |\hat{x}_{il}| < T_l$ then, i is placed in $\bar{G}_{l+1}$ iff the random toss of an unbiased coin K_i lands heads; and upon tails, it is not placed in any sample group. The GHSS level sampling scheme is similar to [17].

The averaged estimator and NOCOLLISION. For each item i included in a group sample $\bar{G}_l$ or shelf sample $\bar{S}_j$, an estimate X_i for $|x_i|$ is obtained using the corresponding AvgEst structure of that level or shelf, provided the event NOCOLLISION(i) succeeds. If i is sampled into $\bar{G}_l$, then NOCOLLISION(i) holds if there is a set $R_l(i) \subset [2s]$ of table indices of the AvgEst_l structure such that for each $r \in R_l(i)$, i does not collide under the hash function h_{lr} with any of the items that are the top-C_l absolute estimated frequencies using HH_l. An analogous definition holds if i is included in the jth shelf sample. Assuming NOCOLLISION(i) holds, the estimate X_i is defined as the average of the estimates obtained from the tables whose indices are in the set $R_l(i)$ (resp. $R_j(i)$ if i was discovered in shelf j), that is, $X_i = (1/|R(i)|) \sum_{r \in R(i)} T_r[h_r(i)] \cdot \overline{\omega_r(i)} \cdot \mathrm{sgn}(\hat{x_i})$. Further, we check whether $(1 - \bar{\epsilon})T_l \leq X_i \leq (1 + \bar{\epsilon})T_{l-1}$ (resp. $(1 - \bar{\epsilon})U_j \leq X_i \leq (1 + \bar{\epsilon})U_{j+1}$, if i is in shelf j sample), otherwise, i is dropped from the sample.

Estimating F_p. The estimate for the pth frequency moment, $\hat{F}_p$, is the sum of the contribution from the shelf samples $\bar{S}_j, j \in [J]$, and the contribution from the sample groups $\bar{G}_l$, $l = 0, \ldots, L$. For an item $i \in \bar{G}_l$, let $l_d(i)$ be the level at which an item i is discovered. Let $\hat{F}_p^{\text{SHELF}} = \sum_{j=1}^{J} \sum \left\{ X_i^p \mid i \in \bar{S}_j, (1 - \bar{\epsilon})U_j \leq |X_j| \leq (1 + \bar{\epsilon})U_{j+1} \right\}$, and $\hat{F}_p^{\text{GHSS}} = \sum_{l=0}^{L} 2^L \sum \left\{ X_i^p \mid i \in \bar{G}_l, l_d(i) < L, (1 - \bar{\epsilon})T_{l_d} \leq X_i < (1 + \bar{\epsilon})T_{l_d - 1} \right\} + 2^L \sum_{l_d(i) = L} |\hat{x}_i|^p$. The final estimate is $\hat{F}_p = \hat{F}_p^{\text{SHELF}} + \hat{F}_p^{\text{GHSS}}$.

3.2 Analysis

Notation. Let $F_2^{\text{res}}(k)$ be the sum of the squares of all coordinates except the top-k absolute coordinates. For a GHSS level $l \in [L]$, $F_2^{\text{res}}(l, k)$ is the random k-residual second moment of the frequency vector in the sampled substream $\mathcal{S}_l$. Let (1) $\text{GOODF}_2 \equiv F_2 \leq \hat{F}_2 \leq (1 + 0.001/(2p))F_2$, (2) $\text{SMALLRES}_l \equiv F_2^{\text{res}}(2C_l, l) \leq 1.5 F_2^{\text{res}} \left(\lceil (2\alpha)^l C \rceil \right) / 2^{l-1}$, $l = 0, 1, \ldots, L$, (3) $\text{SMALLRES} \equiv \forall l \in \{0, 1, \ldots, L\}$ SMALLRES_l, (4) $\text{GOODLASTLEVEL} \equiv (\hat{f}_{iL} = f_i)$ and $\forall i \notin \mathcal{S}_L, (\hat{f}_{iL} = 0)$. We condition the analysis on the "good event" $\mathcal{G} \equiv \text{GOODF}_2 \wedge \text{SMALLRES} \wedge \text{GOODLASTLEVEL}$, that we show holds with probability $1 - \min(O(\delta), n^{-\Omega(1)})$.

▶ **Lemma 6.** *$\mathcal{G}$ holds with probability* $1 - \min(O(\delta), n^{-\Omega(1)})$.

The range of item frequencies is subdivided into frequency groups , so that each item belongs to exactly one shelf frequency group or to exactly one GHSS frequency group. The frequency group corresponding to the shelf j is $[U_j, U_{j+1})$, for $j = 1, \ldots, J$, where, $U_{J+1} = \infty$ and $U_0 = T_0$. The frequency group corresponding to level l of GHSS is $[T_l, T_{l-1})$, where, $T_L = 0$ and $T_{-1} = U_1$. Let S_j (resp. G_l) denote the set of items whose frequency belongs to the frequency group corresponding to shelf j (resp. group l). A few other events are used in the analysis. If $i \in G_l$, then, $\Pr[\text{NOCOLLISION}(i)] \geq 1 - \exp\{-\Theta(\log n)\}$ as shown in [17] (Lemma 30). If $i \in S_j$, $\Pr[\text{NOCOLLISION}(i)] \geq 1 - \exp\{-\Theta(w_j)\}$. We condition on the following events.

$$\text{GOODEST}(i) \equiv \forall l \in [0, \ldots, L], i \in \mathcal{S}_l \Rightarrow |\hat{x}_{il} - x_i| \leq \left(F_2^{\text{res}}\left(2C_l, l\right)/C_l\right)^{1/2} \quad (5)$$

$$\text{ACCUEST}(i) \equiv \forall l \in [0, \ldots, L], i \in \mathcal{S}_l \Rightarrow |\hat{x}_{il} - x_i| \leq \left(F_2^{\text{res}}\left((2\alpha)^l C\right)/(2(2\alpha)^l C)\right)^{1/2} . \quad (6)$$

As shown in [17], (a) GOODEST(i) and ACCUEST(i) each hold with probability $1 - n^{-\Omega(1)}$, and, (b) GOODEST$(i) \wedge$ SMALLRES imply the event ACCUEST(i). For an item i that is discovered at some shelf j, GOODEST(i) is the same as ACCUEST(i) and is defined as $|\hat{x}_{ij} - x_i| \leq \left(F_2^{\text{res}}\left(U_j\right)/U_j\right)^{1/2}$ and holds with probability $1 - \exp\{-\Theta(w_j)\}$.
Lemma 7 extends the approximate 2-wise independence property of the sampling scheme of [17] to an approximate d-wise independence property.

▶ **Lemma 7.** *Let* $I = \{i_1, \ldots, i_d\} \subset [n]$ *and* $1 \leq h \leq d$. *Let* ACCUEST$(\{i_1, \ldots, i_h\}) \equiv \bigwedge_{k=1}^{h}$ ACCUEST(i_k). *Then, assuming d-wise independence of the hash functions,* $\sum_{\substack{l_j=0,1,\ldots,L,\\ \forall j=1,2,\ldots,h}} 2^{l_1+l_2+\ldots+l_h} Pr\{\bigwedge_{j=1}^{h} i_j \in \bar{G}_{l_j} | \bigwedge_{j=h+1}^{d} i_j \in \mathcal{S}_{l_j}, \mathcal{G}, \text{ACCUEST}(\{i_1, \ldots, i_h\})\} \in \prod_{j=1}^{h}\left(1 \pm 2^{\text{level}(i_j)+1} n^{-c}\right)$.

Lemma 8 bounds $|X_i - \mathbf{E}[X_i]|$ using the $2d$th moment method.

▶ **Lemma 8.** *Suppose* $d \leq O(\log n)$ *and even and let* $s \geq 300\log(n)$. *Then we have that* $Pr\{|X_i - |x_i|| > \left(\frac{dF_2^{res}(2C)}{(s/9)C}\right)^{1/2} \mid \text{NOCOLLISION}, \text{GOODEST}\} < 2^{-2d+1}$.

▶ **Lemma 9** ([22]). $\left|\mathbf{E}[X_i^p] - |x_i|^p \mid \mathcal{G}, \text{GOODEST}, \text{NOCOLLISION}\right| \leq |x_i|^p n^{-\Omega(1)}$.

For $i \in [n]$, let x_{li} be an indicator variable that is 1 iff $i \in \mathcal{S}_l$. Let X_i denote $|\hat{x}_i|$ when $l_d(i) = L$ and otherwise, let its meaning be unchanged. Let z_{il} be an indicator variable that is 1 if $i \in \bar{G}_l$ and 0 otherwise. Define $\hat{F}_p = \sum_{i\in[n]} Y_i$. where, $Y_i = \sum_{l'=0}^{L} 2^{l'} z_{il'} X_i^p$. Let $\mathcal{H} = \mathcal{G} \cap$ NOCOLLISION $\cap$ GOODEST and $G' = \text{lmargin}(G_0) \cup_{l=1}^{L} G_l$.

▶ **Lemma 10.** *Let* $B \geq O(n^{1-2/p}\epsilon^{-4/p}\log^{2/p}(1/\delta)))$. *For integral* $0 \leq d_1, d_2 \leq \lceil\log(1/\delta)\rceil$, *we have,* $\mathbf{E}\left[\left(\sum_{i\in G'}(Y_i - \mathbf{E}[Y_i \mid \mathcal{H}])\right)^{d_1}\left(\sum_{i\in G'}(\overline{Y_i} - \mathbf{E}[\overline{Y_i} \mid \mathcal{H}])\right)^{d_2} | \mathcal{H}\right] \leq \left(\frac{\epsilon F_p}{20}\right)^{d_1+d_2}$.

3.3 Analysis for the case $\delta \geq n^{-O(1)}$

For the case $\delta = n^{-O(1)}$, the shelf structure is not needed. Redefine the group G_0 to correspond to the frequency range $[T_0, \infty]$. The lemmas in this section assume that the family $\{\omega_{lr}(i)\}_{i\in[n]}$ are $O(\log(1/\delta) + \log(n))$-wise independent, and independent across l, r and all hash functions are also $O(\log(1/\delta) + \log(n))$-wise independent.

▶ **Lemma 11.** *Let* $1 \le e, g \le \lceil \log(1/\delta) \rceil, l \in mid(G_0)$ *and* $|x_l| \ge \left(\frac{F_2^{res}(C)}{C}\right)^{1/2}$. *Then,* $\mathbf{E}\left[(Y_l - \mathbf{E}[Y_l \mid \mathcal{H}])^e \left(\overline{Y_l} - \mathbf{E}\left[\overline{Y_l} \mid H\right]\right) \mid \mathcal{H}\right]$ *is real and is at most* $\left(\frac{a|x_l|^{2p-2}F_2^{res}(C)}{\rho C}\right)^{(e+g)/2}$ *for some constant* a. *Further,* $\left|\mathbf{E}\left[(Y_l - \mathbf{E}[Y_l \mid \mathcal{H}])^e\right] \mid \mathcal{H}\right| \le |x_l|^{pe} n^{-\Omega(e)}$.

The calculation of the dth central moment for the contribution to $\hat{F}_p$ from the items in $\text{mid}(G_0)$ requires an upper bound on the following combinatorial sums. $Q(S_1, S_2) = \sum_{q=1}^{\min(S_1,S_2)} \sum_{\substack{e_1+\ldots+e_q=S_1 \\ e'_j s \ge 1}} \sum_{\substack{g_1+\ldots+g_q=S_2 \\ g'_j s \ge 1}} \binom{S_1}{e_1,\ldots,e_q}\binom{S_2}{g_1,\ldots,g_q}$, and
$R(S) = \sum_{q=1}^{\lfloor S/2 \rfloor} \sum_{h_1+\ldots+h_q=S, h'_j s \ge 2} \binom{S}{h_1,\ldots,h_q} \sum_{\{i_1,\ldots,i_q\}} \prod_{r\in[q]} |x_{i_r}|^{(p-1)h_r} \prod_{r\in[q]} h_r^{h_r/2}$.

▶ **Lemma 12.** $Q(S_1, S_2) \le R(S_1 + S_2) \le (16e(S_1 + S_2)F_{2p-2})^{(S_1+S_2)/2}$.

▶ **Lemma 13.** *Let* $C \ge O(n^{1-2/p}/\log(n))\epsilon^{-2}\log(1/\delta))$. *Then, for* $0 \le d_1, d_2 \le \log(1/\delta)$, $\mathbf{E}\left[\left(\sum_{i\in mid(G_0)}(Y_i - \mathbf{E}[Y_i \mid \mathcal{H}])\right)^{d_1}\left(\sum_{i\in mid(G_0)}(\overline{Y_i} - \mathbf{E}\left[\overline{Y_i} \mid H\right])\right)^{d_2} \mid \mathcal{H}\right] \le \left(\frac{\epsilon F_p}{10}\right)^{d_1+d_2}$.

▶ **Lemma 14.** *Let* $C \ge Kn^{1-2/p}\epsilon^{-2}\log(1/\delta)/\log(n) + Ln^{1-2/p}\epsilon^{-4/p}\log^{2/p}(1/\delta)$, *where,* K, L *are constants. Then, for* $d = \lceil \log(1/\delta) \rceil$, $\mathbf{E}\left[\left(\sum_{i\in S}(Y_i - \mathbf{E}[Y_i \mid \mathcal{H}])\right)^d\left(\sum_{i\in S}(\overline{Y_i} - \mathbf{E}\left[\overline{Y_i} \mid \mathcal{H}\right])\right)^d \mid \mathcal{H}\right] \le \left(\frac{\epsilon F_p}{5}\right)^{2d}$. *It follows that* $Pr\left[\left|\hat{F}_p - F_p\right| \ge (\epsilon/2)F_p\right] \le \delta$.

Since, $\mathcal{H}$ holds with probability $1 - 1/n^{-c}$, for any constant c by choosing $s = \Theta(\log n)$ appropriately, we have the following theorem.

▶ **Theorem 15.** *For each* $0 < \epsilon < 1$ *and* $7/8 \ge \delta \ge n^{-c}$, *for any constant* c, *there is a sketching algorithm that* (ϵ, δ)*-approximates* F_p *with sketching dimension* $O\left(n^{1-2/p}\left(\epsilon^{-2}\log(1/\delta) + \epsilon^{-4/p}\log^{2/p}(1/\delta)\log n\right)\right)$ *and update time (per stream update)* $O((\log n)\log(1/\delta)))$.

3.4 Analysis for the case $\delta = n^{-\omega(1)}$

We now extend the analysis for failure probability δ smaller than $n^{-\Theta(1)}$ and up to $\delta = 2^{-n^{\Omega(1)}}$. For the GHSS structure, NOCOLLISION and GOODEST may hold only with probability $1 - n^{-\Theta(1)}$. We first show that the number of items that fail to satisfy NOCOLLISION or GOODEST is at most $O(\log(1/\delta)/\log n)$ with probability $1 - O(\delta)$. The following lemmas assume the parameter sizes for B, C, C_l, H_J and H_j as described earlier.

▶ **Lemma 16.** *With probability* $1 - O(\delta)$, *the number of elements for which* GOODEST *or* NOCOLLISION *fails is at most* $O(\log(1/\delta))/(\log n)$.

Thus, it is possible that legitimate items are not discovered, or are dropped due to collisions, or mistakenly classified and their contribution added to samples. Let $\text{Error}^{\text{GHSS}}$ denote the total contribution of such items to $\hat{F}_p^{\text{GHSS}}$ and let $\text{Error}^{\text{SHELF}}$ denote the error arising in the estimate of $\hat{F}_p^{\text{SHELF}}$ due to analogous errors. As described earlier, we mainly emphasize the more interesting and complicated case when $H_J = o(H_0)$ (otherwise, $J = 1$).

▶ **Lemma 17.** $Error^{\text{GHSS}} \le O(\epsilon^2 F_p/\log n)$ *and* $Error^{\text{SHELF}} \le O(\max(\epsilon^2 F_p/(\log n), O(\epsilon^p F_p)))$, *each with probability* $1 - \delta/n^{\Omega(1)}$.

We first prove a refinement of Lemma 11.

▶ **Lemma 18.** *[Refinement of Lemma 11.] Let* $1 \le e, g \le \lceil \log(1/\delta) \rceil$, $l \in S_j$ *and* $\log(1/\delta) = \omega(\log n)$. *Assume that* ACCUEST$(l)$ *holds and* $H_j \ge \Omega(p^2 E_J)$ *and* $|x_l| \ge (F_2/E_j)^{1/2}$. *Then,* $\mathbf{E}\left[\left(\left(1 + \frac{Z_l}{|x_l|}\right)^p - 1\right)^e \left(\left(1 + \frac{\overline{Z_l}}{|x_l|}\right)^p - 1\right)^g \mid \mathcal{H}\right]$ *is real and bounded above by* $c^h |x_l|^{-h}\left(\frac{F_2}{H_j}\right)^{h/2}\left(\min\left(\frac{h}{w_j}, 1\right)\right)^{h/2}$, *where,* $h = e + g$ *and* c *is an absolute constant.*

Lemma 19 considers the $2d$th central moment of the contribution to $\hat{F}_p^{\text{SHELF}}$ from all but the outermost shelf, and from the set of outermost shelf items denoted S_J, separately. Let $S' = S_1 \cup \ldots \cup S_{J-1}$.

▶ **Lemma 19.** *Let $0 \leq d_1, d_2 \leq \lceil \log(1/\delta) \rceil$ and integral and c_1, c_2 be constants. Then,* $\mathbf{E}\left[\left(\sum_{i\in S'}(Y_i - \mathbf{E}\left[Y_i \mid \mathcal{H}\right])\right)^{d_1}\left(\sum_{i\in S'}(\overline{Y_i} - \overline{\mathbf{E}\left[Y_i \mid \mathcal{H}\right]})\right)^{d_2} \mid \mathcal{H}\right] \leq (c_1\epsilon F_p)^{d_1+d_2}$. $\mathbf{E}\left[\left(\sum_{i\in S_J}(Y_i - \mathbf{E}\left[Y_i \mid \mathcal{H}\right])\right)^{d_1}\left(\sum_{i\in S_J}(\overline{Y_i} - \overline{\mathbf{E}\left[Y_i \mid \mathcal{H}\right]})\right)^{d_2} \mid \mathcal{H}\right] \leq (c_2\epsilon F_p)^{d_1+d_2}$.

Combining Lemmas 10, 13 and 19 with Lemma 17, we obtain the following.

▶ **Lemma 20.** *$\exists$ constant c s.t. for $1 \leq d \leq \lceil \log(1/\delta) \rceil$,* $\mathbf{E}\left[\left(\sum_{i\in[n]}(Y_i - \mathbf{E}\left[Y_i \mid \mathcal{H}\right])\right)^{d}\left(\sum_{i\in[n]}(\overline{Y_i} - \mathbf{E}\left[\overline{Y_i} \mid \mathcal{H}\right])\right)^{d} \mid \mathcal{H}\right] \leq (c\epsilon F_p)^{2d}$. *Hence, $Pr\left[\left|\hat{F}_p - F_p\right| \leq \epsilon F_p)\right] < \delta$.*

▶ **Theorem 21.** *For each $0 < \epsilon < 1$ and $7/8 \geq \delta \geq 2^{-n^{\Omega(1)}}$, there is a sketching algorithm that (ϵ, δ)-approximates F_p with sketching dimension $O\big(n^{1-2/p}\big(\epsilon^{-2}\log(1/\delta) + \epsilon^{-4/p}\log^{2/p}(1/\delta)\log n\big)\big)$ and update time (per stream update) $O((\log n)\log(1/\delta))$.*

References

1. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *JCSS*, 58(1):137–147, 1999.
2. Alexandr Andoni. High frequency moment via max stability. Available at http://web.mit.edu/andoni/www/papers/fkStable.pdf.
3. Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms from precision sampling. *CoRR*, abs/1011.1263, 2010. URL: `http://arxiv.org/abs/1011.1263`.
4. Alexandr Andoni, Huy L. Nguyen, Yury Polyanskiy, and Yihong Wu. "Tight Lower Bound for Linear Sketches of Moments". In *Proceedings of International Conference on Automata, Languages and Programming, (ICALP)*, jul 2013. Version published as arXiv:1306.6295, June 2013.
5. Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1345–1364, 2016.
6. Z. Bar-Yossef, T.S. Jayram, R. Kumar, and D. Sivakumar. "An information statistics approach to data stream and communication complexity". In *Proceedings of ACM Symposium on Theory of Computing STOC*, pages 209–218, 2002.
7. Lakshminath Bhuvanagiri, Sumit Ganguly, Deepanjan Kesh, and Chandan Saha. Simpler algorithm for estimating frequency moments of data streams. In *SODA*, pages 708–713, 2006. `doi:10.1145/1109557.1109634`.
8. Vladimir Braverman and Rafail Ostrovsky. Recursive sketching for frequency moments. *CoRR*, abs/1011.2571, 2010.
9. Emmanuel Candès, Justin Romberg, and Terence Tao. "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information". *IEEE Trans. Inf. Theory*, 52(2):489–509, feb 2006.
10. Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *CCC*, pages 107–117, 2003.
11. Moses Charikar, Kevin Chen, and Martin Farach-Colton. "Finding frequent items in data streams". *Theoretical Computer Science*, 312(1):3–15, 2004. Preliminary version appeared in Proceedings of ICALP 2002, pages 693-703.

12 Don Coppersmith and Ravi Kumar. An improved data stream algorithm for frequency moments. In *SODA*, 2004.

13 David L. Donoho. "Compressed Sensing". *IEEE Trans. Inf. Theory*, 52(4):1289–1306, apr 2006.

14 Sumit Ganguly. Estimating frequency moments of data streams using random linear combinations. In *RANDOM*, 2004.

15 Sumit Ganguly. A hybrid algorithm for estimating frequency moments of data streams, 2004. Manuscript.

16 Sumit Ganguly. Polynomial estimators for high frequency moments. *CoRR*, abs/1104.4552, 2011.

17 Sumit Ganguly. "Taylor Polynomial Estimator for Estimating Frequency Moments". In *Proceedings of International Conference on Automata, Languages and Programming, (ICALP)*, 2015. Full version in arXiv:1506.01442.

18 Moritz Hardt and David P. Woodruff. How robust are linear sketches to adaptive inputs? In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 121–130, 2013.

19 P. Indyk and D. Woodruff. Optimal approximations of the frequency moments of data streams. In *STOC*. ACM, 2005.

20 Piotr Indyk. "Stable Distributions, Pseudo Random Generators, Embeddings and Data Stream Computation". In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 189–197, 2000.

21 Y. I. Ingster and L.A. Suslina. *"Non-parametric goodness-of-fit testing under Gaussian models"*, volume 169 of *Lecture Notes in Statistics*. Springer-Verlag, 2003.

22 Daniel Kane, Jelani Nelson, Ely Porat, and David Woodruff. "Fast Moment Estimation in Data Streams in Optimal Space". In *Proceedings of 2011 ACM Symposium on Theory of Computing, version arXiv:1007.4191v1 July*, 2010.

23 Daniel Kane, Jelani Nelson, Ely Porat, and David Woodruff. "Fast Moment Estimation in Data Streams in Optimal Space". In *Proceedings of 2011 ACM Symposium on Theory of Computing, version arXiv:1007.4191v1 July*, 2011.

24 Daniel M. Kane, Raghu Meka, and Jelani Nelson. Almost optimal explicit johnson-lindenstrauss families. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011, Princeton, NJ, USA, August 17-19, 2011. Proceedings*, pages 628–639, 2011.

25 Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. Fast moment estimation in data streams in optimal space. In *STOC*, pages 745–754, 2011.

26 Daniel M. Kane, Jelani Nelson, and David Woodruff. "An Optimal Algorithm for the Distinct Elements Problem". In *Proceedings of ACM International Symposium on Principles of Database Systems (PODS)*, pages 41–52, 2010.

27 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. "On the Exact Space Complexity of Sketching and Streaming Small Norms". In *Proceedings of ACM Symposium on Discrete Algorithms (SODA)*, 2010.

28 Christian Konrad. Maximum matching in turnstile streams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 840–852, 2015.

29 Yi Li, Huy L. Nguyen, and David P. Woodruff. On sketching matrix norms and the top singular vector. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1562–1581, 2014.

30 Yi Li and David Woodruff. "A Tight Lower Bound for High Frequency Moment Estimation with Small Error". In *Proceedings of International Workshop on Randomization and Computation (RANDOM)*, 2013.

31 Morteza Monemizadeh and David P. Woodruff. 1-pass relative-error l_p-sampling with applications. In *SODA*, 2010.

32 Eric Price and David P. Woodruff. Applications of the shannon-hartley theorem to data streams and sparse recovery. In *ISIT*, 2012.

33 Alexandre B. Tsybakov. *"Introduction to Nonparametric Estimation"*. Springer, 1 edition, 2008.

34 Omri Weinstein and David P. Woodruff. The simultaneous communication of disjointness with applications to data streams. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 1082–1093, 2015.

35 David P. Woodruff. *"Sketching as a Tool for Numerical Linear Algebra"*. Foundations and Trends in Theoretical Computer Science 10:1-2, Now Publications, 2014.

Quasi-PTAS for Scheduling with Precedences using LP Hierarchies

Shashwat Garg[1]
Eindhoven University of Technology, Netherlands
s.garg@tue.nl

Abstract

A central problem in scheduling is to schedule n unit size jobs with *precedence constraints* on m identical machines so as to minimize the makespan. For $m = 3$, it is not even known if the problem is NP-hard and this is one of the last open problems from the book of Garey and Johnson.

We show that for fixed m and ϵ, polylog(n) rounds of Sherali-Adams hierarchy applied to a natural LP of the problem provides a $(1+\epsilon)$-approximation algorithm running in quasi-polynomial time. This improves over the recent result of Levey and Rothvoss, who used $r = (\log n)^{O(\log \log n)}$ rounds of Sherali-Adams in order to get a $(1 + \epsilon)$-approximation algorithm with a running time of $n^{O(r)}$.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Approximation algorithms, hierarchies, scheduling, rounding techniques

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.59

Related Version A full version of the paper is available at `https://arxiv.org/pdf/1708.04369.pdf`.

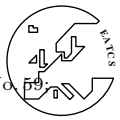
Acknowledgements We would like to thank Seeun William Umboh, Martin Böhm and Nikhil Bansal for helpful discussions throughout this work.

1 Introduction

A central problem in scheduling is the following: suppose we are given n unit jobs which have to be processed non-preemptively on m identical machines. There is also a precedence order among the jobs: if $i \prec j$, then job i has to be completed before j can begin. The goal is to find a schedule of the jobs with the minimum makespan, which is defined as the time by which all the jobs have finished.

This problem admits an easy $(2 - \frac{1}{m})$ approximation algorithm which was given by Graham [5] in the 60's and is one of the landmark results in scheduling. This algorithm is known as the *list-scheduling* algorithm and works as follows: at every time $t = 1, 2, \ldots$, if there is an empty slot on any of the m machines, schedule any *available* job there, where a job is available if it is not yet scheduled and all the jobs which must precede it have already been scheduled. This simple greedy algorithm is essentially the best algorithm for the problem and for almost half a century it was an open problem whether one can get a better approximation algorithm. In fact, this was one of the ten open problems in Schuurman and Woeginger's influential list of open problems in scheduling [11]. It was known since the 70's that it is NP-hard to get an approximation factor better than 4/3 [8]. Slight improvements were given

[1] Supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 022.005.025.



by Lam and Sethi [6] who gave a $2 - \frac{2}{m}$ approximation algorithm, and Gengal and Ranade [3] who gave a $2 - \frac{7}{3m+1}$ approximation algorithm for $m \geq 4$. Finally in 2010, Svensson [13] showed that assuming a variant of Unique Games conjecture due to Bansal and Khot [1], for any constant $\epsilon > 0$ there is no $(2 - \epsilon)$ approximation algorithm for the problem.

However, this still leaves open the problem for the important case when m is a constant. In fact in practice, usually the number of jobs are very large but there are only a few machines. Surprisingly, for $m = 3$, it is not even known if the problem is NP-hard. This is one of the four problems from the book of Garey and Johnson [4] whose computational complexity is still unresolved.

In order to get a better algorithm for the case when m is a constant, a natural strategy is to write a linear program (LP), and for this problem, one such LP is the time-indexed LP (1), in which we first make a guess T of the makespan and then solve the LP. The value of the LP is the smallest T for which the LP is feasible, and the worst case ratio of the optimal makespan and the value of the LP is known as the integrality gap of the LP. It is well known that LP (1) has an integrality gap of at least $2 - \frac{2}{m+1}$ (see e.g. [9]), which suggests that one needs to look at stronger convex relaxations in order to get a better algorithm. Such a stronger convex relaxation can be obtained by applying a few rounds of a hierarchy to the LP, and in this paper, we will use the Sherali-Adams hierarchy [12]. It is known that just one round of Sherali-Adams hierarchy reduces the integrality gap to 1 for $m = 2$ and thus, the problem can be solved exactly in this case (credited to Svensson in [10]). Claire Mathieu in a workshop in Dagstuhl 2010 asked if one can get a $(1 + \epsilon)$-approximation algorithm using $f(\epsilon, m)$ rounds of Sherali-Adams hierarchy for some function f independent of n, which would imply a PTAS for the problem when m is a constant. This is also Open Problem 1 in Bansal's recent list of open problems in scheduling presented at MAPSP 2017.

To get some intuition behind why hierarchies should help in this problem, let us first look at the analysis for Graham's list-scheduling algorithm. At the end of this algorithm, the number of time slots which are busy, that is where all the m machines have some job scheduled on them, is a lower bound on the optimum. Also, the number of non-busy time slots is a lower bound on the optimum. This is because there must be a *chain* of jobs $j_1 \prec j_2 \prec \cdots \prec j_k$ such that one job from this chain is scheduled at each non-busy time, and the length of any chain in the instance is clearly a lower bound on the optimum. This implies that the makespan given by the algorithm, which is the sum of the number of busy and non-busy time slots, is a 2-approximation of the optimum makespan, and a slightly more careful argument gives the guarantee of $(2 - 1/m)$. Now the key idea is that if the instance given to us has a maximum chain length of at most ϵ times the optimal makespan, then Graham's list-scheduling algorithm already gives a $(1 + \epsilon)$-approximation, and hierarchies provide, via conditionings, a good way to "effectively" reduce the length of the chains in any given instance.

Though the question of whether one can get a $(1 + \epsilon)$-approximation algorithm using $f(\epsilon, m)$ rounds of Sherali-Adams hierarchy is still unresolved, a major breakthrough was made recently by Levey and Rothvoss [9], who gave a $(1 + \epsilon)$-approximation algorithm using $r = (\log n)^{O(m^2 \log\log n/\epsilon^2)}$ rounds of Sherali-Adams. This gives an algorithm with a running time of $n^{O(r)}$, which is faster than exponential time but worse than quasi-polynomial time.

1.1 Our Result

In this paper, we improve over the result of Levey and Rothvoss [9] by giving a $(1 + \epsilon)$-approximation algorithm which runs in quasi-polynomial time. Formally, we show the following:

▶ **Theorem 1.** *The natural LP* (1) *for the problem augmented with r rounds of Sherali-Adams hierarchy has an integrality gap of at most $(1+\epsilon)$, where $r = O_{m,\epsilon}(\log^{O(m^2/\epsilon^2)} n)$. Moreover, there is a $(1+\epsilon)$-approximation algorithm for this problem running in time $n^{O(r)}$.*

Throughout the paper, we use the notation $O_{m,\epsilon}(.)$ to hide factors depending only on m and ϵ. The natural LP for the problem is the following:

$$\begin{aligned}
\sum_{t=1}^{T} y_{jt} &= 1 && \forall j \in [n] \\
\sum_{j} y_{jt} &\leq m && \forall t \leq T \\
\sum_{t' \leq t} y_{jt'} &\geq \sum_{t' \leq t+1} y_{it'} && \forall t \leq T,\ \forall j \prec i \\
y_{jt} &\geq 0 && \forall t \leq T,\ \forall j \in [n]
\end{aligned} \tag{1}$$

Here T is our guess on the optimum makespan. In an integral solution, $y_{jt} = 1$ if job j is scheduled at time t, and 0 otherwise. The first constraint ensures that each job is scheduled at exactly one time and the second constraint ensures that no more than m jobs are scheduled at any time. The third constraint ensures that if $j \prec i$, then job i can only be scheduled at a time strictly later than job j.

1.2 Overview of Our Algorithm

Let us first give an overview of the algorithm of Levey-Rothvoss [9] since our algorithm builds up on it.

Previous approach

At a high-level, the algorithm of Levey-Rothvoss [9] works by constructing a laminar family of intervals, where the topmost level has one interval $[1, T]$ and each succeeding level is constructed by dividing each interval of the previous level into two equal sized intervals, as shown in the figure below. Thus, there are $(1 + \log T)$ levels where level ℓ contains 2^ℓ intervals, each of size $\frac{T}{2^\ell}$ for $\ell = 0, 1, \ldots, \log T$. This laminar family can be thought of as being a binary tree of depth $\log T$ with the interval $[1, T]$ as the root and the level ℓ intervals being vertices at depth ℓ.

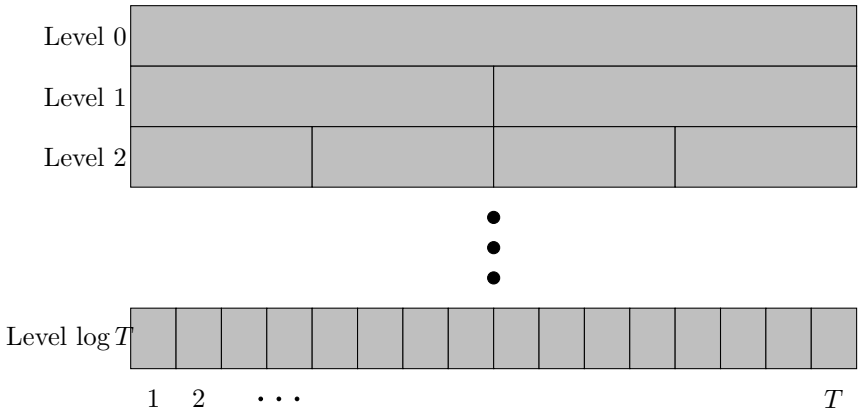


Figure 1 Construction of the laminar family used in the algorithm.

Each job j is first assigned to the smallest interval in this laminar family which fully contains the fractional support of j as per the solution of the LP. Let $k = O(\log \log n)$ and let us call the top k^2 levels in the laminar family as the *top levels* and the level succeeding it, that is the level k^2 as the *bottom level*. Their algorithm conditions (see Section 2 for the definition of conditioning) roughly 2^{k^2} times in order to reduce the maximum chain length among the jobs assigned to the top levels. Once the length of the chains in the top levels is reduced, the last k of the top levels are discarded from the instance, and the sub-instances corresponding to each interval in the bottom level is recursively solved in order to get a partial schedule for all the jobs except those assigned to the top levels. The discarding of the k levels is done in order to create a large gap between the top levels and the bottom level. Having such a gap makes it easier to schedule the remaining jobs in the top levels in the gaps of the partial schedule and Levey-Rothvoss [9] give an elegant algorithm to do this, provided that the maximum chain length among the jobs in the top levels is small. This step increases the makespan by at most a $(1 + \epsilon/\log n)$ factor, which adds up to a loss of a $(1 + \epsilon)$ factor in total over the at most $\log n$ depth of the recursion. Finally, one must also schedule the jobs in the k levels which were discarded; to do this without increasing the makespan by more than a $(1 + \epsilon)$ factor, it suffices to ensure that these k discarded levels contain at most an ϵ fraction of the jobs contained in the top levels. Let us call such a set of k consecutive levels, which contains at most an ϵ fraction of the number of jobs in the levels above it, as a *good batch*.

Now the reason they had to condition $2^{k^2} = 2^{O((\log \log n)^2)}$ times, which leads to the running time of $n^{O(2^{k^2})}$, comes from the fact that they condition on every interval in the top k^2 levels. And this is necessary to ensure that a good batch exists. For example, the number of jobs contained in the levels $[pk, (p + 1)k)$ may be about $e^{\epsilon p} \cdot (\epsilon T/\log n)$ for all $p < (1/\epsilon) \ln(m \log n) \approx k$, in which case there is no good batch in the first $o(k^2)$ levels.

Our approach

To get around the above issue, we observe the following: if, after conditioning on only the top Ck levels, where $C = O(1/\epsilon^2)$ is a big enough constant, there does not exist a good batch in the top Ck levels, then in fact a $(1 - \epsilon)$ fraction of the jobs in the top Ck levels must lie in the last $(1/\epsilon^2)k$ levels, that is, in levels from $Ck - (1/\epsilon^2)k$ to $Ck - 1$. This implies that we can discard the jobs in the first $Ck - (1/\epsilon^2)k$ levels by charging them to the jobs in the levels from $Ck - (1/\epsilon^2)k$ to $Ck - 1$, and in doing so we only discard an ϵ fraction of the total number of jobs.

Notice that we have only conditioned about $2^{Ck} = \text{polylog}(n)$ times till now as there are these many intervals in the top Ck levels. The next crucial observation is that after deleting the top $Ck - (1/\epsilon^2)k$ levels, the sub-instances defined by each of the subtrees rooted at the intervals on the level $Ck - (1/\epsilon^2)k$ can be solved independently of each other. This means that we can perform conditioning in parallel on each such sub-instance, and thus in total, we will condition at most $2^{Ck} \cdot \log T = \text{polylog}(n)$ times, since the depth of the recursion is at most the height of the tree.

Now it might also happen that we already find a good batch in the top Ck levels and in this case, we follow a strategy similar to Levey-Rothvoss [9] by recursing on the bottom intervals to find a partial schedule and fitting the jobs in the top levels in this partial schedule. This step might discard an ϵ fraction of the jobs in the top levels. These two cases, one where we recurse because there is no good batch in the top Ck levels and one where we recurse because there is a good batch in the top Ck levels, might interleave in a complicated manner. We show that the number of jobs ever discarded in the algorithm due to each type of recursion is at most an $O(\epsilon)$ fraction of the total number of jobs, which implies that we can achieve a makespan of $(1 + \epsilon)T$.

The above high-level description skims over a few important issues. One big challenge in the above approach is to ensure that the number of jobs discarded in the cases where we do not find a good batch stays small during the whole algorithm. Even though this is the case in one such recursive call, this might not happen over all the recursive calls taken together and we might end up discarding a constant fraction of the jobs. To get over this obstacle, we carefully control which interval each job is assigned to: if a job j is assigned to an interval I but after conditioning on some job $i \neq j$ which is assigned to a level lower than j, the fractional support of j shrinks to a sub-interval of I, then we will still keep j assigned to I, rather than moving it down the laminar family. This ensures that each job is not charged more than once for discarded jobs and thus the total number of discarded jobs is at most ϵn. This however slightly changes the way jobs are assigned to intervals and the techniques developed by Levey and Rothvoss [9] cannot be immediately applied to fit the jobs of the top levels in the partial schedule of the bottom levels in the case when a good batch exists. To tackle this issue, we will allow each job in the top levels to be scheduled outside of its (current) fractional support as long as it doesn't violate the precedence constraints with the jobs in the bottom levels. With this modification, we will be able to fit the jobs in the top levels in the partial schedule of the bottom levels without discarding more than an ϵ fraction of the top jobs. This implies that in both types of recursions, we only discard an $O(\epsilon)$ fraction of the jobs.

2 Preliminaries on Sherali-Adams Hierarchy

In this section, we state the basic facts about Sherali-Adams hierarchy which we will need. We refer the reader to the excellent surveys [7, 2, 10] for a more extensive introduction to hierarchies.

Consider a linear program with n variables $y_1, \ldots, y_n$ where for each $i \in [n]$, $0 \leq y_i \leq 1$. For $s \geq 0$, the s^{th}-round Sherali-Adams lift of this linear program is another linear program with variables $y_S^{(s)}$ for each $S \subseteq [n]$ satisfying $|S| \leq s+1$, and some additional constraints. We will often denote $y_{\{i\}}^{(s)}$ by $y_i^{(s)}$ for simplicity.

If we think of y_i as the probability that $y_i = 1$, intuitively the variables $y_S^{(s)}$ should equal the probability that each $i \in S$ has $y_i = 1$, that is we would like to have that $y_S^{(s)} = \Pi_{i \in S} y_i$. As these constraints are not convex, we can only impose some linear conditions implied by them. In particular, for every constraint $a^T y \leq b$ of the starting LP, we add, for every $S, T \subseteq [n]$ such that $|S| + |T| \leq s$, a new constraint given by

$$\sum_{T' \subseteq T} (-1)^{|T'|} \left(\sum_{i=1}^{n} a_i y_{S \cup T' \cup \{i\}}^{(s)} - b y_{S \cup T'}^{(s)} \right) \leq 0. \tag{2}$$

If $y_S^{(s)} = \Pi_{i \in S} y_i$ was indeed true for all $|S| \leq s+1$, then the above inequality can be succinctly written as $(a^T y - b) \cdot \Pi_{i \in S} y_i \cdot \Pi_{i \in T}(1 - y_i) \leq 0$, and are thus valid constraints for all $0-1$ solutions.

Observe that an s^{th}-round Sherali-Adams lift of an LP with n variables and m constraints is just another LP with $n^{O(s)}$ variables and $m \cdot n^{O(s)}$ constraints. Letting $y^{(s)}$ denote a feasible solution of the s^{th}-round Sherali-Adams lift, $y^{(s)}$ is also feasible for all $s' \leq s$ rounds of Sherali-Adams and in particular is a feasible solution of the starting LP.

Conditioning

Given a feasible solution $y^{(s)}$ of the s^{th}-round Sherali-Adams lift and $i \in [n]$ such that $y_i^{(s)} > 0$, then we can *condition on the event* $y_i = 1$ to get a feasible solution $z^{(s-1)}$ of the $(s-1)^{th}$-round Sherali-Adams lift defined as

$$z_S^{(s-1)} = \frac{y_{S\cup\{i\}}^{(s)}}{y_i^{(s)}} \qquad \forall S : |S| \leq s.$$

The fact that $z^{(s-1)}$ is a feasible solution of the $(s-1)^{th}$-round Sherali-Adams lift follows easily from (2). Moreover, z satisfies $z_i^{(s-1)} = 1$ and the following useful property:

▶ **Observation 2.** *If for some $j \in [n]$, $y_j^{(s)} = 0$ and we condition on $y_i = 1$ for any $i \in [n]$, then $z_j^{(s-1)} = 0$.*

Proof. Using the Sherali-Adams lift (2) of the constraint $y_i \leq 1$ with $S = \{j\}$ and $T = \phi$, we get $y_{\{i,j\}}^{(s)} \leq y_j^{(s)}$. This gives $z_j^{(s-1)} = \frac{y_{\{i,j\}}^{(s)}}{y_i^{(s)}} \leq \frac{y_j^{(s)}}{y_i^{(s)}} = 0$. ◀

One can think of the solution $z_S^{(s-1)}$ as giving the conditional probability of $y_S^{(s)} = 1$ given $y_i^{(s)} = 1$. By conditioning on a variable y_i to be 1, we will mean that we replace the current fractional solution $y^{(s)}$ with the fractional solution $z^{(s-1)}$ as in above. Observation 2 implies that conditioning can never increase the support of any variable, or in other words, if the probability that $y_j = 1$ is zero, then the conditional probability that $y_j = 1$ conditioned on $y_i = 1$, is also zero.

3 Algorithm

Before we describe our algorithm, we first develop some notation. Let T denote the value of the LP (1) and let $y^{(s)}$ denote the feasible solution of the s^{th}-round Sherali-Adams lift of the LP we get after we condition $r - s$ times in the algorithm. We will say that we are in *round* s of the algorithm if we have conditioned $r - s$ times so far. So we will start the algorithm in round r with solution $y^{(r)}$, and if we condition in round s, we go to round $s-1$ with solution $y^{(s-1)}$.

For each job j, define the *fractional support interval of j in round s* as $F_j^{(s)} := [r_j^s, d_j^s]$, where r_j^s is the smallest time t for which $y_{jt}^{(s)} > 0$ and d_j^s is the largest time for which $y_{jt}^{(s)} > 0$ (r_j and d_j are used to symbolize release time and deadline). In other words, $F_j^{(s)}$ is the minimal interval which fully contains the fractional support of job j in $y^{(s)}$. By Observation 2, upon conditioning, the fractional support interval can only shrink, that is $F_j^{(s-1)} \subseteq F_j^{(s)}$.

For each job j, we also define a *support interval* $S_j^{(s)}$. We will initially set $S_j^{(r)} := F_j^{(r)}$. In later rounds, we will update $S_j^{(s)}$ in such a way that $F_j^{(s)} \subseteq S_j^{(s)} \subseteq F_j^{(r)}$. Intuitively, $S_j^{(s)}$ reflects our knowledge in round s of where j can be scheduled. Notice that we might schedule j outside of the fractional support interval $F_j^{(s)}$.

A schedule of jobs is called a *feasible schedule* if it satisfies the precedence constraints among all the jobs and a *partial feasible schedule* if it schedules some of the jobs and satisfies the precedence constraints among them. In order to get a feasible schedule of all the jobs with a makespan of at most $(1+\epsilon)T$, it suffices to show the following:

▶ **Theorem 3.** *We can find a partial feasible schedule $\sigma : [n] \to [T] \cup \{DISCARDED\}$ such that $\sigma(j) = DISCARDED$ for at most ϵT jobs.*

Clearly a schedule σ as in Theorem 3 has makespan at most T. Having such a partial feasible schedule, we can easily convert it to a feasible schedule of all the n jobs with a makespan of at most $(1+\epsilon)T$: iterate through every job j discarded in σ and find the earliest time t by when all the jobs which must precede j have either already been scheduled or are as of yet discarded. Create a new time slot between times t and $t+1$ containing only job j. This increases the makespan by one for every job discarded in σ.

Laminar Family.

A laminar family of intervals is defined in the following manner. The topmost level, level 0, has one interval $[1, T]$. Each succeeding level is constructed by dividing each interval of the previous level into two equal sized intervals[2]. Thus there are $(1 + \log T)$ levels, where level ℓ contains 2^ℓ intervals each of size $\frac{T}{2^\ell}$ for $\ell = 0, 1, \ldots, \log T$. This laminar family can be thought of as being a binary tree of depth $\log T$ with the interval $[1, T]$ as the root, and the level ℓ intervals as being vertices at depth ℓ.

Let $\mathcal{I}_\ell$ denote the set of intervals at level ℓ of the laminar family. For an interval $I \in \mathcal{I}_\ell$, a *sub-interval of* I is any interval $I' \subseteq I$ of the laminar family, including I itself; and $I_{\text{left}}, I_{\text{right}}$ will denote the left and right sub-intervals respectively of I in $\mathcal{I}_{\ell+1}$. By the midpoint of I, we will mean the right boundary of I_{left}.

Job j is *assigned* to interval I in round s if I is the smallest interval in the laminar family such that $S_j^{(s)} \subseteq I$. This assignment of jobs to intervals depends on $S_j^{(s)}$ and will change as s and $S_j^{(s)}$ change during the algorithm. Let $I^{(s)}(j)$ denote the interval to which j is assigned in round s of the algorithm. For an interval I in the laminar family, let $\mathcal{J}^{(s)}(I)$ denote the set of jobs assigned to I, and let $\mathcal{J}^{(s)}(\mathcal{I}_\ell)$ denote the set of jobs assigned to intervals in $\mathcal{I}_\ell$ in round s of the algorithm.

Batches.

Let $k = \log(\frac{32m}{\epsilon} \cdot \log n)$. For $p \geq 0$, define the p^{th} batch as

$$\mathcal{B}_p = \{\mathcal{I}_{pk}, \mathcal{I}_{pk+1}, \ldots, \mathcal{I}_{(p+1)k-1}\}.$$

That is, it denotes the set of k consecutive levels starting from level pk till level $(p+1)k-1$. Let $\mathcal{J}^{(s)}(\mathcal{B}_p)$ denote the set of jobs assigned to intervals in batch p in round s. Batch $\mathcal{B}_p$ for $p \geq 1$ is called a *good batch with respect to* $[T]$ *in round* s if

$$|\mathcal{J}^{(s)}(\mathcal{B}_p)| \leq \frac{\epsilon}{4m} \sum_{i=0}^{p-1} |\mathcal{J}^{(s)}(\mathcal{B}_i)|. \tag{3}$$

We will omit the "with respect to $[T]$" if it is clear from the context that we start the summation in the right hand side of (3) from the first batch in $[T]$. Similarly, we will omit the "in round s" if s is clear from the context.

Algorithm.

We can now describe our algorithm and split its description in two steps for clearer exposition. Let $C = 2(4m/\epsilon)^2 + 1, k = \log(\frac{32m}{\epsilon} \cdot \log n)$ and $\delta = \frac{\epsilon}{8mCk2^{Ck}\log n}$. The reader can think of

[2] Without loss of generality, T is a power of 2. Otherwise, we can add a few dummy jobs at the end which must succeed all other jobs and which make T a power of 2.

these parameters as being $k = \Theta_{m,\epsilon}(\log\log n)$ and $\delta = \Theta_{m,\epsilon}(1/\text{polylog}(n))$. s will always denote the current round of the algorithm, unless otherwise specified. We initialise $s := r$ and for each job j, $S_j^{(r)} := F_j^{(r)}$.

Schedule($y^{(r)}, T$):[3]

1. **Step 1: Reducing chain length in the top $qk \leq Ck$ levels**

   In this step, we will reduce the length of the chains in each interval I in the top qk levels of the laminar family to at most $\delta|I|$, for some $q \leq C$. This is done by going down the levels, starting from level 0 till level $qk-1$, where q is chosen such that

   a. after having conditioned on all the levels from 0 to $qk-1$, $\mathcal{B}_{q-1}$ is a good batch, or

   b. we have already conditioned on the top Ck levels and found no good batch, in which case we set $q = C$.

   The conditioning on the levels and update of S_j's is done as follows. For $\ell = 0, 1, \ldots, qk-1$:

   - Let $s_{\text{old}} = s$ and for each j, let $\ell(j)$ denote the level of the interval $I^{(s_{\text{old}})}(j)$, that is the level to which j is assigned at the beginning of this iteration of the loop.
   - We go over every interval $I \in \mathcal{I}_\ell$ and do the following: if $\mathcal{J}^{(s)}(I)$ has a chain of length more than $\delta|I|$, let j be the first job in this chain. We condition on j lying in I_{right}. After every conditioning, update $s := s-1$ and set $S_j^{(s)}$ for every job j as follows:
     - if $\ell(j) < \ell$, let m_j denote the midpoint of $I^{(s_{\text{old}})}(j)$ and $[t_r, t_d] := F_j^{(s)}$. If $F_j^{(s)} \subseteq I_{\text{left}}$, then we set $S_j^{(s)} := [t_r, m_j + 1]$, and if $F_j^{(s)} \subseteq I_{\text{right}}$, then we set $S_j^{(s)} := [m_j, t_d]$. Otherwise, set $S_j^{(s)} := F_j^{(s)}$.
     - if $\ell(j) \geq \ell$, set $S_j^{(s)} := F_j^{(s)}$.

   That is, the support intervals $S_j^{(s)}$ are set such that if we condition on jobs in level ℓ, then the jobs assigned to a level $\ell' < \ell$ before the conditionings stay assigned to level ℓ', and for all other jobs, $S_j^{(s)}$ equals the fractional support interval $F_j^{(s)}$.

2. **Step 2: Recursion**

   There are two cases to consider here, depending on which of (a) or (b) took place in the previous step.

   (i) If (a) occurred, perform a recursion of type 1.

   This step is similar to the algorithm of [9]. We discard all the jobs in the good batch $\mathcal{B}_{q-1}$. Then for each interval $I \in \mathcal{I}_{qk}$, we recursively call Schedule$(y^{(s)}, I)$ to obtain a schedule $\tilde{\sigma}_I$, which are put together to form a partial feasible schedule $\tilde{\sigma}$ for the jobs assigned to a level $\ell \geq qk$.

   Then we fit the jobs in the top levels, that is the jobs in $\mathcal{J}^{(s)}(\mathcal{B}_0) \cup \cdots \cup \mathcal{J}^{(s)}(\mathcal{B}_{q-2})$ in the empty slots in $\tilde{\sigma}$. We give more details of how this is done in Section 4.2.1. Some jobs in the top levels will be discarded while doing this.

   Call this step a *recursion of type* 1. The number of jobs discarded in this step, that is the jobs in batch $\mathcal{B}_{q-1}$ along with the jobs in the top levels which are discarded, will be referred to as the jobs discarded due to this step. Notice that this does not include the jobs discarded in each recursive call to the intervals in $\mathcal{I}_{qk}$.

   (ii) If (b) occurred, perform a recursion of type 2.

   In this case, we discard all the jobs in $\mathcal{J}^{(s)}(\mathcal{B}_0) \cup \cdots \cup \mathcal{J}^{(s)}(\mathcal{B}_{C-(4m/\epsilon)^2-1})$. Then for each interval $I \in \mathcal{I}_{(C-(4m/\epsilon)^2)k}$, we recursively call Schedule$(y^{(s)}, I)$ to get a

[3] When the algorithm is called on an interval of length $L \leq 2^{Ck}$, we can just "brute force" by conditioning mL times to find an exact solution. We avoid writing this explicitly in the algorithm for simplicity.

schedule σ_I which are put together to form a partial feasible schedule σ for all the jobs assigned to a level $\ell \geq (C - (4m/\epsilon)^2)k$.

Call this step a *recursion of type 2*. The number of jobs discarded in this step, that is the jobs in batches $\mathcal{B}_0, \ldots, \mathcal{B}_{C-(4m/\epsilon)^2-1}$ will be referred to as the jobs discarded due to this step. Just like before, this does not include the jobs discarded in each recursive call to the intervals in $\mathcal{I}_{(C-(4m/\epsilon)^2)k}$.

In each type of recursion, we recurse on multiple sub-instances defined by intervals of some level. It is important that the recursions on these sub-instances are done independently of each other. That is, we pass the same (current) Sherali-Adams solution to each recursive call, and conditionings done in one recursive call are independent of conditionings done in any other recursive call, and thus do not affect the fractional solution of any other recursive call.

4 Analysis

In this section, we prove Theorem 3 which will imply Theorem 1. We split the analysis into two parts: in the first part, we give a bound on the number of rounds of Sherali-Adams needed in the algorithm. In the second part, we show that we discard at most ϵT jobs during the algorithm and schedule all other jobs by time T, thus proving Theorem 3.

But first, we need to show that the algorithm is well-defined.

▶ **Observation 4.** *In step* 1*, when we condition on an interval I by finding a chain $\mathcal{C}$ in I and conditioning the first job j in this chain to lie in I_{right}, this is possible to do. Moreover, this assigns every job in $\mathcal{C}$ to a sub-interval of I_{right}.*

Proof. For the first part of the observation, we need to show that $F_j^{(s)} \cap I_{\text{right}} \neq \phi$, where s is the round of the algorithm just before we condition on j in I. As j is assigned to I in round s, it must be that $S_j^{(s)} \cap I_{\text{right}} \neq \phi$. The support intervals are updated in a way such that we can only have $S_j^{(s)} \neq F_j^{(s)}$ after we condition on a level below that of j. But because we always condition on the levels from top to bottom, we must have $S_j^{(s)} = F_j^{(s)}$. This proves the first part of the observation.

The moreover part follows easily now since every other job $i \in \mathcal{C}$ satisfies $j \prec i$ and must start scheduling only after j. ◀

4.1 Bounding number of rounds of Sherali-Adams

Let $r(|I|)$ denote the number of rounds of Sherali-Adams the algorithm uses when run on the instance defined by the subtree rooted at interval I of the laminar family. Our goal in this subsection is to show $r(T) \leq r$ for $r = O_{m,\epsilon}(\log^{O(m^2/\epsilon^2)} n)$.

We first give an upper bound on the number of conditionings done in one interval I.

▶ **Lemma 5.** *The algorithm conditions at most m/δ times on any interval I in step* 1*.*

Proof. Let s_{old} denote the round of the algorithm just before we start to condition in I, and let $\ell \geq 0$ be such that $I \in \mathcal{I}_\ell$. Each time we condition in I, we assign at least $\delta|I|$ jobs in I to a sub-interval of I_{right} (by Observation 4).

Also, no job assigned to a level $\ell' < \ell$ in round s_{old} moves down the laminar family during conditionings done in I. And for all other jobs, they only get assigned to a sub-interval. Thus no new job is assigned to I while we are conditioning in I.

Using $F_j^{(s_{\text{old}})} \subseteq S_j^{(s_{\text{old}})}$ and the second constraint of LP (1), there can be at most $m|I|$ jobs in total assigned to I in round s_{old}. Thus, the number of times we condition in I is at most $\frac{m|I|}{\delta|I|} = \frac{m}{\delta}$. ◀

▶ **Lemma 6.** *The algorithm conditions at most $2^{Ck}m/\delta$ times in step 1 of the algorithm.*

Proof. By Lemma 5, we condition at most m/δ times per interval. As we condition on the topmost qk levels and hence on at most $2^{qk} \leq 2^{Ck}$ intervals, we condition at most $2^{Ck}m/\delta$ times in step 1. ◀

In step 2 of the algorithm, if we do a recursion of type 1 then we recurse on every interval at level $qk \geq k$. Otherwise, if we do a recursion of type 2 then we recurse on every interval at level $(C - (4m/\epsilon)^2)k \geq k$. In either case we recurse on every interval of some level $\ell \geq k$ and thus on an interval of size at most $T/2^k$. Because the conditionings done in one recursive call are done independently of the conditionings in any other recursive call, the total number of rounds of Sherali-Adams we need can be bounded by the following recurrence:

$$r(T) \leq \frac{2^{Ck}m}{\delta} + r(T/2^k)$$

where the base case is $r(2^{Ck}) = 2^{Ck}m$, and thus we get

$$r(T) \quad \leq \quad \frac{2^{Ck}m \log T}{\delta} \leq \frac{8m^2Ck(\log^2 n)2^{2Ck}}{\epsilon} = O_{m,\epsilon}((\log n)^{5+\frac{64m^2}{\epsilon^2}}) = r.$$

4.2 Bounding number of jobs discarded

In this subsection, we bound the number of jobs discarded in the algorithm and show that it is at most ϵT. We will separately bound the number of jobs discarded due to recursions of type 1 and recursions of type 2 and show that each is at most $\epsilon T/2$. The former uses a result proved by Levey and Rothvoss [9] but which needs to be heavily adapted to our algorithm. The latter uses a simple charging argument.

4.2.1 Jobs discarded due to recursions of type 1.

Suppose we perform a recursion of type 1 when the algorithm is called on the interval I of the laminar family. To be consistent with the notation of [9], we will call the set of jobs $\mathcal{J}^{(s)}(\mathcal{B}_{q-1})$ as $\mathcal{J}_{\text{middle}}$, the set of jobs $\mathcal{J}^{(s)}(\mathcal{B}_0) \cup \cdots \cup \mathcal{J}^{(s)}(\mathcal{B}_{q-2})$ as $\mathcal{J}_{\text{top}}$ and the jobs in the levels below these as $\mathcal{J}_{\text{bottom}}$ (here we are reindexing the batches such that the first level starts from interval I).

▶ **Claim 7.**

$$|\mathcal{J}_{middle}| \leq \frac{\epsilon}{4m}|\mathcal{J}_{top}|.$$

Proof. Follows from the fact that $\mathcal{B}_{q-1}$ is a good batch and (3). ◀

After discarding all the jobs in $\mathcal{J}_{\text{middle}}$, the algorithm recursively finds a partial feasible schedule $\tilde{\sigma}$ of the jobs in $\mathcal{J}_{\text{bottom}}$. Let $\mathcal{J}' \subseteq \mathcal{J}_{\text{bottom}}$ be the set of jobs scheduled by $\tilde{\sigma}$. The algorithm will then attempt to extend $\tilde{\sigma}$ to a schedule σ of the jobs in $\mathcal{J}_{\text{top}} \cup \mathcal{J}'$. We will be able to do this by discarding only a few jobs from $\mathcal{J}_{\text{top}}$. More formally:

▶ **Lemma 8.** *When the algorithm is called on an interval I, we can extend $\tilde{\sigma}$ to a feasible schedule σ of the jobs in $(\mathcal{J}_{top} \setminus \mathcal{J}_{discard}) \cup \mathcal{J}'$ where*

$$|\mathcal{J}_{discard}| \leq \frac{\epsilon |I|}{4 \log n}.$$

We defer the proof of Lemma 8 to the full version of the paper. We show below how Lemma 8 implies that we discard at most $\epsilon T/2$ jobs in all recursions of type 1.

▶ **Lemma 9.** *Total number of jobs discarded in all recursions of type 1 during the algorithm is at most $\epsilon T/2$.*

Proof. Using Claim 7 and Lemma 8, if we perform a recursion of type 1 when the algorithm is called on the interval I, the number of jobs discarded is at most

$$\frac{\epsilon}{4m} |\mathcal{J}_{\text{top}}| + \frac{\epsilon |I|}{4 \log n}.$$

Over all recursions of type 1, the first term sums up to at most $\epsilon n/4m \leq \epsilon T/4$. For any $\ell \geq 0$, the second term sums up to $\frac{\epsilon T}{4 \log n}$ over all intervals $I \in \mathcal{I}_\ell$. As there are at most $\log n$ levels, the second term also sums up to $\epsilon T/4$ over all recursions of type 1. ◀

4.2.2 Jobs discarded due to recursions of type 2.

Let $\epsilon' = \epsilon/4m$. Recall that in a recursion of type 2, we delete all the jobs assigned to levels 0 to $(C - (1/\epsilon')^2)k - 1$ and retain only the later $(1/\epsilon')^2$ batches. We show below that in such a case, at least a $(1 - \epsilon')$ fraction of the jobs in the top C batches are in the last $(1/\epsilon')^2$ batches and thus, by deleting the jobs in the first $C - (1/\epsilon')^2$ batches we only delete an ϵ' fraction of the jobs.

▶ **Lemma 10.** *If case (b) occurs in step 1 of the algorithm, then*

$$\sum_{i=0}^{C-(1/\epsilon')^2-1} |\mathcal{J}^{(s)}(\mathcal{B}_i)| \leq \epsilon' \sum_{i=C-(1/\epsilon')^2}^{C-1} |\mathcal{J}^{(s)}(\mathcal{B}_i)|. \tag{4}$$

Proof. Let $S = \sum_{i=0}^{C-(1/\epsilon')^2-1} |\mathcal{J}^{(s)}(\mathcal{B}_i)|$, the left hand side of (4). Case (b) occurs in step 1 of the algorithm if none of the batches $\mathcal{B}_p$ for $p \in [C - (1/\epsilon')^2, C - 1]$ are good. But then for $p \in [C - (1/\epsilon')^2, C - 1]$, we must have

$$|\mathcal{J}^{(s)}(\mathcal{B}_p)| > \epsilon' \sum_{i=0}^{p-1} |\mathcal{J}^{(s)}(\mathcal{B}_i)| \geq \epsilon' \sum_{i=0}^{C-(1/\epsilon')^2-1} |\mathcal{J}^{(s)}(\mathcal{B}_i)| = \epsilon' S.$$

This implies (4) as

$$\sum_{i=C-(1/\epsilon')^2}^{C-1} |\mathcal{J}^{(s)}(\mathcal{B}_i)| \geq \sum_{i=C-(1/\epsilon')^2}^{C-1} \epsilon' S = \left(\frac{1}{\epsilon'}\right)^2 \epsilon' S = \frac{S}{\epsilon'}.$$ ◀

This implies that when we discard the top $C - (1/\epsilon')^2$ batches, we are only discarding at most an ϵ' fraction of the jobs in the next $(1/\epsilon')^2$ batches. We can imagine this as putting a charge of ϵ' on every job in the last $(1/\epsilon')^2$ batches. Thus the total charge on all the jobs at the end of the algorithm is an upper bound on the number of jobs discarded in recursions of type 2 during the algorithm.

▶ **Lemma 11.** *For every job j, we put a charge on j at most once.*

Proof. Fix a job j and suppose we put a charge on j at least once. When we put a charge on j for the first time, then in some recursion of type 2 it must have been assigned to the lowest $(1/\epsilon')^2$ batches among the top C batches. The algorithm will then recurse on every interval at level $(C-(1/\epsilon')^2)k$ and thus job j is now in the top $(1/\epsilon')^2$ batches in one of the recursive calls.

Let $I \in \mathcal{I}_{(C-(1/\epsilon')^2)k}$ be such that j is assigned to a sub-interval of I. When we recursively call the algorithm on I, the first $(1/\epsilon')^2$ batches already satisfy the property that any interval I' in them has maximum chain length at most $\delta|I'|$. Thus in step 1 of the algorithm, we will not condition on any interval in the top $(1/\epsilon')^2$ batches. This implies that job j always stays assigned to the top $(1/\epsilon')^2$ batches; this is because the assignment of a job to an interval can only change when we condition on an interval at the same level or at a level above that of the job.

Now suppose we put a charge on j again. Then we must have once again done a recursion of type 2 within the recursive call to I. But j is assigned to the topmost $(1/\epsilon')^2$ batches in this instance and in a recursion of type 2, we delete the topmost $C-(1/\epsilon')^2 > (1/\epsilon')^2$ batches and put a charge on only the later $(1/\epsilon')^2$ batches, which leads to a contradiction. ◀

▶ **Lemma 12.** *Number of jobs discarded in recursions of type 2 throughout the algorithm is at most $\epsilon T/2$.*

Proof. Because the number of jobs discarded in recursions of type 2 throughout the algorithm is at most the total charge on all the jobs and by Lemma 11, each job is charged at most once, we get that the number of jobs discarded in recursions of type 2 is at most

$$\epsilon' n = \epsilon n/4m \leq \epsilon T/4 \leq \epsilon T/2.$$ ◀

References

1. Nikhil Bansal and Subhash Khot. Optimal long code test with one free bit. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 453–462. IEEE, 2009.
2. Eden Chlamtac and Madhur Tulsiani. Convex relaxations and integrality gaps. In *Handbook on semidefinite, conic and polynomial optimization*, pages 139–169. Springer, 2012.
3. Devdatta Gangal and Abhiram Ranade. Precedence constrained scheduling in (2- 7/3p+ 1) optimal. *Journal of Computer and System Sciences*, 74(7):1139–1146, 2008.
4. Michael R Garey. Ds johnson computers and intractability. *A Guide to the Theory of NP-Completeness*, 1979.
5. Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell Labs Technical Journal*, 45(9):1563–1581, 1966.
6. Shui Lam and Ravi Sethi. Worst case analysis of two scheduling algorithms. *SIAM Journal on Computing*, 6(3):518–536, 1977.
7. Monique Laurent. A comparison of the sherali-adams, lovász-schrijver, and lasserre relaxations for 0–1 programming. *Mathematics of Operations Research*, 28(3):470–496, 2003.
8. Jan Karel Lenstra and AHG Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978.
9. Elaine Levey and Thomas Rothvoss. A (1+epsilon)-approximation for makespan scheduling with precedence constraints using LP hierarchies. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 168–177. ACM, 2016. URL: `http://dl.acm.org/citation.cfm?id=2897518`, `doi:10.1145/2897518.2897532`.

10 Thomas Rothvoß. The lasserre hierarchy in approximation algorithms. *Lecture Notes for the MAPSP*, pages 1–25, 2013.

11 Petra Schuurman and Gerhard J Woeginger. Polynomial time approximation algorithms for machine scheduling: Ten open problems. *Journal of Scheduling*, 2(5):203–213, 1999.

12 Hanif D Sherali and Warren P Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990.

13 Ola Svensson. Conditional hardness of precedence constrained scheduling on identical machines. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 745–754. ACM, 2010.

ARRIVAL: Next Stop in CLS

Bernd Gärtner
Department of Computer Science, ETH Zürich, Switzerland
gaertner@inf.ethz.ch

Thomas Dueholm Hansen[1]
Department of Computer Science, University of Copenhagen, Denmark
dueholm@di.ku.dk

Pavel Hubáček[2]
Computer Science Institute, Charles University, Prague, Czech Republic
hubacek@iuuk.mff.cuni.cz

Karel Král[3]
Computer Science Institute, Charles University, Prague, Czech Republic
kralka@iuuk.mff.cuni.cz

Hagar Mosaad
Department of Computer Science and Engineering, German University in Cairo, Egypt
hagar.omar@student.guc.edu.eg

Veronika Slívová[4]
Computer Science Institute, Charles University, Prague, Czech Republic
slivova@iuuk.mff.cuni.cz

Abstract

We study the computational complexity of Arrival, a zero-player game on n-vertex switch graphs introduced by Dohrau, Gärtner, Kohler, Matoušek, and Welzl. They showed that the problem of deciding termination of this game is contained in $\mathsf{NP} \cap \mathsf{coNP}$. Karthik C. S. recently introduced a search variant of Arrival and showed that it is in the complexity class PLS. In this work, we significantly improve the known upper bounds for both the decision and the search variants of Arrival.

First, we resolve a question suggested by Dohrau et al. and show that the decision variant of Arrival is in $\mathsf{UP} \cap \mathsf{coUP}$. Second, we prove that the search variant of Arrival is contained in CLS. Third, we give a randomized $\mathcal{O}(1.4143^n)$-time algorithm to solve both variants.

Our main technical contributions are (a) an efficiently verifiable characterization of the unique witness for termination of the Arrival game, and (b) an efficient way of sampling from the state space of the game. We show that the problem of finding the unique witness is contained in CLS, whereas it was previously conjectured to be FPSPACE-complete. The efficient sampling procedure yields the first algorithm for the problem that has expected runtime $\mathcal{O}(c^n)$ with $c < 2$.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases CLS, switch graphs, zero-player game, $\mathsf{UP} \cap \mathsf{coUP}$

[1] Partially supported by BARC which is funded by the VILLUM Foundation grant 16582.
[2] Supported by the project 17-09142S of GA ČR, Charles University project UNCE/SCI/004, and Charles University project PRIMUS/17/SCI/9. This work was done under financial support of the Neuron Fund for the support of science.
[3] The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 616787.
[4] Supported by the project 17-09142S of GA ČR, Charles University project PRIMUS/17/SCI/9, and Charles University grant SVV-2017-260452.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 60; pp. 60:1–60:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.60

Related Version A full version of the paper is available at [8], https://arxiv.org/abs/1802.07702.

Acknowledgements We wish to thank Karthik C. S. for helpful discussions and suggestions.

1 Introduction

Variants of switch graphs have applications and are studied for example in combinatorics and in automata theory (cf. [11] and the references therein). Dohrau et al. [5] introduced Arrival, a natural computational problem on switch graphs, which they informally described as follows:

> Suppose that a train is running along a railway network, starting from a designated origin, with the goal of reaching a designated destination. The network, however, is of a special nature: every time the train traverses a switch, the switch will change its position immediately afterwards. Hence, the next time the train traverses the same switch, the other direction will be taken, so that directions alternate with each traversal of the switch.
> Given a network with origin and destination, what is the complexity of deciding whether the train, starting at the origin, will eventually reach the destination? [5]

The above rather straightforward question remains unresolved. Dohrau et al. [5] showed that deciding Arrival is unlikely to be NP-complete (by demonstrating that it is in NP ∩ coNP), but it is currently not known to be efficiently solvable.

To determine whether the train eventually reaches its destination, it is natural to consider a *run profile*, i.e., the complete transcript describing how many times the train traversed each edge. Dohrau et al. [5] presented a natural integer programming interpretation of run profiles called *switching flows*, which have the advantage of being trivial to verify. The downside of switching flows is that they do not guarantee to faithfully represent the number of times each edge has been traversed; a switching flow might contain superfluous circulations compared to a valid run profile. Nevertheless, Dohrau et al. [5] proved that the existence of a switching flow implies that the train reaches its destination, and thus a switching flow constitutes an NP witness for Arrival.

The coNP membership was shown by an insightful observation about the structure of switch graphs. Specifically, the train reaches its destination d if and only if it never enters a node from which there is no directed path to d. The railway network can thus be altered so that all such vertices point to an additional "dead-end" vertex $\bar{d}$. The coNP witness is then simply a switching flow from the origin to the dead-end $\bar{d}$.

Given that the decision variant of Arrival is in NP ∩ coNP, it is natural to study the search complexity of Arrival in the context of total search problems with the guaranteed existence of a solution, i.e., within the complexity class TFNP (which contains the search analogue of NP ∩ coNP). Total search problems are classified into subclasses of TFNP using the methodology proposed by Papadimitriou [13] that clusters computational problems according to the type of argument assuring the existence of a solution. Karthik C. S. [14] noticed that the search for a switching flow is a prime candidate to fit into the hierarchy of TFNP problems. He introduced S-Arrival, a search version of Arrival that seeks a switching flow to either the destination d or the dead-end vertex $\bar{d}$, and showed that it is contained in the complexity class PLS [10] of total problems amenable to local search.

Fearnley et al. [6] recently studied multiple variants of reachability games on switch graphs and as one of their results gave a lower bound on the complexity of deciding ARRIVAL. Specifically, they showed that Arrival is NL-hard.

1.1 Our Results

One of the open problems suggested by Dohrau et al. [5] was whether deciding the termination of Arrival is contained in UP ∩ coUP (recall that UP is a subclass of NP such that for each YES-instance there is a unique certificate). Recall that given a railway network with an origin o and a destination d, the transcript of the route of the train captured in the run profile from o to d (if it exists) is unique. We show that it is possible to efficiently decide whether a switching flow corresponds to a run profile, which provides a positive answer to the above question and places Arrival inside UP ∩ coUP. We similarly also improve the upper bound on the search complexity of Arrival: We show that S-Arrival is contained in the complexity class CLS. Daskalakis and Papadimitriou [4] introduced CLS to classify problems that can be reduced to local search over continuous domains. CLS contains multiple important search problems such as solving simple stochastic games, finding equilibria in congestion games, and solving linear complementarity problems on P-matrices. For all of these problems, as well as for S-Arrival, we currently do not have a polynomial time algorithm, and they are not known to be complete for some subclass of TFNP.

We establish the containment in CLS through a reduction to End-Of-Metered-Line (EOML), a total search problem that was recently introduced by Hubáček and Yogev [9] who also showed that it is in CLS. In EOML we are given a source in a directed graph with vertices of in-degree and out-degree at most one, and the task is to find a sink or a source different from the given trivial source. The access to the graph is given locally via information about the successor and predecessor of each vertex together with its distance from the trivial source (for the formal definition see Definition 18).

Our result makes it unlikely for S-Arrival to be PLS-hard, which was one of the possibilities suggested by the containment in PLS shown by Karthik C. S. [14]. This is due to known black-box separations among subclasses of TFNP [12, 2], which suggest that CLS is a proper subclass of PLS. Note that our reduction from S-Arrival to End-Of-Metered-Line results in instances with a significantly restricted structure: the End-Of-Metered-Line graph consists only of a single path and many isolated vertices. We believe that this structure may in future work be used to show that S-Arrival is contained in FP.

Our reduction from S-Arrival to End-Of-Metered-Line also implies that we can use an algorithm by Aldous [1] to solve S-Arrival. The algorithm is randomized and runs in $\mathcal{O}(2^{n/2}poly(n))$ expected time on switch graphs with n vertices. This is the first algorithm with expected runtime $\mathcal{O}(c^n)$ for $c < 2$. (A trivial $\mathcal{O}(2^n poly(n))$ time algorithm can be obtained by following the path of the train through the network.) Aldous' algorithm, in fact, solves any problem in PLS. It samples a large number of candidate solutions and then performs a local search from the best sampled solution. The advantage of our reduction is that the resulting search space for End-Of-Metered-Line is small enough to make Aldous' algorithm useful, unlike in the previous reduction by Karthik C. S. [14] that showed containment in PLS.

Fearnley et al. [7] recently gave a reduction from P-matrix linear complementarity problems (PLCP) to End-Of-Metered-Line. As in our case for Arrival, this implies that Aldous' algorithm can be used to solve PLCP. In fact this gives the fastest known randomized algorithm for PLCP, running in expected time $\mathcal{O}(2^{n/2}poly(n))$ for input matrices of dimension $n \times n$. Fearnley et al. do not make this observation themselves, but it is

straightforward to check that their reduction also gives an efficient representation of the search space. Although Aldous' algorithm is very simple, it non-trivially improves the best runtime of algorithms for multiple problems. We believe that this way of applying Aldous' algorithm is a powerful technique that will produce additional results in the future.

1.2 Technical remarks

Recall that a switching flow is a run profile with additional superfluous circulations compared to the valid run profile. Our main technical observation is a characterization of switching flows that correspond to the valid run profile. Given a switch graph G and a switching flow f, we consider the subgraph G^* induced over the railway network by the "last-used" edges; for every vertex v, we include in G^* only the outgoing edge that was, according to the switching flow, used by the train last time it left from v. Note that such last-used edges can be efficiently identified simply by considering the parity of the total number of visits at every vertex. When f is a valid run profile, then it is straightforward to see that the subgraph G^* is acyclic. We show that this property is in fact a characterization, i.e., any switching flow for which the induced graph G^* is acyclic must be a run profile. Given that this property is easy to check, we can use it to efficiently verify run profiles as UP witnesses. (The coUP witness is then a run profile to the dead-end at $\bar{d}$.)

For our reduction from S-Arrival to End-Of-Metered-Line we extend the above observation to partial switching flows that are not required to end at the destination. The vertices of the End-Of-Metered-Line graph created by our reduction correspond to partial switching flows in the S-Arrival instance. The directed edges connect partial run profiles to their natural successors and predecessors, i.e., the partial run extended or shortened by a single step of the train. Any switching flow that does not correspond to some partial run profile is an isolated vertex in the End-Of-Metered-Line graph. Finally, the trivial source is the empty switching flow, and the distance from it can be computed for any partial run simply as the number of steps taken by the train so far. Given that there is only a single path in the resulting End-Of-Metered-Line graph and that its sink is exactly the complete run, we get that the unique solution to the End-Of-Metered-Line instance gives us a solution for the original instance of S-Arrival.

To make the reduction efficiently computable, we need to address the verification of partial run profiles. As it turns out, partial run profiles can be efficiently verified using the graph G^*, in a similar way to complete run profiles discussed above. The main difference is that the graph of last-used edges for a partial run profile can contain a cycle, as the train might visit the same vertex multiple times on its route to the destination. However, we show that there is at most one cycle in G^*, which always contains the current end-vertex of the partial run. The complete characterization of partial run profiles (which covers also full run profiles) is given in Lemma 9, and the formal reduction is described in Section 4.2.1.

Finally, we show that every partial run profile is uniquely determined by its last-used edges and its end-vertex. This limits the size of the search space for the EOML instances that are produced by our reduction, which allows us to efficiently use Aldous' algorithm [1] to solve Arrival and S-Arrival.

2 Preliminaries

In the rest of the paper we use the following standard notation. For $k \in \mathbb{N}$, we denote by $[k]$ the set $\{1, \ldots, k\}$. For a graph $G = (V, E)$, we reserve $n = |V|$ for the number of vertices. The basic object that we study are switch graphs, as defined by Dohrau et al. [5].

Algorithm 1: RUN.

Input : a switch graph $G = (V, E, s_0, s_1)$ and two vertices $o, d \in V$
Output : for each edge $e \in E$, the number of times the train traversed e

```
1  v ← o                                                  // position of the train
2  ∀u ∈ V set s_curr[u] ← s_0(u) and s_next[u] ← s_1(u)
3  ∀e ∈ E set r[e] ← 0                                    // initialize the run profile
4  step ← 0 while v ≠ d do
5  |  (v, w) ← s_curr[v]                                 // compute the next vertex
6  |  r[s_curr[v]]++                                     // update the run profile
7  |  swap(s_curr[v], s_next[v])
8  |  v ← w                                              // move the train
9  |  step ← step + 1
10 return r
```

▶ **Definition 1** (switch graph). A *switch graph* is a tuple $G = (V, E, s_0, s_1)$ where $s_0, s_1 \colon V \to V$ and $E = \{(v, s_0(v)), (v, s_1(v)) \mid \forall v \in V\}$.[5] In order to avoid cumbersome notation, we slightly overload the use of s_0, s_1 and treat both as functions from vertices to edges; that is by $s_b(v)$ we denote the edge $(v, s_b(v))$ for $b \in \{0, 1\}$. We use this convention throughout the paper unless stated otherwise.

The ARRIVAL problem was formally defined by Dohrau et al. [5] as follows.

▶ **Definition 2** (Arrival [5]). Given a switch graph $G = (V, E, s_0, s_1)$ and two vertices $o, d \in V$, the ARRIVAL problem is to decide whether the algorithm RUN (Algorithm 1) terminates, i.e., whether the train reaches the destination d starting from the origin o.

To simplify theorem statements and our proofs, we assume without loss of generality that both $s_0(d)$ and $s_1(d)$ end in d.

A natural witness for termination of the RUN procedure considered in previous work (e.g. [5]) is a switching flow. We extend the definition of a switching flow to allow for partial switching flows that do not necessarily end in the desired destination d.

▶ **Definition 3** ((partial) switching flow, end-vertex). Let $G = (V, E, s_0, s_1)$ be a switch graph. For $o, d \in V$, we say that $\boldsymbol{f} \in \mathbb{N}^{2n}$ is a *switching flow from o to d* if the following two conditions hold.

Kirchhoff's Law (flow conservation):

$$\forall v \in V \colon \sum_{e=(u,v)\in E} \boldsymbol{f}_e - \sum_{e=(v,w)\in E} \boldsymbol{f}_e = [v = d] - [v = o] \ ,$$

where $[\cdot]$ is the indicator variable of the event in brackets.

Parity Condition:

$$\forall v \in V \colon \boldsymbol{f}_{s_1(v)} \leq \boldsymbol{f}_{s_0(v)} \leq \boldsymbol{f}_{s_1(v)} + 1 \ .$$

Kirchoff's law means that o emits one unit of flow, d absorbs one unit of flow, and at all other vertices, in-flow equals out-flow. If $d = o$, we have a circulation.

Given an instance $(G = (V, E, s_0, s_1), o, d)$ of ARRIVAL, we say that $\boldsymbol{f}$ is a *switching flow* if it is a switching flow from o to d. A vector $\boldsymbol{f} \in \mathbb{N}^{2n}$ is called a *partial switching flow* iff $\boldsymbol{f}$ is a switching flow from o to v for some vertex $v \in V$. We say that v is the *end-vertex* of the partial switching flow. We denote the end-vertex of $\boldsymbol{f}$ by $v_{\boldsymbol{f}}$.

[5] Whenever $s_0(v) = s_1(v)$ for some vertex $v \in V$ we depict them as multiple edges in figures.

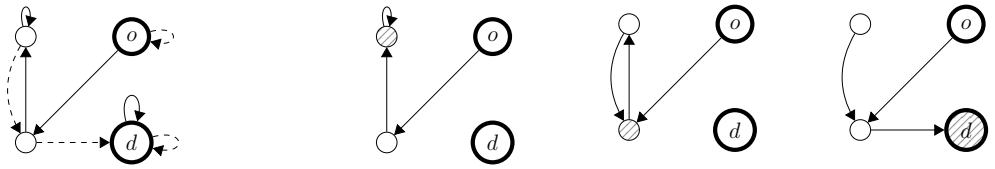


Figure 1 An example of a switch graph G and cycles in the graphs G^* corresponding to partial run profiles after 3, 4, and 5 steps of the train (respectively from left to right). We use hatching to highlight the current end-vertex.

▶ **Definition 4** ((partial) run profile)**.** A *run profile* is the switching flow $\boldsymbol{r}$ returned by the algorithm Run (Algorithm 1) upon termination. A *partial run profile* is a partial switching flow corresponding to some intermediate value of $\boldsymbol{r}$ in the algorithm Run (Algorithm 1).

▶ **Observation 5** (Dohrau et al. [5, Observation 1])**.** *Each (partial) run profile is a (partial) switching flow.*

▶ **Observation 6.** *An end-vertex $v_{\boldsymbol{f}}$ of a switching flow $\boldsymbol{f}$ is computable in polynomial time.*

Proof. It is sufficient to determine which vertex has a net in-flow of one. ◀

3 The Complexity of Run Profile Verification

Dohrau et al. [5] proved that it is possible to efficiently verify whether a given vector is a switching flow. In this section we show that we can also efficiently verify whether a switching flow is a run profile. Combining this with the results by Dohrau et al. [5], we prove that the decision problem of Arrival is in $\mathsf{UP} \cap \mathsf{coUP}$ (see Section 4.1) and that the search problem of Arrival lies in the complexity class CLS (see Section 4.2). As outlined in Section 1.2, our approach for verification of run profiles is based on finding a cycle in a natural subgraph of the railway network G defined below. Specifically, we consider the subgraph of G that contains only the last visited outgoing edge of each vertex, i.e., every vertex has out-degree at most one.

▶ **Definition 7** ($G^*_{\boldsymbol{f}}$)**.** Let $(G = (V, E, s_0, s_1), o, d)$ be an instance of Arrival, and let $\boldsymbol{f} \in \mathbb{N}^{2n}$ be a partial switching flow. We define a graph $G^*_{\boldsymbol{f}} = (V, E^*)$ as follows

$$E^* = \left\{ s_0(v) \colon \forall v \in V \text{ s.t. } \boldsymbol{f}_{s_0(v)} \neq \boldsymbol{f}_{s_1(v)} \right\} \cup \left\{ s_1(v) \colon \forall v \in V \text{ s.t. } \boldsymbol{f}_{s_0(v)} = \boldsymbol{f}_{s_1(v)} > 0 \right\}.$$

▶ **Observation 8.** *Given a partial switching flow $\boldsymbol{f}$, the graph $G^*_{\boldsymbol{f}}$ can be computed in polynomial time.*

▶ **Lemma 9.** *A partial switching flow $\boldsymbol{f}$ is a partial run profile iff $\boldsymbol{f}_{s_0(d)} = \boldsymbol{f}_{s_1(d)} = 0$ and one of the following two conditions holds:*

1. *There exists no cycle in $G^*_{\boldsymbol{f}}$.*
2. *There exists exactly one cycle in $G^*_{\boldsymbol{f}}$ and this cycle contains the end-vertex of $\boldsymbol{f}$.*

The main idea of the proof is based on the following fact: a switching flow f which is *not* a run profile must contain a circulation (as shown by Dohrau et al. [5]). Let f be a switching flow that we get from a run profile r by adding some flows on cycles, then the last added circulation (the last added cycle) must form a cycle in the corresponding graph G^*_f. On the

other hand, a cycle containing the end-vertex is formed in G^*_f whenever the train arrives to a previously visited vertex. An illustration of the graph G^* at consecutive steps of the algorithm Run, with the corresponding evolution of the end-vertices and cycles, is given in Figure 1.

The complete proof of Lemma 9 is provided in the full version [8].

▶ **Lemma 10.** *It is possible to verify in polynomial time whether a vector is a run profile.*

Proof. We can check that a vector $\boldsymbol{f}$ is a switching flow in polynomial time due to Dohrau et al. [5]. The construction of the graph $G^*_{\boldsymbol{f}}$ is polynomial by Observation 8. Lemma 9 gives us a polynomial time procedure to check if $\boldsymbol{f}$ is also a run profile as it is sufficient to check if $G^*_{\boldsymbol{f}}$ contains more than one cycle or whether it has a cycle not containing the end-vertex. This check can be done by a simple modification of the standard depth-first search on $G^*_{\boldsymbol{f}}$. ◀

4 The Computational Complexity of Arrival

In this section we use our efficient structural characterization of run profiles from Lemma 9 to improve the known results about the computational complexity of Arrival. Specifically, we show that the decision version of Arrival is in $\mathsf{UP} \cap \mathsf{coUP}$ and the search version is in CLS.

4.1 The Decision Complexity of Arrival

Our upper bound on the decision complexity of Arrival follows directly from the work of Dohrau et al. [5] by application of Lemma 10.

▶ **Theorem 11.** Arrival *is in* $\mathsf{UP} \cap \mathsf{coUP}$.

Proof. The unique UP certificate for a YES-instance of Arrival is the run profile $\boldsymbol{r}$ returned by the algorithm Run. Clearly, for each YES-instance there exists only one such vector $\boldsymbol{r}$ and $\boldsymbol{r}$ does not exist for NO-instances. By Lemma 10, we can determine whether a candidate switching flow $\boldsymbol{r}$ is a run profile in polynomial time.

The coUP membership follows directly from the reduction of NO-instances of Arrival to YES-instances of Arrival as suggested by Dohrau et al. [5]. The reduction adds to the original graph G a new vertex $\bar{d}$, and for each vertex $v \in V$ such that there is no directed path from v to the destination d, the edges $s_0(v)$ and $s_1(v)$ are replaced with edges $(v, \bar{d})$. This alteration of the original switch graph can be performed in polynomial time. Dohrau et al. [5] proved that the train eventually arrives either at d or $\bar{d}$. The unique coUP witness for Arrival is then a run profile from o to the dead-end $\bar{d}$. ◀

4.2 The Search Complexity of Arrival

The search complexity of Arrival was first studied by Karthik C. S. [14], who introduced a total search variant of Arrival as follows.

▶ **Definition 12** (S-Arrival [14])**.** Given a switch graph $G = (V, E, s_0, s_1)$ and a pair of vertices $o, d \in V$, define a graph G' as follows:

1. Add a new vertex $\bar{d}$.
2. For each vertex v such that there is no directed path from v to d, replace edges $s_0(v)$ and $s_1(v)$ with edges $(v, \bar{d})$.
3. Edges $s_0(d), s_1(d), s_0(\bar{d})$, and $s_1(\bar{d})$ are self-loops.

The problem S-Arrival is to find a switching flow in G' either from o to d or from o to $\bar{d}$.

The above Definition 12 is motivated by the proof of membership in NP ∩ coNP by Dohrau et al. [5]. Namely, in order to ensure that a solution for S-Arrival always exists, it was necessary to add to the switch graph G the dead-end vertex $\bar{d}$.

Note that our method for efficient verification of run profiles from Lemma 10 allows us to define a more natural version of S-Arrival directly on the graph G without any modifications. Instead of relying on the dead-end vertices, we can use the fact that a partial run profile with an edge that was visited for $2^n + 1$ times is an efficiently verifiable witness for NO-instances of Arrival.

▶ **Definition 13** (S-Arrival - simplified)**.** Given a switch graph $G = (V, E, s_0, s_1)$ and a pair of vertices $o, d \in V$, the S-Arrival problem asks us to find one of the following:

1. a run profile $\boldsymbol{r} \in \mathbb{N}^{[2n]}$ from o to d, or
2. a run profile $\boldsymbol{r} \in \mathbb{N}^{[2n]}$ from o to any $v \in V$ such that
   - $\boldsymbol{r}_{(u,v)} = 2^n + 1$, where u is the last vertex visited by the train before it reached the end-vertex v of $\boldsymbol{r}$, and
   - $\boldsymbol{r}_{e'} \leq 2^n$ for all $e' \neq (u, v)$.

The correspondence of the above version of S-Arrival to the original one follows formally from the following lemma.

▶ **Lemma 14** (Karthik C. S. [14, Lemma 1])**.** *For any $G = (V, E, s_0, s_1)$ and a pair of vertices $o, d \in V$. Let $\boldsymbol{r}$ be a run profile (thus $v_{\boldsymbol{r}} = d$), then $\boldsymbol{r}_e \leq 2^n$ for each edge $e \in E$.*

To argue membership of our version of S-Arrival in TFNP, we need to show that both types of solutions in Definition 13 can be verified efficiently. Solutions of the first type are simply run profiles, and we have already shown that they can be verified in polynomial time in Lemma 10. In order to be able to verify solutions of the second type, it remains to argue that for any partial run profile, the immediate predecessor of its end-vertex can be determined in polynomial time.

▶ **Lemma 15.** *Let $\boldsymbol{r}$ be a partial run profile after $R \geq 1$ steps and u be the vertex visited by the train at step $R - 1$. Then*

1. *either u is the unique predecessor of $v_{\boldsymbol{r}}$ in $G^*_{\boldsymbol{r}}$, or*
2. *there is a single cycle in $G^*_{\boldsymbol{r}}$ containing $v_{\boldsymbol{r}}$ and u is the predecessor of $v_{\boldsymbol{r}}$ on this cycle.*

Proof. First, note that if u is the end-vertex one step before $v_{\boldsymbol{r}}$ becomes the end-vertex then $G^*_{\boldsymbol{r}}$ must contain the edge $(u, v_{\boldsymbol{r}})$, as it is the last edge used by the train to leave u. Thus, in the first case (when $v_{\boldsymbol{r}}$ has only one predecessor in $G^*_{\boldsymbol{r}}$) the immediate predecessor of $v_{\boldsymbol{r}}$ in the partial run $\boldsymbol{r}$ is unambiguously given by the only predecessor of $v_{\boldsymbol{r}}$ in $G^*_{\boldsymbol{r}}$.

For the second case we show that $G^*_{\boldsymbol{r}}$ contains a directed cycle C (containing the end-vertex $v_{\boldsymbol{r}}$) and u is unambiguously given by the predecessor of $v_{\boldsymbol{r}}$ in $G^*_{\boldsymbol{r}}$ that lies on C. We find the cycle C by constructing the longest possible directed path $c_0 = v_{\boldsymbol{r}}, c_1, \ldots, c_k$ in $G^*_{\boldsymbol{r}}$ without repeating vertices. Note that it cannot happen that c_k has no outgoing edge in $G^*_{\boldsymbol{r}}$. Otherwise, $\boldsymbol{r}$ would have two different end-vertices $v_{\boldsymbol{r}}$ and c_k (as having no outgoing edge in $G^*_{\boldsymbol{r}}$ means that the train has never left this vertex). By Lemma 9, the directed edge from c_k has to end in the end-vertex $v_{\boldsymbol{r}}$, or else there would be a cycle in $G^*_{\boldsymbol{r}}$ that avoids $v_{\boldsymbol{r}}$.

The algorithm Run takes R steps to generate the run profile $\boldsymbol{r}$, i.e., $\sum_{e \in E} \boldsymbol{r}_e = R$. Let $t_{\boldsymbol{r}} \colon V \to \{0, 1, \ldots, R - 1\}$ be the function returning the last step after which a vertex was left by the train in the partial run profile $\boldsymbol{r}$. Observe that, except for the edge through which

the train arrived to $v_{\boldsymbol{r}}$,[6] it holds for all edges $(x, y) \in G^*_{\boldsymbol{r}}$ that $t_{\boldsymbol{r}}(x) < t_{\boldsymbol{r}}(x) + 1 \leq t_{\boldsymbol{r}}(y)$. However, the above inequality cannot hold for all edges on the cycle C, and thus C has to contain the last used edge and the train had to be in c_k at step $R - 1$. ◀

▶ **Observation 16.** S-Arrival *from Definition 12 reduces to simplified* S-Arrival *from Definition 13.*

Proof. Given a solution of the second type of the simplified S-Arrival, i.e., the long run profile, we can get a run profile $\boldsymbol{r}$ to $\bar{d}$ in polynomial time. For each vertex u we can determine whether there is an oriented path from it to the destination d, and if there is no such path we set $\boldsymbol{r}_{s_0(v)} = \boldsymbol{r}_{s_1(v)} = 0$. We compute the end vertex $v_{\boldsymbol{r}}$ and set $s_0(v_{\boldsymbol{r}}) = 1$. All other components of $\boldsymbol{r}$ are set according to the original solution of the simplified S-Arrival. ◀

4.2.1 S-Arrival is in CLS

Karthik C. S. [14] showed that S-Arrival is contained in the class PLS. We improve this result and prove that S-Arrival is in fact contained in CLS. As a by-product, we also obtain a randomized algorithm for S-Arrival with runtime $\mathcal{O}(1.4143^n)$ which is the first algorithm for this problem with expected runtime $\mathcal{O}(c^n)$ for $c < 2$.

The class of total search problems that are amenable to "continuous" local search was defined by Daskalakis and Papadimitriou [4] using the following canonical problem.

▶ **Definition 17** (CLS [4])**.** CLS is the class of total search problems reducible to the following problem called CLOpt.

Given two arithmetic circuits $f\colon [0,1]^3 \to [0,1]^3$ and $p\colon [0,1]^3 \to [0,1]$, and two real constants $\varepsilon, \lambda > 0$, find either a point $x \in [0,1]^3$ such that $p(f(x)) \leq p(x) + \varepsilon$ or a pair of points $x, x' \in [0,1]^3$ certifying that either p or f is not λ-Lipschitz.

Instead of working with CLOpt, we use as a gateway for our reduction a problem called End-Of-Metered-Line (EOML) which was recently defined and shown to lie in CLS by Hubáček and Yogev [9].

▶ **Definition 18** (End-Of-Metered-Line)**.** Given circuits $S, P\colon \{0,1\}^m \to \{0,1\}^m$, and $V\colon \{0,1\}^m \to [2^m] \cup \{0\}$ such that $P(0^m) = 0^m \neq S(0^m)$ and $V(0^m) = 1$, find a string $x \in \{0,1\}^m$ satisfying one of the following:

1. either $P(S(x)) \neq x$ or $S(P(x)) \neq x \neq 0^m$,
2. $x \neq 0^m$ and $V(x) = 1$,
3. either $V(x) > 0$ and $V(S(x)) - V(x) \neq 1$ or $V(x) > 1$ and $V(x) - V(P(x)) \neq 1$.

The circuits S, P from Definition 18 implicitly represent a directed graph with vertices labelled by binary strings of length m, where each vertex has both out-degree and in-degree at most one. The circuit P represents the predecessor and the circuit S represents the successor of a given vertex as follows: there is an edge from a vertex u to a vertex v iff $S(u) = v$ and $P(v) = u$. Finally, the circuit V can be thought of as an odometer that returns the distance from the trivial source at 0^m or value 0 for vertices lying off the path starting at the trivial source. The task in End-Of-Metered-Line is to find a sink or a source different from the trivial one at 0^m (the solutions of the second and of the third type in Definition 18 ensure that V behaves as explained above).

We are now ready to present our reduction from S-Arrival to End-Of-Metered-Line.

[6] The inequality does not hold for $v_{\boldsymbol{r}}$, since $t(v_{\boldsymbol{r}})$ has not been updated to the time R yet.

▶ **Theorem 19.** S-Arrival *can be reduced to* End-Of-Metered-Line*, and thus it is contained in* CLS*.*

Proof. Let (G, o, d) be an instance of S-Arrival. We construct an instance of EOML that contains a vertex for each candidate partial switching flow over the switch graph G, i.e., for each vector with $2n$ coordinates and values from $[2^n + 1] \cup \{0\}$. The EOML instance will comprise of a directed path starting at the initial (empty) partial run profile 0^{2n}. Each vertex on the path has an outgoing edge to its consecutive partial run profile. Any vertex that does not correspond to a partial run profile becomes a self-loop. Finally, the valuation circuit V returns either the number of steps in the corresponding partial run profile or the zero value if the vertex does not correspond to a partial run profile. Formal description of the circuits S, P, and V defining the above EOML graph is provided in the full version [8]. A polynomial bound on the size of the circuits S, P, and V follows directly from Observation 8 (computing G^*), Lemma 10 (testing whether a given vector is a partial run profile), Observation 6 (computing the end-vertex), and Lemma 15 (computing the previous position of the train).

Lemma 9 and Lemma 15 imply that the EOML graph indeed consists of a single directed path and isolated vertices with self-loops. By the construction of V (it outputs the number of steps of the train), there are no solutions of the second or the third type (cf. Definition 18). Thus, the EOML instance has a unique solution which has to correspond to a run profile in the original S-Arrival instance or to a partial run profile certifying that the train ran for too long (see the second type of solution in Definition 13). ◀

5 An $\mathcal{O}(1.4143^n)$ Algorithm for S-Arrival

Consider any problem that can be put into the complexity class PLS, i.e., can be reduced to the canonical PLS-complete problem LocalOpt (see also [14, Definition 1]):

▶ **Definition 20** (LocalOpt)**.** Given circuits $S\colon \{0,1\}^m \to \{0,1\}^m$, and $V\colon \{0,1\}^m \to [2^m] \cup \{0\}$, find a string $x \in \{0,1\}^m$ such that $V(x) \geq V(S(x))$.

Aldous [1] introduced the following simple algorithm that can be used to solve LocalOpt: pick $2^{m/2}$ binary strings uniformly and independently at random from $\{0,1\}^m$, and let $x_{\max}$ be the selected string that maximizes the value $V(x)$. Starting from $x = x_{\max}$, repeatedly move to the successor $S(x)$, until $V(x) \geq V(S(x))$. He showed that the expected number of circuit evaluations performed by the algorithm before finding a local optimum is at most $\mathcal{O}(m2^{m/2})$. Note that in the case of EOML it is possible that all sampled solutions are isolated vertices in the EOML graph. In this case the 0^m string has the best known value, and the search is started from there.

In the case of S-Arrival, the PLS membership proof of Karthik C. S. constructs the circuits for successor and valuation with $\mathcal{O}(n^2)$ input bits [14, Theorem 2]; the EOML instance constructed in Theorem 19 – proving CLS and in particular PLS membership – yields circuits of $\mathcal{O}(n^2)$ input bits as well. This number is too high to yield a randomized algorithm of non-trivial runtime.

This number of $\mathcal{O}(n^2)$ input bits comes from the obvious encoding of a partial run profile: each of the $2n$ edges has a nonnegative integer flow value of at most $2^n + 1$. But in fact, a *terminating* run has at most $n2^n$ partial run profiles, as no *vertex-state pair* (v, s_curr) can repeat in Algorithm 1. In other words, a partial run profile $\boldsymbol{f}$ is determined by its end-vertex $v_{\boldsymbol{f}}$ as well as the positions of all switches at the time of the corresponding visit of $v_{\boldsymbol{f}}$. This means that a partial run profile can be encoded with $n + \log_2 n$ bits, and if we had a PLS or CLS membership proof of S-Arrival with circuits of this many inputs only, we could solve S-Arrival in time $O(\text{poly}(n) \times 2^{n/2})$.

Next, we show that such membership proofs indeed exist. For this, we show that the above encoding (of a partial run profile by an end-vertex and the positions of all switches) can be efficiently decoded: given an end-vertex and the positions of all switches, we can efficiently compute a unique candidate for a corresponding partial run profile. The resulting encoding and decoding circuits can be composed with the ones in Theorem 19 to obtain PLS and CLS membership proofs with circuits of $n + \log_2 n$ input bits, and hence yield a randomized algorithm of runtime $O(\text{poly}(n) \times 2^{n/2}) = O(1.4143^n)$, as explained above.

5.1 Decoding Partial Run Profiles

We work with instances of S-Arrival as in Definition 12, i.e., there is always a run profile either to d or to $\bar{d}$. This is without loss of generality [5, 14].

▶ **Definition 21.** Let $G = (V, E, s_0, s_1)$ be a switch graph, and let $o, d, \bar{d} \in V$ be as in Definition 12. The *parity* of a run profile $\boldsymbol{f} \in \mathbb{N}^{2n}$ is the vector $\boldsymbol{p_f} \in \{0,1\}^{n-2}$ defined by $\boldsymbol{p}_v = \boldsymbol{f}_{s_0(v)} - \boldsymbol{f}_{s_1(v)} \in \{0,1\}$, $v \in V \setminus \{d, \bar{d}\}$.

Note that we do not care about (the parity of) the switches at d and $\bar{d}$, since the algorithm stops as soon as d or $\bar{d}$ is reached.

Here is the main result of this section. For a given target vertex $t \in V$ and given parity $\boldsymbol{p}$, there is exactly one candidate for a partial run profile from o to t with parity $\boldsymbol{p}$. Moreover, this candidate can be computed by solving a system of linear equations.

▶ **Lemma 22.** *Let $(G, o, d, \bar{d})$ be an instance of* S-Arrival*, let $t \in V$ and $\boldsymbol{p} \in \{0,1\}^{n-2}$. Then there exists exactly one vector $\boldsymbol{f} \in \mathbb{R}^{2(n-2)}$ such that the following conditions hold.*
Kirchhoff's Law (flow conservation):

$$\forall v \in V \setminus \{d, \bar{d}\}\colon \sum_{e=(u,v)\in E} \boldsymbol{f}_e - \sum_{e=(v,w)\in E} \boldsymbol{f}_e = [v = t] - [v = o] \tag{1}$$

where $[\cdot]$ is the indicator variable of the event in brackets.
Parity Condition: $\boldsymbol{p_f} = \boldsymbol{p}$*, i.e.,*

$$\forall v \in V \setminus \{d, \bar{d}\}\colon \boldsymbol{f}_{s_0(v)} - \boldsymbol{f}_{s_1(v)} = \boldsymbol{p}_v. \tag{2}$$

Before we prove Lemma 22, let us draw a crucial conclusion: The unique partial run profile $\boldsymbol{f} \in \mathbb{N}^{2n}$ with end-vertex t and parity $\boldsymbol{p}$ (if such f exists – note that we are only guaranteed a real-valued $\boldsymbol{f}$) necessarily satisfies (1) and (2). Hence, we may use Lemma 22 to get the entries $\boldsymbol{f}_e$ for all edges except the ones leaving d and $\bar{d}$. Only if all the entries are nonnegative and integral and satisfy (1) at d and $\bar{d}$ (under $\boldsymbol{f}_{s_0(d)} = \boldsymbol{f}_{s_1(d)} = \boldsymbol{f}_{s_0(\bar{d})} = \boldsymbol{f}_{s_1(\bar{d})} = 0$) do we have a candidate for a partial run profile. Hence, there is a unique candidate, and given t and a $\boldsymbol{p}$, this candidate can be efficiently found.

Proof of Lemma 22. Set $m = 2(n-2)$, $V' = V \setminus \{d, \bar{d}\}$ and let $A \in \mathbb{Z}^{m \times m}$ be the coefficient matrix of the linear system (1), (2) in the variables $\boldsymbol{f}_e$. We show that A is invertible.

Let $\boldsymbol{q} \in \mathbb{R}^m$ be the vector such that $\boldsymbol{q}_{(v,s_i(v))} = -1$ if $s_i(v) = d$ and $\boldsymbol{q}_{(v,s_i(v))} = 0$ otherwise. We show that $\boldsymbol{q}$ can be expressed as a linear combination of the rows of A in a unique way, from which invertibility of A and the statement of the lemma follow.

Let us use coefficients λ_v for each $v \in V'$ for the rows corresponding to the flow conservation constraints (1), and coefficients μ_v for each $v \in V'$ for the rows corresponding to the parity constraints (2). The column of A corresponding to variable $\boldsymbol{f}_{(v,s_i(v))}$, has a -1 entry from the flow conservation constraint at v, and a 1 entry (if $i = 0$) or a -1 entry (if

$i = 1$) from the parity constraint at v. If $s_i(v) \neq d, \bar{d}$, there is another 1 entry from the flow conservation constraint at $s_i(v)$. All other entries are zero. The equations that express $\boldsymbol{q}$ as a linear combination of rows of A are therefore the following.

$$\begin{aligned}\forall v \in V' :\ -\lambda_v + \mu_v + \lambda_{s_0(v)} \cdot [s_0(v) \neq d, \bar{d}] &= \boldsymbol{q}_{(v,s_0(v))},\\ \forall v \in V' :\ -\lambda_v - \mu_v + \lambda_{s_1(v)} \cdot [s_1(v) \neq d, \bar{d}] &= \boldsymbol{q}_{(v,s_1(v))}.\end{aligned}$$

Or equivalently:

$$\lambda_v - \mu_v = \begin{cases} \lambda_{s_0(v)}, & \text{if } s_0(v) \neq d, \bar{d} \\ 1, & \text{if } s_0(v) = d \\ 0, & \text{if } s_0(v) = \bar{d} \end{cases} \qquad v \in V', \tag{3}$$

$$\lambda_v + \mu_v = \begin{cases} \lambda_{s_1(v)}, & \text{if } s_1(v) \neq d, \bar{d} \\ 1, & \text{if } s_1(v) = d \\ 0, & \text{if } s_1(v) = \bar{d} \end{cases} \qquad v \in V'. \tag{4}$$

We now show that there are unique coefficients λ_v, μ_v satisfying these equations. Let us define $\lambda_d = 1$ and $\lambda_{\bar{d}} = 0$. Adding corresponding equations of (3) and (4) then yields

$$\lambda_d = 1, \quad \lambda_{\bar{d}} = 0, \quad \lambda_v = \frac{1}{2}\left(\lambda_{s_0(v)} + \lambda_{s_1(v)}\right), \quad \forall v \in V'.$$

These are exactly the equations for the vertex values in a *stoppping simple stochastic game* on the graph G with only average or degree-1 vertices and sinks d and $\bar{d}$ (stopping means that d or $\bar{d}$ are reachable from everywhere which is exactly what we require in a switch graph). Condon proved that these values are unique [3]. This also determines the μ_v's uniquely. ◀

6 Conclusion and Open Problems

We showed that candidate run profiles in Arrival can be efficiently verified due to their structure. This allowed us to improve the known upper bounds for the search complexity of Arrival and S-Arrival. Here we mention some natural questions arising from our work.

- Are there any non-trivial graph properties that make Arrival or S-Arrival efficiently solvable? Given that we currently do not know of any polynomial time algorithm for Arrival on general switch graphs, we could study the complexity of Arrival on some interesting restricted classes of switch graphs.
- Are there other natural problems in $\mathsf{UP} \cap \mathsf{coUP}$ such that their corresponding search variant is reducible to EOML? Does End-Of-Metered-Line capture the computational complexity of any TFNP problem with unique solution? Fearnley et al. [7] recently gave a reduction from the PLCP to EOML. Given that Arrival and PLCP can be both reduced to EOML, yet another intriguing question is whether there exists any reduction between the two.
- As mentioned in Section 1.1, the reduction from PLCP to EOML by Fearnley et al. [7] implies that PLCP can be solved faster with Aldous' algorithm [1] than with any other known algorithm. It would be interesting to see whether Aldous' algorithm can similarly give improved runtimes for other problems than Arrival and PLCP.

References

1. David Aldous. Minimization algorithms and random walk on the d-cube. *The Annals of Probability*, 11(2):403–413, 1983.
2. Josh Buresh-Oppenheim and Tsuyoshi Morioka. Relativized NP search problems and propositional proof systems. In *19th Annual IEEE Conference on Computational Complexity (CCC 2004), 21-24 June 2004, Amherst, MA, USA*, pages 54–67, 2004.
3. Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.
4. Constantinos Daskalakis and Christos H. Papadimitriou. Continuous local search. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 790–804, 2011.
5. Jérôme Dohrau, Bernd Gärtner, Manuel Kohler, Jiří Matoušek, and Emo Welzl. ARRIVAL: A zero-player graph game in NP∩coNP. In Martin Loebl, Jaroslav Nešetřil, and Robin Thomas, editors, *A Journey Through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 367–374. Springer International Publishing, 2017.
6. John Fearnley, Martin Gairing, Matthias Mnich, and Rahul Savani. Reachability switching games. *CoRR*, abs/1709.08991, 2017. URL: `http://arxiv.org/abs/1709.08991`.
7. John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. CLS: new problems and completeness. *CoRR*, abs/1702.06017, 2017. URL: `http://arxiv.org/abs/1702.06017`.
8. Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubáček, Karel Král, Hagar Mosaad, and Veronika Slívová. ARRIVAL: next stop in CLS. *CoRR*, abs/1802.07702, 2018. `arXiv:1802.07702`.
9. Pavel Hubáček and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1352–1371, 2017.
10. David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.
11. Bastian Katz, Ignaz Rutter, and Gerhard J. Woeginger. An algorithmic study of switch graphs. *Acta Inf.*, 49(5):295–312, 2012.
12. Tsuyoshi Morioka. Classification of search problems and their definability in bounded arithmetic. *Electronic Colloquium on Computational Complexity (ECCC)*, 2001. URL: `https://eccc.weizmann.ac.il/eccc-reports/2001/TR01-082/index.html`.
13. Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.
14. Karthik C. S. Did the train reach its destination: The complexity of finding a witness. *Inf. Process. Lett.*, 121:17–21, 2017.

Improved Bounds for Shortest Paths in Dense Distance Graphs

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Adam Karczmarz[1]
University of Warsaw, Poland
a.karczmarz@mimuw.edu.pl

Abstract

We study the problem of computing shortest paths in so-called *dense distance graphs*, a basic building block for designing efficient planar graph algorithms. Let G be a plane graph with a distinguished set ∂G of *boundary vertices* lying on a constant number of faces of G. A distance clique of G is a complete graph on ∂G encoding all-pairs distances between these vertices. A dense distance graph is a union of possibly many unrelated distance cliques.

Fakcharoenphol and Rao [7] proposed an efficient implementation of Dijkstra's algorithm (later called *FR-Dijkstra*) computing single-source shortest paths in a dense distance graph. Their algorithm spends $O(b \log^2 n)$ time per distance clique with b vertices, even though a clique has b^2 edges. Here, n is the total number of vertices of the dense distance graph. The invention of FR-Dijkstra was instrumental in obtaining such results for planar graphs as nearly-linear time algorithms for multiple-source-multiple-sink maximum flow and dynamic distance oracles with sublinear update and query bounds.

At the heart of FR-Dijkstra lies a data structure updating distance labels and extracting minimum labeled vertices in $O(\log^2 n)$ amortized time per vertex. We show an improved data structure with $O\left(\frac{\log^2 n}{\log^2 \log n}\right)$ amortized bounds. This is the first improvement over the data structure of Fakcharoenphol and Rao in more than 15 years. It yields improved bounds for all problems on planar graphs, for which computing shortest paths in dense distance graphs is currently a bottleneck.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases shortest paths, dense distance graph, planar graph, Monge matrix

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.61

Related Version A full version of this paper is available at [11], `https://arxiv.org/abs/1602.07013`.

Acknowledgements We thank Piotr Sankowski for helpful discussions. The first author also thanks Oren Weimann and Shay Mozes for discussions about Monge matrices and FR-Dijkstra.

[1] Supported by the grants 2014/13/B/ST6/01811 and 2017/24/T/ST6/00036 of the Polish National Science Center.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 61; pp. 61:1–61:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Finding a truly subquadratic, strongly polynomial algorithm for many of the most basic real-weighted graph problems like the single-source shortest paths or the maximum flow on sparse digraphs seems to be very difficult. However, the situation changes significantly if we restrict ourselves to planar digraphs, which constitute an important class of sparse graphs. In this regime the ultimate goal is to obtain linear or almost linear time complexity.

In their breakthrough paper, Fakcharoenphol and Rao gave the first nearly-linear time algorithm for single-source shortest paths in real-weighted planar graphs [7]. Their algorithm had $O(n \log^3 n)$ time complexity. Although their upper bound was eventually improved to $O\left(n \frac{\log^2 n}{\log \log n}\right)$ by Mozes and Wulff-Nilsen [22], the techniques introduced in [7] proved very useful in obtaining not only nearly-linear time algorithms for other static planar graph problems, but also first sublinear dynamic algorithms for shortest paths and maximum flows.

A major contribution of Fakcharoenphol and Rao was introducing the general concept of a *dense distance graph*. Let G be a real-weighted plane digraph and let ∂G denote some subset of its vertices, called *boundary vertices*, such that there exist $\ell = O(1)$ faces $f_1, \ldots, f_\ell$ of G satisfying $\partial G \subseteq V(f_1) \cup \ldots V(f_\ell)$. Such graphs with a topologically nice boundary typically emerge after decomposing a plane graph using a cycle separator. For example, by using a cycle separator of Miller [19], one can decompose any n-vertex triangulated plane graph H into two subgraphs H_{in} and H_{out} such that (i) $H_{\text{in}} \cup H_{\text{out}} = H$, (ii) H_{in} and H_{out} are smaller than H by a constant factor, (iii) the set $\partial H_{\text{in}} = \partial H_{\text{out}} = V(H_{\text{in}}) \cap V(H_{\text{out}})$ has size $O(\sqrt{n})$ and lies both on a single face of H_{in} and on a single face of H_{out}.

We define a *distance clique* of G, denoted $\mathrm{DC}(G)$, to be a complete graph on ∂G such that the weight of an edge uv is equal to the length of the shortest path from u to v in G. A dense distance graph is a union of possibly many unrelated distance cliques.

We note that such a definition of a dense distance graph (also used in [23]) is a bit more general than that of Fakcharoenphol and Rao [7], who defined it only with respect to a recursive decomposition of G using cycle-separators. In fact, subsequently dense distance graphs have been also defined a bit differently with respect to so-called r-divisions [13], and even the two sides of a cycle-separator [15] (i.e., $\mathrm{DC}(H_{\text{in}}) \cup \mathrm{DC}(H_{\text{out}})$ in the above example). The definition we assume in this paper captures all these cases.

Suppose we are given q distance cliques $\mathrm{DC}(G_1), \ldots, \mathrm{DC}(G_q)$ explicitly. Let $\mathrm{DDG} = \bigcup_{i=1}^{q} \mathrm{DC}(G_i)$, $V = \partial G_1 \cup \ldots \cup \partial G_q$ and $n = |V|$. Clearly, DDG has $\sum_{i=1}^{q} |\partial G_i|^2$ edges in total. Fakcharoenphol and Rao showed how to compute single-source shortest paths in such graph DDG with non-negative edge weights in only $O\left(\sum_i |\partial G_i| \log^2 n\right)$ time, i.e., for each $\mathrm{DC}(G_i)$ one only needs to spend time nearly-linear in the number of *vertices* of $\mathrm{DC}(G_i)$, as opposed to its number of edges, i.e., $|\partial G_i|^2$. Their method is often called the *FR-Dijkstra*, as it follows the overall approach of Dijkstra's algorithm. Whereas Dijkstra's algorithm uses a priority queue to maintain its distance labels and extract a non-visited vertex with minimum label, a much more sophisticated data structure is used in FR-Dijkstra. This data structure is capable of relaxing many edges in a single step, by leveraging the fact that certain submatrices of the adjacency matrix of a distance clique are *Monge matrices*.

Applications of Dense Distance Graphs and FR-Dijkstra. Fakcharoenphol and Rao originally employed FR-Dijkstra to construct their dense distance graph recursively and answer distance queries on it. However, the applications of FR-Dijkstra proved much broader and thus it has become an important planar graph primitive used to obtain numerous breakthrough results in recent years. We briefly cover the most important of these results below.

The dense distance graphs and FR-Dijksta have been used to break the long-standing $O(n \log n)$ barrier for computing minimal s,t-cuts [12] in undirected planar graphs and global min-cuts in both undirected [17] and directed [20] planar graphs. Borradaile et al. [5] developed an oracle answering arbitrary min s,t-cut queries in an weighted undirected planar graph after only near-linear preprocessing. This result has been later generalized to bounded-genus graphs [3], thus proving the usefulness of FR-Dijkstra in more general graph classes.

The most sophisticated applications of FR-Dijkstra are probably those related to computing maximum flow in directed planar graphs. Borradaile et al. [4] gave a nearly-linear time max-flow algorithm for the case of multiple source and multiple sinks and maximum bipartite matching. Later, Łącki et al. [18] gave a nearly-linear time algorithm computing the maximum flow values between a specified source and all possible sinks.

Most recently, Asathulla et al. [2] used FR-Dijkstra to break through the $O(n^{3/2})$ barrier for minimum-cost bipartite weighted matching with integer weights. Cabello [6] showed the first truly subquadratic algorithm for computing a diameter of a weighted planar graph. Even though it mainly builds on a new concept of additively weighted Voronoi diagrams for planar graphs, dense distance graphs and FR-Dijkstra are still used extensively in his work. The diameter algorithm was later improved by Gawrychowski et al. [9] to run in $O(n^{5/3} \operatorname{polylog} n)$. Currently, [9] does not require FR-Dijkstra, but it seems that using it would be again required if one gave a more efficient Voronoi diagrams construction algorithm for planar graphs. Last but not least, FR-Dijkstra has been instrumental to obtaining virtually all *exact* dynamic algorithms for shortest paths, maximum flows and minimum cuts in planar graphs, with sublinear update/query bounds [7, 12, 13, 14, 17].

Significance. Dense distance graphs are pivotal in designing efficient planar graph algorithms, and therefore obtaining fine-grained bounds for computing and manipulating them is an important direction. Although a better algorithm (in comparison to the recursive method of [7]) running in $O((|V| + |\partial G|^2) \log n)$ time has been proposed for computing a distance clique [14], improving the FR-Dijkstra itself proved very challenging and no progress over [7] has been made so far in the most general setting that we study.

Related Work. For the important case of a *dense distance graph over an r-division*, i.e., when the individual graphs G_i are the pieces of an *r-division with few holes* of a single planar graph (see e.g., [16]), Mozes et al. [21] gave an algorithm for computing single source shortest paths in $O\left(\frac{n}{\sqrt{r}} \log^2 r\right)$ time. The original FR-Dijkstra runs in $O\left(\frac{n}{\sqrt{r}} \log n \log r\right)$ time in that case. Hence, [21] does not improve over it in the case of $r = \operatorname{poly} n$, which emerges in many important applications, e.g., [2, 3, 4, 13, 18]. However, dense distance graphs over r-divisions with $r = \operatorname{polylog}(n)$ have also found applications, most notably in $O(n \log \log n)$ algorithms for minimum cuts [12, 17, 20]. Computing shortest paths in dense distance graphs is not a bottleneck in those algorithms, though. For other applications of dense distance graphs over r-divisions with small r, consult [21].

Our Contribution. In this paper we show an algorithm for computing single-source shortest paths in a DDG in $O\left(\sum_{i=1}^{q} |\partial(G_i)| \frac{\log^2 n}{\log^2 \log n}\right)$ time, which is asymptotically faster than FR-Dijkstra in *all* cases. Specifically, for a dense distance graph defined over an r-division, the algorithm runs in $O\left(\frac{n}{\sqrt{r}} \frac{\log^2 n}{\log^2 \log n}\right)$ time.

We treat the problem of computing shortest paths in DDG from a purely data-structural perspective. At a high level, instead of developing an entirely new shortest paths algorithm,

we propose a new data structure for maintaining distance labels and extracting minimum labeled vertices in amortized $O\left(\frac{\log^2 n}{\log^2 \log n}\right)$ time, as opposed to $O(\log^2 n)$ time in [7].

In [7], a distance clique is first partitioned into *square* Monge matrices, each handling a subset of its edges. For any such matrix, a separate data structure is used for relaxing the corresponding edges and extracting the labels possibly induced by these edge relaxations. Recall that in the case of Dijkstra's algorithm, the improvement from $O(m \log n)$ to $O(m + n \log n)$ is obtained by noticing that relaxing edges is cheaper than extracting minimum labeled vertices. Consequently, one can use a Fibonacci heap [8] in place of a binary heap. We show that in the case of the data structure originally used in [7] for handling Monge matrices, the situation is in a sense the opposite: label extractions can be made cheaper than edge relaxations. We make use of this fact by proposing a biased scheme of partitioning distance cliques into *rectangular* (as opposed to square) Monge matrices, different than in [7]. Whereas in [7], the partition follows from a very natural idea of splitting face boundary into halves, our partition is tailored to exploit this asymmetry between the cost of processing a row and the cost of processing a column.

Our result implies an immediate improvement by a factor of $O(\log^2 \log n)$ in the time complexity for a number of planar digraph problems such as multiple-source multiple-sink maximum flows, maximum bipartite matching [4], single-source all-sinks maximum flows [18], for which the best known time bounds were $O(n \log^3 n)$, i.e., already nearly-linear. It also yields polylog-logarithmic speed-ups to both preprocessing and query/update algorithms of dynamic algorithms for shortest paths and max-flows [12, 13, 14]. More generally, we make polylog-logarithmic improvements to all previous results (such as [2]). for which the bottleneck of the best known algorithm is computing shortest paths in a dense distance graph. A more detailed discussion on the implications of our result and on how FR-Dijkstra is used in different algorithms for planar graphs can be found in the full version [11].

It should be noted that for small values of r, such as $r = \text{polylog}(n)$, our algorithm does not improve on [21] for the case of a dense distance graph over an r-division.

Model of Computation. We assume the standard word-RAM model with word size $\Omega(\log n)$. However, we stress that our algorithm works in the very general case of *real* edge lengths, i.e., we are only allowed to perform arithmetical operations on lengths and compare them.

Outline of the Paper. In Section 2 we introduce the matrix notation that we use and state some important properties of Monge matrices. In Section 3 we give an overview of our shortest paths algorithm and also discuss the main ideas behind the improved data structure for reporting column minima of a staircase Monge matrix in an online fashion.

In Sections 4, 5 and 6 we develop the increasingly more powerful data structures for reporting column minima in online Monge matrices. Each of these data structures is used in a black-box manner in the following section.

Due to space limitations, many technical details, most proofs and discussion of the applications can be found in the full version [11].

2 Monge Matrices and Their Minima

In this paper we define a *matrix* to be a partial function $\mathcal{M} : R \times C \to \mathbb{R}$, where R (called *rows*) and C (called *columns*) are some totally ordered finite sets. Set $R = \{r_1, \ldots, r_k\}$ and $C = \{c_1, \ldots, c_l\}$, where $r_1 \leq \ldots \leq r_k$ and $c_1 \leq \ldots \leq c_l$. If for $r_i, r_j \in R$ we have $r_i \leq r_j$, we

also say that r_i is (weakly) *above* r_j and r_j is (weakly) *below* r_i. Similarly, when c_i, c_j we have $c_i < c_j$, we say that c_i is *to the left* of c_j and c_j is *to the right* of c_i.

For some matrix $\mathcal{M}$ defined on rows R and columns C, for $r \in R$ and $c \in C$ we denote by $\mathcal{M}_{r,c}$ an *element* of $\mathcal{M}$. An element is the value of $\mathcal{M}$ on pair (r, c), if defined.

For $R' \subseteq R$ and $C' \subseteq C$ we define $\mathcal{M}(R', C')$ to be a *submatrix* of $\mathcal{M}$. $\mathcal{M}(R', C')$ is a partial function on $R' \times C'$ satisfying $\mathcal{M}(R', C')_{r,c} = \mathcal{M}_{r,c}$ for any $(r, c) \in R' \times C'$ such that $\mathcal{M}_{r,c}$ is defined. We sometimes abuse this notation by writing $\mathcal{M}(R', c')$ or $\mathcal{M}(r', C')$ when R' or C' are single-element, i.e., when $R' = \{r'\}$ or $C' = \{c'\}$.

The *minimum* of a matrix $\min\{\mathcal{M}\}$ is defined as the minimum value of the partial function $\mathcal{M}$. The *column minimum* of $\mathcal{M}$ in column c is defined as $\min\{\mathcal{M}(R, \{c\})\}$.

We call a matrix $\mathcal{M}$ *rectangular* if $\mathcal{M}_{r,c}$ is defined for every $r \in R$ and $c \in C$. A matrix is called *staircase* (*flipped staircase*) if $|R| = |C|$ and $\mathcal{M}_{r_i,c_j}$ is defined iff $i \le j$ ($i \ge j$ resp.).

Finally, a *subrectangle* of $\mathcal{M}$ is a rectangular matrix $\mathcal{M}(\{r_a, \ldots, r_b\}, \{c_x, \ldots, c_y\})$ for $1 \le a \le b \le k$, $1 \le x \le y \le l$. We define a *subrow* to be a subrectangle with a single row.

For a matrix $\mathcal{M}$ and a function $d : R \to \mathbb{R}$, define the *offset matrix* $\mathrm{off}(\mathcal{M}, d)$ to be a matrix $\mathcal{M}'$ such that for all r, c such that $\mathcal{M}_{r,c}$ is defined, we have $\mathcal{M}'_{r,c} = \mathcal{M}_{r,c} + d(r)$.

We say that a matrix $\mathcal{M}$ with rows R and columns C is a *Monge matrix*, if for each $r_1, r_2 \in R$, $r_1 \le r_2$ and $c_1, c_2 \in C$, $c_1 \le c_2$ such that all elements $\mathcal{M}_{r_1,c_1}, \mathcal{M}_{r_1,c_2}, \mathcal{M}_{r_2,c_1}, \mathcal{M}_{r_2,c_2}$ are defined, the *Monge property* holds, i.e., we have

$$\mathcal{M}_{r_2,c_1} + \mathcal{M}_{r_1,c_2} \le \mathcal{M}_{r_1,c_1} + \mathcal{M}_{r_2,c_2}.$$

▶ **Fact 1.** *Let $\mathcal{M}$ be a Monge matrix. For any $R' \subseteq R$ and $C' \subseteq C$, $\mathcal{M}(R', C')$ is also a Monge matrix.*

▶ **Fact 2.** *Let $\mathcal{M}$ be a rectangular Monge matrix. Assume that for some $c \in C$ and $r \in R$, $\mathcal{M}_{r,c}$ is a column minimum of c. Then, for each column c^- to the left of c, there exists a row $r^- \ge r$, such that $\mathcal{M}_{r^-,c^-}$ is a column minimum of c^-. Similarly, for each column c^+ to the right of c, there exists a row $r^+ \le r$, such that $\mathcal{M}_{r^+,c^+}$ is a column minimum of c^+.*

▶ **Fact 3.** *Let $\mathcal{M}$ be a Monge matrix and let $d : R \to \mathbb{R}$. Then $\mathrm{off}(\mathcal{M}, d)$ is also Monge.*

▶ **Fact 4.** *Let $\mathcal{M}$ be a rectangular Monge matrix and assume R is partitioned into disjoint blocks $\mathcal{R} = R_1, \ldots, R_a$ such that each R_i is a contiguous group of subsequent rows and each R_i is above R_{i+1}. Assume also that the set C is partitioned into blocks $\mathcal{C} = C_1, \ldots, C_b$ so that C_i is to the left of C_{i+1}. Then, a matrix $\mathcal{M}'$ with rows $\mathcal{R}$ and columns $\mathcal{C}$ defined as $\mathcal{M}'_{R_i,C_j} = \min\{\mathcal{M}(R_i, C_j)\}$, is also a Monge matrix.*

▶ **Fact 5.** *Let $\mathcal{M}$ be a rectangular Monge matrix. Let $r \in R$ and $C = \{c_1, \ldots, c_l\}$. The set of columns $C_r \in C$ having one of their column minima in row r is contiguous, that is either $C_r = \emptyset$ or $C_r = \{c_a, \ldots, c_b\}$ for some $1 \le a \le b \le l$.*

3 Shortest Paths in a Dense Distance Graph: an Overview

Recall that we are explicitly given q graphs $\mathrm{DC}(G_1), \ldots, \mathrm{DC}(G_q)$, such that each $\mathrm{DC}(G_i)$ is a complete digraph encoding the distances between boundary vertices ∂G_i of a plane digraph G_i. Additionally, we assume that ∂G_i is distributed into some $O(1)$ faces of G_i. We also assume that the distances between the boundary vertices of G_i are non-negative.

Let $\mathrm{DDG} = \mathrm{DC}(G_1) \cup \ldots \cup \mathrm{DC}(G_q)$, $V = \partial G_1 \cup \ldots \cup \partial G_q$ and $n = |V|$. Our goal is to find an efficient algorithm for computing single-source shortest paths in DDG.

As the graphs $\mathrm{DC}(G_i)$ are given explicitly, we can assume that we are allowed to preprocess each $\mathrm{DC}(G_i)$ once in time asymptotically no more than the time used to construct it, which is clearly $\Omega(|\partial G_i|^2)$. To the best of our knowledge, in all known applications this time is $\Theta((|V(G_i)| + |\partial G_i|^2) \log |V(G_i)|)$, which is the running time of Klein's algorithm [14]. After the preprocessing stage, we may need to handle multiple shortest-path queries.

In order to obtain the speedup over FR-Dijkstra we use a subtle combination of techniques. The single-source shortest paths in DDG are computed with an optimized implementation of Dijkstra's algorithm. Recall that Dijkstra's algorithm run from the source s grows a set S of *visited* vertices of the graph such that the lengths $d(v)$ of the shortest paths $s \to v$ for $v \in S$ are already known. Initially $S = \{s\}$ and we repeatedly choose a vertex $y \in V \setminus S$ such that the value (a distance estimate) $z(y) := \min_{x \in S}\{d(x) + \ell(x, y) : (x, y) \in E\}$ is the smallest. The vertex y is then added to S with $d(y) = z(y)$. The vertices $y \in V \setminus S$ are typically stored in a priority queue with keys $z(y)$, which allows to choose the best y efficiently.

Since the vertices of ∂G_i lie on $O(1)$ faces of a planar digraph G_i, we can exploit the fact that many of the shortest paths represented by $\mathrm{DC}(G_i)$ have to cross. Formally, this is captured by the following lemma. Denote by $\mathrm{DC}(G_i)[u, v]$ the weight of uv in $\mathrm{DC}(G_i)$.

▶ **Lemma 1** ([22]). *Each $DC(G_i)$ can be decomposed into $O(1)$ (possibly flipped) staircase Monge matrices D_i of at most $|\partial G_i|$ rows and columns. For each $u, v \in \partial G_i$ we have:*

- *for each $\mathcal{M} \in D_i$ such that $\mathcal{M}_{u,v}$ is defined, $\mathcal{M}_{u,v} \geq DC(G_i)[u, v]$.*
- *there exists $\mathcal{M} \in D_i$ such that $\mathcal{M}_{u,v}$ is defined and $\mathcal{M}_{u,v} = DC(G_i)[u, v]$.*

The decomposition can be computed in $O(|\partial G_i|)^2)$ time if ∂G_i is a subset of a single face of G_i and in $O((|V(G_i)| + |\partial G_i|^2) \log |V(G_i)|)$ time otherwise.

In other words, the adjacency matrix of $\mathrm{DC}(G_i)$ can be partitioned into a constant number of staircase Monge matrices. Consequently, a natural approach to maintaining the minimum distance estimate $z(y)$, for $y \notin S$, is to split the work needed to accomplish this task between the individual matrices $\mathcal{M} \in \bigcup_{i=1}^{q} D_i$ that encode the edges of DDG. Then, it is sufficient to design a data structure reporting the column minima of the offset matrix $\mathrm{off}(\mathcal{M}, d)$ in an online fashion. Specifically, the data structure has to handle *row activations* intermixed with extractions of the column minima in non-decreasing order. Once Dijkstra's algorithm establishes the distance $d(v)$ to some vertex v, the row of $\mathrm{off}(\mathcal{M}, d)$ corresponding to v is activated and becomes available to the data structure. This row contains values $d(v) + \ell(v, w)$, where $\ell(v, w)$ is, by Lemma 1, no less than the length of the edge vw in DDG. Alternatively, a minimum in some column corresponding to v (in the revealed part of $\mathrm{off}(\mathcal{M}, d)$) may be used by Dijkstra's algorithm to establish a new distance label $z(v) = d(v)$, even though not all rows of $\mathrm{off}(\mathcal{M}, d)$ have been revealed so far. In this case, we can guarantee that all the inactive rows of $\mathrm{off}(\mathcal{M}, d)$ contain entries not smaller than $d(v)$ and hence we can safely extract the column minimum of $\mathrm{off}(\mathcal{M}, d)$.

Such an approach was also used by Fakcharoenphol and Rao [7] and Mozes et al. [21], who both dealt with staircase Monge matrices by using a recursive partition into *square* Monge matrices, which are easier to handle. In particular, Fakcharoenphol and Rao showed that a sequence of row activations and column minima extractions can be performed on an $m \times m$ square Monge matrix in $O(m \log m)$ time. The recursive partition assigns each row and column to $O(\log |\partial G_i|)$ square Monge matrices. As a result, in [7], the total time for handling all the square matrices is $O(|\partial G_i| \log^2 |\partial G_i|)$. The details and the pseudocode of the above shortest path algorithm can be found in the full version [11].

The Data Structure. Developing an improved data structure reporting the column minima of an online offset staircase Monge matrix is the main contribution of this paper. This goal is achieved in three steps, presented in the following three sections in a bottom-up fashion. Below we sketch the main ideas behind these steps.

Our first component is a refined data structure for handling row activations and column minima extractions on a *rectangular* Monge matrix, described in Section 4. We show a data structure supporting any sequence of operations on a $k \times l$ matrix in $O\left(k\frac{\log m}{\log\log m} + l\log m\right)$ total time, where $m = \max(k, l)$. In comparison to [7], we do not map all the columns to active rows containing the current minima. Instead, the columns are assigned *potential row sets* of bounded size that are guaranteed to contain the "currently optimal" rows. This relaxed notion allows to remove the seemingly unavoidable binary search at the heart of [7] and instead use the SMAWK algorithm [1] to split the potential row sets once they become too large. The maintenance of a priority queue used for reporting the column minima in order is possible with the recent efficient data structure supporting subrow minimum queries in Monge matrices [10] and priority queues with $O(1)$ time Decrease-Key operation [8].

The second step is to relax the requirements posed on a data structure handling rectangular $k \times l$ Monge matrices. It is motivated by the following observation. Let $\Delta > 0$ be an integer. Imagine we have found the minima of l/Δ evenly spread, *pivot* columns $c_1, \ldots, c_{l/\Delta}$. Denote by $r_1, \ldots, r_{l/\Delta}$ some rows containing the corresponding minima. A well-known property of Monge matrices implies that for any column c' lying between c_i and c_{i+1}, we only have to look for a minimum of c' in rows $r_i, \ldots, r_{i+1}$. Thus, the minima in the remaining columns can be found in $O(k\Delta + l)$ total time. In Section 5 we show how to adapt this idea to an online setting that fits our needs. The columns are partitioned into $O(l/\Delta)$ *blocks* of size at most Δ. Each block is conceptually contracted to a single column: an entry in row r is defined as the minimum in row r over the contracted columns. For sufficiently small values of Δ, such a minimum can be computed in $O(1)$ time using the data structure of [10]. Locating a block minimum can be seen as an introduction of a new pivot column. We handle the block matrix with the data structure of Section 4 and prove that the total time needed to correctly report all the column minima is $O\left(k\frac{\log m}{\log\log m} + k\Delta + l + \frac{l}{\Delta}\log m\right)$. In particular, for $\Delta = \log^{1-\epsilon} m$, this bound becomes $O\left(k\frac{\log m}{\log\log m} + l\log^{\epsilon} m\right)$.

Finally, in Section 6 we exploit the asymmetry of per-row and per-column costs of the developed block data structure for rectangular matrices by using a different partition of a staircase Monge matrix. Our partition is biased towards columns, i.e., the matrix is split into *rectangular* (as opposed to square) Monge matrices, each with roughly poly-logarithmically more columns than rows. Consequently, the total number of rows in these matrices is $O\left(|\partial G_i|\frac{\log|\partial G_i|}{\log\log|\partial G_i|}\right)$, whereas the total number of columns is only slightly larger, i.e., $O\left(|\partial G_i|\log^{1+\epsilon}|\partial G_i|\right)$. This yields a data structure handling staircase Monge matrices in $O\left(|\partial G_i|\frac{\log^2|\partial G_i|}{\log^2\log|\partial G_i|}\right)$ total time. By plugging this data structure into our shortest path algorithm, we obtain the following theorem.

▶ **Theorem 2.** *The single-source shortest paths computations in DDG can be performed in* $O\left(\sum_{i=1}^{q}|\partial G_i|\frac{\log^2 n}{\log^2\log n}\right)$ *time. The required preprocessing time per each* G_i *is* $O(|\partial G_i|^2)$ *if* ∂G_i *lies on a single face of* G_i*, and* $O\left(|V(G_i)| + |\partial G_i|^2)\log|V(G_i)|\right)$ *otherwise.*

4 Online Column Minima of a Rectangular Offset Monge Matrix

Let $\mathcal{M}_0$ be a rectangular $k \times l$ Monge matrix. Let $R = \{r_1, \ldots, r_k\}$ and $C = \{c_1, \ldots, c_l\}$ be the sets of rows and columns of $\mathcal{M}_0$, respectively. Set $m = \max(k, l)$.

Let $d : R \to \mathbb{R}$ be an offset function and set $\mathcal{M} = \text{off}(\mathcal{M}_0, d)$. By Fact 3, $\mathcal{M}$ is also a Monge matrix. Our goal is to design a data structure capable of reporting the column minima of $\mathcal{M}$ in increasing order of their values. However, the function d is not entirely revealed beforehand, as opposed to the matrix $\mathcal{M}_0$. There is an initially empty, growing set $\overline{R} \subseteq R$ containing the rows for which $d(r)$ is known. Alternatively, the set $\overline{R}$ can be seen as "active" rows of $\mathcal{M}$ that can be accessed by the data structure. There is also a set $\overline{C} \subseteq C$ containing the remaining columns for which we have not reported the minima yet. Initially, $\overline{C} = C$ and $\overline{C}$ shrinks over time. We also provide a mechanism to guarantee that the rows that have not been revealed do not influence the smallest of the column minima of $\mathcal{M}(R, \overline{C})$.

The exact set of operations we support is the following:

- ACTIVATE-ROW(r), where $r \in R \setminus \overline{R}$ – add r to the set $\overline{R}$.
- LOWER-BOUND() – compute the number $\min\{\mathcal{M}(\overline{R}, \overline{C})\}$.
- ENSURE-BOUND-AND-GET() – inform the data structure that we indeed have $\min\{\mathcal{M}(R \setminus \overline{R}, C)\} \geq \min\{\mathcal{M}(\overline{R}, \overline{C})\} = $ LOWER-BOUND(), that is, the smallest element of $\mathcal{M}(R, \overline{C})$ does not depend on the values of $\mathcal{M}$ located in rows $R \setminus \overline{R}$.
  Observe that such claim implies that for some column $c \in \overline{C}$ we have $\min\{\mathcal{M}(R, c)\} = \min\{\mathcal{M}(\overline{R}, \overline{C})\}$, which in turn means that we are able to find the minimum element in column c. The function returns any such c and removes it from the set $\overline{C}$.
- CURRENT-MIN-ROW(c), where $c \in C$ – compute r, where $r \in \overline{R}$ is a row such that $\min\{\mathcal{M}(\overline{R}, c)\} = \mathcal{M}_{r,c}$. If $\overline{R} = \emptyset$, return **nil**. Note that c is not necessarily in $\overline{C}$.
  Additionally, we require CURRENT-MIN-ROW to have the following property: once the column c is moved out of $\overline{C}$, CURRENT-MIN-ROW(c) always returns the same row. Moreover, for $c_1, c_2 \in C$, $c_1 < c_2$, we have CURRENT-MIN-ROW$(c_1) \geq$ CURRENT-MIN-ROW(c_2).

Note that ACTIVATE-ROW increases the size of $\overline{R}$ and thus cannot be called more than k times. Analogously, ENSURE-BOUND-AND-GET decreases the size of $\overline{C}$ so it cannot be called more than l times. Actually, in order to reveal all the column minima with this data structure, the operation ENSURE-BOUND-AND-GET has to be called *exactly* l times.

4.1 The Components

The Subrow Minimum Query Data Structure. Given $r \in \overline{R}$ and a, b, $1 \leq a \leq b \leq l$, a subrow minimum query $S(r, a, b)$ computes a column $c \in \{c_a, \ldots, c_b\}$ such that $\mathcal{M}_{r,c} = \min\{\mathcal{M}(r, \{c_a, \ldots, c_b\})\}$. We use the following theorem of Gawrychowski et al. [10].

▶ **Theorem 3** ([10]). *Given a $k \times l$ rectangular Monge matrix $\mathcal{M}$, a data structure supporting subrow minimum queries in $O(\log\log(k + l))$ time can be constructed in $O(l \log k)$ time.*

Recall that $\mathcal{M} = \text{off}(\mathcal{M}_0, d)$. Adding the offset $d(r)$ to all the elements in row r of $\mathcal{M}_0$ does not change the relative order of elements in row r. Hence, the answer to a subrow minimum query $S(r, a, b)$ in $\mathcal{M}$ is the same as the answer to $S(r, a, b)$ in $\mathcal{M}_0$.

Therefor, by building a data structure of Theorem 3 for $\mathcal{M}_0$ we can answer any subrow minimum query in $\mathcal{M}$ in $O(\log\log m)$ time.

The Column Groups. The set C is internally partitioned into disjoint, contiguous *column groups* $\mathcal{C}_1, \ldots, \mathcal{C}_q$ (where $\mathcal{C}_1$ is the leftmost and $\mathcal{C}_q$ is the rightmost), so that $\bigcup_i \mathcal{C}_i = C$. For each $c \notin \overline{C}$, there is a group consisting of a single element c. Such a group is called *done*.

As the groups constitute contiguous segments of columns, we can represent the partition with a subset $F \subseteq C$ containing the first columns of individual groups. Each group is identified with its leftmost column. We use a dynamic predecessor data structure [24] for maintaining the set F. Such representation also allows to split groups and merge neighboring groups in $O(\log \log m)$ time.

The Potential Row Sets. For each $\mathcal{C}_i$ we store a set $P(\mathcal{C}_i) \subseteq \overline{R}$, called a *potential row set*. Between consecutive operations, the potential row sets satisfy the following invariants:

P.1 For any $c \in \mathcal{C}_i$ there exists a row $r \in P(\mathcal{C}_i)$ such that $\min\{\mathcal{M}(\overline{R}, c)\} = \mathcal{M}_{r,c}$.

P.2 The size of any set $P(\mathcal{C}_i)$ is less than 2α, where $\alpha = \sqrt{\log m}$.

P.3 For any $i < j$ and any $r \in P(\mathcal{C}_i)$, $r' \in P(\mathcal{C}_j)$, we have $r \geq r'$.

The sets $P(\mathcal{C}_i)$ are stored as balanced binary search trees, sorted bottom to top. Intuitively, invariant 3 can be maintained because, by Fact 1, $\mathcal{M}(\overline{R}, C)$ is a Monge matrix, so Fact 2 applies. Then, we have $|P(\mathcal{C}_i) \cap P(\mathcal{C}_{i+1})| \leq 1$, so the sum of sizes of sets $P(\mathcal{C}_i)$ is $O(k + l)$.

▶ **Lemma 4.** *An insertion or deletion of some r to $P(\mathcal{C}_i)$ (along with the update of the auxiliary structures) can be performed in $O(\log \alpha + \log \log m)$ time.*

Clearly, one can answer the Current-Min-Row(c) query by finding the relevant group $\mathcal{C}_i$, $c \in \mathcal{C}_i$, and examining the entries $\mathcal{M}_{r,c}$ for $r \in P(\mathcal{C}_i)$. This takes $O(\log \log m + \alpha)$ time.

Upon activation of a new row r, we first merge the groups $\mathcal{C}_j$ such that r contains a current minimum for each column in $\mathcal{C}_j$. The potential row set of the newly formed group is set to $\{r\}$. Next, we insert r to some (at most two) of the existing potential row sets. This might make some $P(\mathcal{C}_i)$ break invariant 2. In such case the group $\mathcal{C}_i$ along with $P(\mathcal{C}_i)$ is split, so that the resulting potential row sets are of size α. The splitting algorithm summarized by the following lemma which leverages the SMAWK algorithm [1] to decrease the per-row cost of a split.

▶ **Lemma 5.** *Let $\mathcal{M}$ be a $u \times v$ rectangular Monge matrix with rows $\{r_1, \ldots, r_u\}$ and columns $C = \{c_1, \ldots, c_v\}$. For any $i \in [1, u]$, in $O\left(u \frac{\log v}{\log u}\right)$ time we can find such $c_s \in C$ that:*

1. *Some minima of columns $c_1, \ldots, c_s$ lie in rows $r_{i+1}, \ldots, r_u$.*
2. *Some minima of columns $c_{s+1}, \ldots, c_v$ lie in rows $r_1, \ldots, r_i$.*

As the split of some fixed $P(\mathcal{C}_i)$ happens at most once per α insertions, we charge the $O\left(\alpha \frac{\log m}{\log \alpha}\right)$ cost of splitting $P(\mathcal{C}_i)$ to the α elements inserted since the last split of $P(\mathcal{C}_i)$. The total number of insertions performed on the potential row sets is $O(k + l)$.

The Priority Queue. A priority queue H contains an element c for each $c \in \overline{C}$. The queue H satisfies the following invariants between any two operations.

H.1 For each $c \in \overline{C}$, the key of c in H is greater than or equal to $\min\{\mathcal{M}(\overline{R}, c)\}$.

H.2 For each group $\mathcal{C}_j$ that is not done, there exists such column $c_j \in \mathcal{C}_j$ that the key of c_j in H is equal to $\min\{\mathcal{M}(\overline{R}, c_j)\} = \min\left\{\mathcal{M}(\overline{R}, \mathcal{C}_j)\right\}$.

Note that by invariants 1 and 2, the key at the top of H is in fact equal to $\min\{\mathcal{M}(\overline{R}, \overline{C})\}$. Hence, Lower-Bound can be implemented trivially in $O(1)$ time. The following lemma follows easily from invariant 1 and Theorem 3.

▶ **Lemma 6.** *We can ensure 2 is satisfied for a single group $\mathcal{C}_j$ in $O(\alpha \log \log m)$ time.*

The detailed description of how the individual operations are implemented can be found in the full version [11]. The performance of our data structure can be summarized as follows.

▶ **Lemma 7.** *Let $\mathcal{M}$ be a $k \times l$ offset Monge matrix. There exists a data structure built in $O(k + l \log m)$ time, supporting* Lower-Bound *in $O(1)$ time and both* Current-Min-Row *and* Ensure-Bound-And-Get *in $O(\log m)$ time. Additionally, any sequence of operations* Activate-Row *is performed in $O\left((k+l)\frac{\log m}{\log\log m}\right)$ total time, where $m = \max(k, l)$.*

5 Online Column Minima of a Block Monge Matrix

Let $\mathcal{M} = \text{off}(\mathcal{M}_0, d)$, R, C, l, k, m be defined as in Section 4. In this section we consider the problem of reporting the column minima of a rectangular offset Monge matrix, but in a slightly different setting. Again, we are given a fixed rectangular Monge matrix $\mathcal{M}_0$ and we also have an initially empty, growing set of rows $\overline{R} \subseteq R$ for which the offsets $d(\cdot)$ are known. Let $\Delta > 0$ be an integral parameter not larger than l. We partition C into a set $\mathcal{B} = \{B_1, \ldots, B_b\}$ of at most $\lceil l/\Delta \rceil$ blocks, each of size at most Δ. The columns in each B_i constitute a contiguous fragment of $c_1, \ldots, c_l$, and each block B_i is to the left of B_{i+1}. We also maintain a shrinking subset $\overline{\mathcal{B}} \subseteq \mathcal{B}$ containing the blocks B_i, such that the minima $\min\{\mathcal{M}(R, B_i)\}$ are not yet known. More formally, for each $B_i \in \mathcal{B} \setminus \overline{\mathcal{B}}$, we have $\min\{\mathcal{M}(R, B_i)\} = \min\{\mathcal{M}(\overline{R}, B_i)\}$. Initially $\overline{\mathcal{B}} = \mathcal{B}$.

For each $c \in C$ not contained in the blocks of $\overline{\mathcal{B}}$, the data structure explicitly maintains the *current minimum*, i.e., the value $\min\{\mathcal{M}(\overline{R}, c)\}$. Moreover, when a new row is activated, we provide the user with columns of $\bigcup(\mathcal{B} \setminus \overline{\mathcal{B}})$ for which the current minima have changed.

For blocks $\overline{\mathcal{B}}$, the data structure only maintains the value $\min\{\mathcal{M}(\overline{R}, \bigcup\overline{\mathcal{B}})\}$. Once the user can guarantee that $\min\{\mathcal{M}(R, \bigcup\overline{\mathcal{B}})\}$ does not depend on the "hidden offsets" of rows $R \setminus \overline{R}$, the data structure moves a block $B_i \in \overline{\mathcal{B}}$ such that $\min\{\mathcal{M}(R, \bigcup\overline{\mathcal{B}})\} = \min\{\mathcal{M}(\overline{R}, B_i)\}$ out of $\overline{\mathcal{B}}$ and makes it possible to access the current minima in the columns of B_i.

More formally, we support the following set of operations:

- Activate-Row(r), where $r \in R \setminus \overline{R}$ – add r to the set $\overline{R}$.
- Block-Lower-Bound() – return $\min\{\mathcal{M}(\overline{R}, \bigcup\overline{\mathcal{B}})\}$.
- Block-Ensure-Bound() – tell the data structure that indeed $\min\{\mathcal{M}(R \setminus \overline{R}, C)\} \geq$ Block-Lower-Bound() $= \min\{\mathcal{M}(\overline{R}, B_i)\}$, for some $B_i \in \overline{\mathcal{B}}$, i.e., the smallest element of $\mathcal{M}(R, \bigcup\overline{\mathcal{B}})$ does not depend on the entries of $\mathcal{M}$ located in rows $R \setminus \overline{R}$.
  As the minimum of $\mathcal{M}(R, B_i)$ can now be computed, B_i is removed from $\overline{\mathcal{B}}$.
- Current-Min(c), where $c \in C$ – for $c \in \bigcup(\mathcal{B} \setminus \overline{\mathcal{B}})$, return the explicitly maintained value $\min\{\mathcal{M}(\overline{R}, \{c\})\}$. For $c \in \bigcup\overline{\mathcal{B}}$, set Current-Min$(c) = \infty$.

Additionally, the data structure provides access to the queue Updates containing the columns $c \in \bigcup(\mathcal{B} \setminus \overline{\mathcal{B}})$ such that the most recent call to either Activate-Row or Block-Ensure-Bound resulted in a change (or an initialization, if $c \in B_i$ and the last update was Block-Ensure-Bound, which moved B_i out of $\overline{\mathcal{B}}$) of the value Current-Min(c).

Note that there can be at most k calls to Activate-Row and no more than $\lceil l/\Delta \rceil$ calls to Block-Ensure-Bound.

5.1 The Data Structure

An Infrastructure for Short Subrow Minimum Queries. In this section we assume that for any $r \in R$ and $1 \leq i, j \leq l$, $j - i + 1 \leq \Delta$, it is possible to compute an answer to a subrow minimum query $S(r, i, j)$ (see Section 4) on matrix $\mathcal{M}_0$ (equivalently: $\mathcal{M}$) in constant time. We call such a subrow minimum query *short*.

The Block Minima Matrix. Define a $k \times b$ matrix $\mathcal{M}'$ with rows R and columns $\mathcal{B}$, such that for all $r_i \in R$ and $B_j \in \mathcal{B}$, $\mathcal{M}'_{r_i,B_j} = \min\{\mathcal{M}(r_i, B_j)\}$. We build the data structure of Section 4 for matrix $\mathcal{M}'$.

▶ **Lemma 8.** *$\mathcal{M}'$ is a Monge matrix and its entries can be accessed in $O(1)$ time.*

The Exact Minima Array. For each column $c \in \bigcup(\mathcal{B} \setminus \overline{\mathcal{B}})$, the value $\texttt{cmin}(c) = \min\{\mathcal{M}(\overline{R}, c)\}$ is stored explicitly. The operation Current-Min(c) returns $\texttt{cmin}(c)$.

Rows Containing the Block Minima. For each $B_j \in (\mathcal{B} \setminus \overline{\mathcal{B}})$ we store the value $y_j = \mathcal{M}'$.Current-Min-Row(B_j). Note that the data structure of Section 4 guarantees that for $B_i, B_j \in (\mathcal{B} \setminus \overline{\mathcal{B}})$ such that $i < j$, we have $y_i \geq y_j$. The set of defined y_j's grows over time.

The Row Candidate Sets. Two subsets D_0 and D_1 of $\overline{R}$ are maintained. The set D_q for $q = 0, 1$ contains the rows of $\overline{R}$ that may still prove useful when computing the initial value of $\texttt{cmin}(c)$ for $c \in \bigcup\{B_i : B_i \in \overline{\mathcal{B}} \wedge i \bmod 2 = q\}$. For each such c, D_q contains a row r such that $\min\{\mathcal{M}(\overline{R}, c)\} = \mathcal{M}_{r,c}$. The sets D_q are also stored in dynamic predecessor structures.

Implementation. We now sketch how the data structure's components are used. Clearly, a call to Block-Lower-Bound translates into a single call Lower-Bound executed on $\mathcal{M}'$. When Block-Ensure-Bound is called, some block B_i is moved out of $\overline{\mathcal{B}}$. Apart from calling Ensure-Bound-And-Get on $\mathcal{M}'$, we have to initialize the values $\texttt{cmin}(c)$ for $c \in B_i$. For each such c, we examine multiple rows of $D_{i \bmod 2}$ when looking for minima, but it can be shown that most of these rows can be discarded from $D_{i \bmod 2}$ afterwards. This in turn allows us to charge the work to the insertions into row candidate sets.

When Activate-Row is called, one has to call Activate-Row on $\mathcal{M}'$ first. Moreover, the values $\texttt{cmin}(c)$ for some $c \in \bigcup(\mathcal{B} \setminus \overline{\mathcal{B}})$ have to be updated. It turns out that all such columns reside in at most two blocks and thus only $O(\Delta)$ additional time is needed.

The detailed implementation of each operation can be found in the full version [11].

▶ **Lemma 9.** *Let $\mathcal{M} = \mathit{off}(\mathcal{M}_0, d)$ be a $k \times l$ rectangular offset Monge matrix. Let Δ be the block size. Assume we can perform subrow minima queries spanning at most Δ columns of $\mathcal{M}_0$ in $O(1)$ time. There exists a data structure initialized in $O(k + l + \frac{l}{\Delta} \log m)$ time and supporting both* Block-Lower-Bound *and* Current-Min *in $O(1)$ time. Any sequence of* Activate-Row *and* Block-Ensure-Bound *operations can be performed in $O\left(k\left(\frac{\log m}{\log\log m} + \Delta\right) + l + \frac{l}{\Delta}\log m\right)$ time, where $m = \max(k, l)$.*

6 Online Column Minima of a Staircase Offset Monge Matrix

In this section we show a data structure supporting a similar set of operations as in Section 4, but in the case when the matrices $\mathcal{M}_0$ and $\mathcal{M} = \mathrm{off}(\mathcal{M}_0, d)$ are staircase Monge matrices with m rows $R = \{r_1, \ldots, r_m\}$ and m columns $C = \{c_1, \ldots, c_m\}$. We still aim at reporting the column minima of $\mathcal{M}$, while the set $\overline{R}$ of revealed rows is extended and new bounds on $\min\{\mathcal{M}(R \setminus \overline{R}, C)\}$ are available.

In comparison to the data structure of Section 4, we loosen the conditions posed on the operations Lower-Bound and Ensure-Bound-And-Get. Now, Lower-Bound might return a value smaller than $\min\{\mathcal{M}(\overline{R}, \overline{C})\}$ and a single call to Ensure-Bound-And-Get might not report any new column minimum at all. However, Ensure-Bound-And-Get can

still only be called if $\min\{\mathcal{M}(R \setminus \overline{R}, C)\} \geq$ Lower-Bound() and the data structure we develop in this section guarantees that a bounded number of calls to Ensure-Bound-And-Get suffices to report all the column minima of $\mathcal{M}$. The exact set of supported operations is:

- Activate-Row(r), where $r \in R \setminus \overline{R}$ – add r to the set $\overline{R}$.
- Lower-Bound() – return a number v such that $\min\{\mathcal{M}(\overline{R}, \overline{C})\} \geq v$.
- Ensure-Bound-And-Get() – tell the data structure that the inequality $\min\{\mathcal{M}(R \setminus \overline{R}, C)\} \geq$ Lower-Bound() holds.
  With this knowledge, the data structure may report some column $c \in \overline{C}$ such that $\min\{\mathcal{M}(R, c)\}$ is known. However, it's also valid to not report any new column minimum (in such case **nil** is returned) and only change the known value of Lower-Bound().
- Current-Min(c), where $c \in C$ – if $c \in C \setminus \overline{C}$, return the known minimum in column c.

6.1 The Data Structure

The Short Subrow Minimum Queries Infrastructure. Let $\Delta = \lceil \log^{1-\epsilon/2} m \rceil$. The following lemma allows us to use the data structure of Lemma 9 with block size Δ.

▶ **Lemma 10.** *The staircase Monge matrix $\mathcal{M}_0$ can be preprocessed in $O(m\Delta \log m)$ time so that subrow minimum queries on $\mathcal{M}_0$ spanning at most Δ columns take $O(1)$ time.*

The Partition of $\mathcal{M}$ into Rectangular Matrices $\mathcal{M}_1, \ldots, \mathcal{M}_q$. We partition the staircase Monge matrix $\mathcal{M}$ into $O(m \log^{\epsilon/2} m)$ non-overlapping *rectangular* Monge matrices $\mathcal{M}_1, \ldots, \mathcal{M}_q$ using the below lemma. Each $\mathcal{M}_i$ is a subrectangle of $\mathcal{M}$, each row r (column c) appears in a set W_r (W^c, resp.) of $O\left(\frac{\log m}{\log \log m}\right)$ ($O\left(\frac{\log^{1+\epsilon/2} m}{\log \log m}\right)$, resp.) subrectangles.

▶ **Lemma 11.** *For any $\epsilon \in (0, 1)$, a staircase matrix $\mathcal{M}$ with m rows and m columns can be partitioned in $O(m)$ time into $O(m \log^{\epsilon/2} m)$ non-overlapping rectangular matrices so that each row appears in $O\left(\frac{\log m}{\log \log m}\right)$ matrices of the partition, whereas each column appears in $O\left(\frac{\log^{1+\epsilon} m}{\log \log m}\right)$ matrices of the partition.*

We build the block data structure of Section 5 for each $\mathcal{M}_i$. For each $\mathcal{M}_i$ we use the same block size Δ. As each $\mathcal{M}_i$ is a subrectangle of $\mathcal{M}$, Lemma 10 guarantees that we can perform subrow minimum queries on $\mathcal{M}_i$ spanning at most Δ columns in $O(1)$ time. Recall that the blocks of the matrix $\mathcal{M}_i$ are partitioned into two sets $\overline{\mathcal{B}}_i$ and $\mathcal{B}_i \setminus \overline{\mathcal{B}}_i$. Denote by $block(\mathcal{M}_i)$ the submatrix $\mathcal{M}_i(\overline{R}_i, \bigcup \overline{\mathcal{B}}_i)$ and by $exact(\mathcal{M}_i)$ the submatrix $\mathcal{M}_i(\overline{R}_i, \bigcup(\mathcal{B}_i \setminus \overline{\mathcal{B}}_i))$. Here, R_i and C_i denote the row and column sets of $\mathcal{M}_i$, respectively.

The Priority Queue H. The core of our data structure is a priority queue H. At any time, H contains an element c for each $c \in \overline{C}$ and at most one element $\mathcal{M}_i$ for each matrix $\mathcal{M}_i$. Thus the size of H never exceeds $O(m \log^{\epsilon/2} m)$. We maintain the following invariants after the initialization and each call Activate-Row or Ensure-Bound-And-Get resulting in $\overline{C} \neq \emptyset$:

H.1 For each $c \in \overline{C}$, the key of c in H is equal to $\min\{\mathcal{M}_i.\text{Current-Min}(c) : \mathcal{M}_i \in W^c\}$.

H.2 For each $\mathcal{M}_i$ such that $block(\mathcal{M}_i)$ is not empty, the key of $\mathcal{M}_i$ in H is equal to $\min\{block(\mathcal{M}_i)\} = \mathcal{M}_i.\text{Block-Lower-Bound}()$.

▶ **Lemma 12.** *Assume invariants 1 and 2 are satisfied. Then H.Min-Key() $\leq \mathcal{M}(\overline{R}, \overline{C})$.*

Implementation. The data structure always returns the top key of H when Lower-Bound is called. Each call to Ensure-Bound-And-Get removes the top element e of H. If e is a column c, the minimum of c is reported. Otherwise $e = \mathcal{M}_i$ and it can be seen that $\mathcal{M}_i$.Block-Ensure-Bound can now be called. Afterwards, $\mathcal{M}_i$ is reinserted into H. Note that the number of times some $\mathcal{M}_j$ gets reinserted into H is no more than the total number of blocks in the matrices $\mathcal{M}_1, \ldots, \mathcal{M}_q$, i.e., $O(m \log^{\epsilon} m)$. The row activations are propagated to the relevant matrices $\mathcal{M}_j$ of the partition.

Both Ensure-Bound-And-Get and Activate-Row may make invariants 1 and 2 violated. However, the keys in H may only need to be decreased. Given that each operation Decrease-Key on H takes only $O(1)$ time, the time needed to update H is dominated by the cost of operations on the individual matrices $\mathcal{M}_j$.

The detailed implementation of the individual operations and analysis can be found in the full version [11]. The following lemma summarizes the performance of our data structure.

▶ **Lemma 13.** *Let $\mathcal{M} = \mathit{off}(\mathcal{M}_0, d)$ be an $m \times m$ offset staircase Monge matrix and let $\epsilon \in (0, 1)$. There exists a data structure that can be initialized in $O\left(m \log^{2-\epsilon} m\right)$ time, supporting both* Lower-Bound *and* Current-Min *in $O(1)$ time. Any sequence of* Activate-Row *and* Ensure-Bound-And-Get *operations takes $O\left(m \frac{\log^2 m}{\log^2 \log m}\right)$ total time. All the column minima are computed after $O(m \log^{\epsilon} m)$ calls to* Ensure-Bound-And-Get.

References

1. Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359.
2. Mudabir Kabir Asathulla, Sanjeev Khanna, Nathaniel Lahn, and Sharath Raghvendra. A faster algorithm for minimum-cost bipartite perfect matching in planar graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 457–476, 2018. doi:10.1137/1.9781611975031.31.
3. Glencora Borradaile, David Eppstein, Amir Nayyeri, and Christian Wulff-Nilsen. All-pairs minimum cuts in near-linear time for surface-embedded graphs. In *32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016, Boston, MA, USA*, pages 22:1–22:16, 2016. doi:10.4230/LIPIcs.SoCG.2016.22.
4. Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM J. Comput.*, 46(4):1280–1303, 2017. doi:10.1137/15M1042929.
5. Glencora Borradaile, Piotr Sankowski, and Christian Wulff-Nilsen. Min *st*-cut oracle for planar graphs with near-linear preprocessing time. *ACM Trans. Algorithms*, 11(3):16:1–16:29, 2015. doi:10.1145/2684068.
6. Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2143–2152, 2017. doi:10.1137/1.9781611974782.139.
7. Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72(5):868–889, 2006. doi:10.1016/j.jcss.2005.05.007.
8. Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

9 Pawel Gawrychowski, Haim Kaplan, Shay Mozes, Micha Sharir, and Oren Weimann. Voronoi diagrams on planar graphs, and computing the diameter in deterministic $\tilde{O}(n^{5/3})$ time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 495–514, 2018. doi:10.1137/1.9781611975031.33.

10 Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Submatrix maximum queries in monge matrices are equivalent to predecessor search. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 580–592, 2015. doi:10.1007/978-3-662-47672-7_47.

11 Paweł Gawrychowski and Adam Karczmarz. Improved bounds for shortest paths in dense distance graphs, 2016. arXiv:1602.07013.

12 Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 313–322, 2011. doi:10.1145/1993636.1993679.

13 Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in monge matrices and partial monge matrices, and their applications. *ACM Trans. Algorithms*, 13(2):26:1–26:42, 2017. doi:10.1145/3039873.

14 Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 146–155, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070454.

15 Philip N. Klein and Shay Mozes. Optimization algorithms for planar graphs, 2017. URL: http://planarity.org.

16 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 505–514, 2013. doi:10.1145/2488608.2488672.

17 Jakub Łącki, and Piotr Sankowski and. Min-cuts and shortest cycles in planar graphs in o(n loglogn) time. In *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, pages 155–166, 2011. doi:10.1007/978-3-642-23719-5_14.

18 Jakub Łącki, Yahav Nussbaum, Piotr Sankowski and Christian Wulff-Nilsen. Single source - all sinks max flows in planar digraphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 599–608, 2012. doi:10.1109/FOCS.2012.66.

19 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986. doi:10.1016/0022-0000(86)90030-9.

20 Shay Mozes, Kirill Nikolaev, Yahav Nussbaum, and Oren Weimann. Minimum cut of directed planar graphs in $O(n \log \log n)$ time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 477–494, 2018. doi:10.1137/1.9781611975031.32.

21 Shay Mozes, Yahav Nussbaum, and Oren Weimann. Faster shortest paths in dense distance graphs, with applications. *Theor. Comput. Sci.*, 711:11–35, 2018. doi:10.1016/j.tcs.2017.10.034.

22 Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n \log^2 n / \log \log n)$ time. In *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part II*, pages 206–217, 2010. doi:10.1007/978-3-642-15781-3_18.

23 Yahav Nussbaum. *Network flow problems in planar graphs*. PhD thesis, Tel Aviv University, 2014.

24 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inf. Process. Lett.*, 6(3):80–82, 1977. doi:10.1016/0020-0190(77)90031-X.

Towards Unified Approximate Pattern Matching for Hamming and L_1 Distance

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Przemysław Uznański
Department of Computer Science, ETH Zürich, Switzerland
przemyslaw.uznanski@inf.ethz.ch

Abstract

Computing the distance between a given pattern of length n and a text of length m is defined as calculating, for every m-substring of the text, the distance between the pattern and the substring. This naturally generalizes the standard notion of exact pattern matching to incorporate dissimilarity score. For both Hamming and L_1 distance only relatively slow $\widetilde{\mathcal{O}}(n\sqrt{m})$ solutions are known for this generalization. This can be overcome by relaxing the question. For Hamming distance, the usual relaxation is to consider the k-bounded variant, where distances exceeding k are reported as ∞, while for L_1 distance asking for a $(1 \pm \varepsilon)$-approximation seems more natural. For k-bounded Hamming distance, Amir et al. [J. Algorithms 2004] showed an $\widetilde{\mathcal{O}}(n\sqrt{k})$ time algorithm, and Clifford et al. [SODA 2016] designed an $\widetilde{\mathcal{O}}((m + k^2) \cdot n/m)$ time solution. We provide a smooth time trade-off between these bounds by exhibiting an $\widetilde{\mathcal{O}}((m + k\sqrt{m}) \cdot n/m)$ time algorithm. We complement the trade-off with a matching conditional lower bound, showing that a significantly faster *combinatorial* algorithm is not possible, unless the combinatorial matrix multiplication conjecture fails. We also exhibit a series of reductions that together allow us to achieve essentially the same complexity for k-bounded L_1 distance. Finally, for $(1 \pm \varepsilon)$-approximate L_1 distance, the running time of the best previously known algorithm of Lipsky and Porat [Algorithmica 2011] was $\widetilde{\mathcal{O}}(\varepsilon^{-2}n)$. We improve this to $\widetilde{\mathcal{O}}(\varepsilon^{-1}n)$, thus essentially matching the complexity of the best known algorithm for $(1 \pm \varepsilon)$-approximate Hamming distance.

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

Keywords and phrases approximate pattern matching, conditional lower bounds, L_1 distance, Hamming distance

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.62

1 Introduction

The basic question in algorithms on strings is pattern matching, which asks for reporting (or detecting) occurrences of a given pattern P of length m in a text T of length n. A particularly relevant variant of this fundamental question is approximate pattern matching, where the goal is to detect fragments of the text that are *similar* to the pattern. This can be restated as computing the text-to-pattern distance, defined as the distance between P and every m-substring of T. If both P and T are over an integer alphabet Σ, two most natural distance functions are Hamming and L_1. Abrahamson [1] showed how to compute the text-to-pattern Hamming distance with a clever application of boolean convolution: a single convolution can be used to count matches generated by a particular letter in time close to linear, and by carefully partitioning the letters into frequent and non-frequent the overall running time can be guaranteed to be $\mathcal{O}(n\sqrt{m \log m})$. With a somewhat similar approach, the same complexity can be achieved for L_1 distance [5]. Naturally, one would like to design

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 62; pp. 62:1–62:13


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

an $\widetilde{\mathcal{O}}(n)$ time algorithm. However, an unpublished result attributed to Indyk [6] is that any significant improvement for Hamming distance by a *combinatorial* algorithm implies a significant improvement for *combinatorial matrix multiplication*, which is believed to be very hard. Later, a direct reduction from Hamming distance to L_1 distance was shown [14], as well as a reverse reduction [8] together with a full suite of two-way reductions between these two metrics and other functions, linking the corresponding complexities (up to poly-logarithmic factors).

A natural relaxation of computing the text-to-pattern distance is to ask for its $(1 \pm \varepsilon)$-approximation. Karloff [10] showed how to use random $\Sigma \to \{0, 1\}$ projections together with boolean convolution to approximate the Hamming distance in $\mathcal{O}(\varepsilon^{-2} n \log^3 m)$. The ε^{-2} dependency was believed to be tight, given the similarly looking lower bound for sketching Hamming distance [16, 9, 4], but Porat and Kopelowitz [11] refuted this by exhibiting a (complicated) $\mathcal{O}(\varepsilon^{-1} n \log \varepsilon^{-1} \log n \log m \log |\Sigma|)$ time algorithm. Later, they presented a much simpler solution with a slightly better complexity of $\mathcal{O}(\varepsilon^{-1} n \log n \log m)$ [12]. For approximating the L_1 distance, it was only known how to achieve $\mathcal{O}(\varepsilon^{-2} n \log m \log |\Sigma|)$ time complexity [15], and breaking the ε^{-2} barrier was open.

Another natural relaxation is to cap the distances at k, that is, return ∞ if the distance exceeds k. We will call this variant the k-bounded distance. For k-bounded Hamming distance, the classical solution by Landau and Vishkin [13] works in $\mathcal{O}(nk)$ time by checking every m-substring with $k+1$ constant-time longest common extension queries (also known as "kangaroo jumps"). Later, Amir et al. [2] improved this to $\mathcal{O}(n\sqrt{k \log k})$ using boolean convolution similarly as in the classical algorithm of Abrahamson, and also showed an $\mathcal{O}((k^3 \log k + m) \cdot n/m)$ time algorithm. Later, Clifford et al. [7] introduced a new repertoire of tools allowing them to further improve the latter complexity to $\mathcal{O}((k^2 \log k + m \operatorname{polylog} m) \cdot n/m)$. In particular, this is near linear-time for $k = \mathcal{O}(\sqrt{m})$. At a very high level, the improvement was obtained by partitioning both the pattern and the text into $\mathcal{O}(k)$ subpatterns and subtexts, such that the total number of blocks in their run-length encoding is small. This reduces the original problem to $\mathcal{O}(k^2)$ instances of pattern matching with mismatches on run-length encoded inputs, which can be solved in $\widetilde{\mathcal{O}}(k^2)$ total time, leading to an $\widetilde{\mathcal{O}}((k^2 + m) \cdot n/m)$ time algorithm. For k-bounded L_1 distance, only an $\mathcal{O}(n\sqrt{k \log k})$ time algorithm was known [3], and designing an almost linear-time algorithm for polynomially large (in m) values of k was open.

Our results. We provide a smooth transition between the $\widetilde{\mathcal{O}}(n\sqrt{k})$ time algorithm of Amir et al. [2] and the $\widetilde{\mathcal{O}}((m + k^2) \cdot n/m)$ solution given by Clifford et al. [7] for k-bounded Hamming distance. This is obtained by reducing k-bounded Hamming distance to $\mathcal{O}(k)$-RLE Hamming distance, in which the run-length encodings of both the pattern and the text consist of $\mathcal{O}(k)$ blocks, and then designing an efficient algorithm for the latter.

▶ **Theorem 1.** *There is a deterministic algorithm that outputs k-bounded text-to-pattern Hamming distance in time $\mathcal{O}((m \log^2 m \log |\Sigma| + k\sqrt{m \log m}) \cdot n/m)$.*

The complexity from Theorem 1 matches the previous solutions at the extreme points $k = \mathcal{O}(\sqrt{m})$ and $k = \Omega(m)$, but provides a better trade-off in-between. See Figure 1. Furthermore, we prove that this trade-off is essentially the best possible. More precisely, we complement the algorithm with a matching conditional lower bound, showing that a significantly faster *combinatorial* algorithm is not possible, unless the popular combinatorial matrix multiplication conjecture fails.

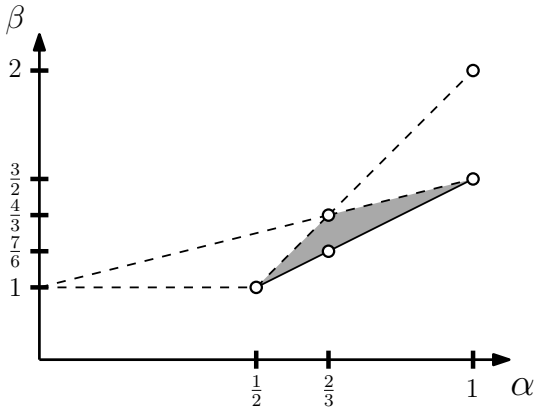


Figure 1 Running time $\mathcal{O}(m^\beta)$ of algorithm from Theorem 1 on instances with $n = \Theta(m)$ and $k = m^\alpha$. Previous algorithms are represented by dashed lines and our algorithm is represented by the solid line. For example, for $k = \Theta(m^{2/3})$ we improve the complexity from $\widetilde{\mathcal{O}}(m^{4/3})$ to $\widetilde{\mathcal{O}}(m^{7/6})$.

▶ **Theorem 2.** *For any positive $\varepsilon, \alpha, \kappa$, such that $\frac{1}{2}\alpha \le \kappa \le \alpha \le 1$ there is no* combinatorial *algorithm solving pattern matching with $k = \Theta(n^\kappa)$ mismatches in time $\mathcal{O}((k\sqrt{m})^{1-\varepsilon} \cdot n/m)$ for a text of length n and a pattern of length $m = \Theta(n^\alpha)$, unless the combinatorial matrix multiplication conjecture fails.*

Next, we move to computing L_1 distance. For computing $(1 \pm \varepsilon)$-approximate text-to-pattern L_1 distance we are able to break the quadratic dependency on $1/\varepsilon$ present in the previous algorithms by designing an $\widetilde{\mathcal{O}}(\varepsilon^{-1}n)$ solution, thus making the complexities of the best known algorithms for $(1 \pm \varepsilon)$-approximate text-to-pattern L_1 distance and Hamming distance essentially equal.

▶ **Theorem 3.** *There is a randomized Monte Carlo algorithm that outputs $(1 \pm \varepsilon)$-approximation of L_1 distance in time $\mathcal{O}(\varepsilon^{-1} n \log^3 n \log m)$. The algorithm is correct with high probability.*

For k-bounded L_1 distance we are able to obtain essentially the same trade-off as for k-bounded Hamming distance.

▶ **Theorem 4.** *There is a deterministic algorithm that outputs k-bounded text-to-pattern L_1 distance in time $\mathcal{O}((m \log^3 m + m \log^2 n + k\sqrt{m \log m} \cdot \log^2 n) \cdot n/m)$.*

In fact, instead of designing a new algorithm for k-bounded L_1 distance we exhibit a series of generic reductions that together allow us to reduce computing k-bounded L_1 distance to computing k-bounded Hamming distance.

▶ **Theorem 5.** *Let $T(n, m, k)$ be the complexity for k-bounded text-to-pattern Hamming distance. Then k-bounded L_1 text-to-pattern distance can be computed in time $\mathcal{O}(n \log^3 m + T(m, m, \mathcal{O}(k)) \cdot \log^2 n \cdot n/m)$.*

Thus, if k-bounded Hamming distance can be computed in time $\widetilde{\mathcal{O}}(n + (k\sqrt{m})^{1-\delta} \cdot n/m)$ for some $\delta \ge 0$ then k-bounded L_1 distance can be computed in time $\widetilde{\mathcal{O}}(n + (k\sqrt{m})^{1-\delta} \cdot n/m)$ as well. Together with a reduction from k-RLE to k-bounded Hamming distance this gives us a clear (although not yet complete) picture of connections between k-bounded and k-RLE text-to-pattern distance under Hamming and L_1 distance summarized in Figure 2.

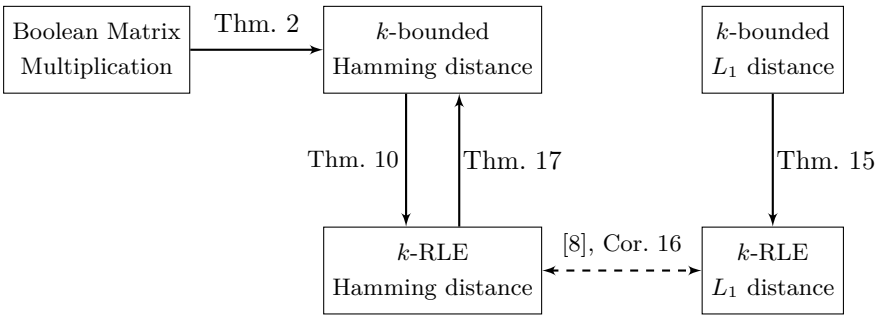


Figure 2 Existing (dashed lines) and new (solid lines) reductions.

Overview of the techniques. Our algorithm for k-bounded Hamming distance (and then also L_1 distance), is a refinement of the approach of Clifford et al. [7]. The general idea is to first consider the periodic structure of the pattern. If the pattern is not very periodic, then m-substrings of the text with a small distance to the pattern cannot occur too often. This allows us to filter the possible occurrences with an algorithm for $(1 \pm \varepsilon)$-approximate Hamming distance and then manually verify the remaining possibilities. Otherwise, the problem can be reduced to multiple smaller instances, in which both the pattern and the text are highly compressible, i.e. their run-length encodings consist of only $\mathcal{O}(k)$ blocks. The first new insight is that, instead of many small instances, it is possible to obtain a single instance of $\mathcal{O}(k)$-bounded Hamming distance, in which the run-length encoding of both the pattern and the text consist of $\mathcal{O}(k)$ blocks. The second observation is that, because the pattern and the text are still of length $\mathcal{O}(m)$, it (again) makes sense to partition the letters into frequent and non-frequent, except that now the threshold is defined with respect to the number of blocks of the pattern containing that letter. For non-frequent letters, we produce a compact representation of their contribution by iterating through every block of the pattern and every block of the text. For frequent letters, we essentially uncompress the corresponding fragments and run the classical convolution.

Our algorithm for $(1 \pm \varepsilon)$-approximate L_1 distance is based on generalized weighted mismatches, similarly as in the previous work [15]: given an arbitrary weight function $\sigma : \Sigma \times \Sigma \to \mathbb{Z}$, we output an array S_σ such that $S[i] = \sum_{j=1}^{m} \sigma(t_{i+j}, p_j)$. This can be computed as follows. For every letter $c \in \Sigma$, construct a new text T^c by setting $T^c[i] = 1$ if $t_i = c$ and $T^c[i] = 0$ otherwise. Similarly, construct a new pattern P^c such that $P^c[i] = \sigma(c, p_i)$. Then, $\sigma(t_{i+j}, p_j) = \sum_{c \in \Sigma} T^c[i + j] \cdot P^c[j]$, so S_σ can be computed with $|\Sigma|$ convolutions in $\mathcal{O}(|\Sigma| n \log m)$ time.

To connect the complexities of computing the text-to-pattern Hamming and L_1 distance we use the recently introduced notion of linearity preserving reductions [8] as a formalization of previously existing reductions between metrics (cf. [14]). The main idea is that in order to show a reduction between two pattern-matching problems, one can represent them as a $(+, \diamond)$ convolution and a $(+, \square)$ convolution, and show how to represent $\square$ as a linear combination of many copies of $\diamond$.

2 Preliminaries

Distance between strings. Let $X = x_1 x_2 \dots x_n$ and $Y = y_1 y_2 \dots y_n$ be two strings over an integer alphabet $[M]$, for some $M = \text{poly}(n)$. We define their L_1 distance as $L_1(X, Y) = \sum_i |x_i - y_i|$, and their Hamming distance as $\text{Ham}(X, Y) = |\{i : x_i \neq y_i\}|$.

Text-to-pattern distance. The text-to-pattern distance between a text $T = t_1 t_2 \dots t_n$ and a pattern $P = p_1 p_2 \dots p_m$ is defined as an array S such that, for every i, $S[i] = d(T[i+1 \,..\, i+m], P)$. Thus, for L_1 distance $S[i] = \sum_{j=1}^{m} |t_{i+j} - p_j|$, while for Hamming distance $S[i] = |\{j \in \{1, \dots, m\} : t[i+j] \neq p[j]\}|$. Then, $(1 \pm \varepsilon)$-approximate distance is an array S_ε such that, for every i, $(1-\varepsilon) \cdot S[i] \leq S_\varepsilon[i] \leq (1+\varepsilon) \cdot S[i]$. Finally, k-bounded distance is an array S_k such that, for every i, $S_k[i] = S[i]$ when $S[i] \leq k$ and $S_k[i] = \infty$ otherwise. Finally, in k-RLE distance we assume that the run-length encoding of both the pattern and the text consist of $\mathcal{O}(k)$ blocks, that is, they contain only $\mathcal{O}(k)$ maximal runs of identical letters.

Model. We assume the standard word RAM model, in which arithmetic operations on integers from $[M]$ take constant time.

Linearity preserving reductions. Say that we want to show a reduction between binary operators $\diamond$ and $\square$. In order for such a reduction to be relevant in the convolutional setting, it needs to be of a specific form. Let t be a fixed integer called the size of the reduction, and suppose there exist integer coefficients $\alpha_1, \dots, \alpha_t$, and functions $f_1, \dots, f_t, g_1, \dots, g_t$ such that, for any x and y, there is $x \square y = \sum_{\ell=1}^{t} \alpha_\ell \cdot (f_\ell(x) \diamond g_\ell(y))$. Then $(+, \square)$-convolution of T and P is computed as a linear combination of $(+, \diamond)$-convolutions $f_\ell(T)$ with $g_\ell(P)$, where $f(X) = f(x_1)f(x_2)\dots f(x_n)$ for $X = x_1 x_2 \dots x_n$. That is, a following equality holds $\sum_{i+j=k} x[i] \square y[j] = \sum_{\ell=1}^{t} \alpha_\ell \cdot \left(\sum_{i+j=k} x[i] \diamond y[j] \right)$.

3 k-bounded Distance

The goal of this section is to prove Theorem 1. We first show how to reduce k-bounded Hamming distance to $\mathcal{O}(n/m)$ instances of $\mathcal{O}(k)$-RLE Hamming distance on inputs of length $\mathcal{O}(m)$. We start with the standard trick of reducing the original problem to $\lceil n/m \rceil$ instances with pattern P of length m and text T of length $2m$ and work with such formulation from now on. Thus, now we need to reduce one such instance to an instance of $\mathcal{O}(k)$-RLE Hamming distance on inputs of length $\mathcal{O}(k)$.

We now highlight the kernelization technique of Clifford et al. [7]. An integer $\pi > 0$ is an x-period of a string $S[1, m]$ if $\mathrm{Ham}(S[\pi+1, m], S[1, m-\pi]) \leq x$ (cf. Definition 1 in [7]). Note that compared to the original formulation, we drop the condition that π is minimal from the definition.

▶ **Lemma 6** (Fact 3.1 in [7]). *If the minimal $2x$-period of the pattern is ℓ, then the starting positions of any two occurrences with x mismatches of the pattern are at distance at least ℓ.*

The first step of the algorithm is to determine the minimal $\mathcal{O}(k)$-period of the pattern. More specifically, we run the $(1+\varepsilon)$-approximate algorithm of Karloff [10] with $\varepsilon = 1$ matching the pattern P against itself. Since Karloff's algorithm is convolution-based, it can be adapted to computing mismatches for partial alignments, that is, between suffixes and prefixes of equal length of the same string. This takes $\mathcal{O}(m \log^2 m \log |\Sigma|)$ time and, by looking at the approximate outputs for offsets not larger than k, allows us to distinguish between two cases: (1) every $4k$-period of the pattern is at least k, or (2) there is an $8k$-period of the pattern that is at most k. Then we run the appropriate algorithm as described below.

No small $4k$-period. We again run Karloff's algorithm with $\varepsilon = 1$, but now we match the pattern with the text. We look for positions i where the 2-approximate algorithm reports at most k mismatches, meaning that $\mathrm{Ham}(P, T[i, i+m-1]) \leq 2k$. By Lemma 6, there

$T' =$ hokuspokusopensezame

h	s	u	e	z
o	p	s	n	a
k	o	o	s	m
u	k	p	e	e

s	u	e	z	#
p	s	n	a	#
o	o	s	m	#
k	p	e	e	#

$P =$ abracadabrab

a	c	b	$	$
b	a	r	$	$
r	d	a	$	$
a	a	b	$	$

$T^\star$ = hsuez opsna koosm ukpee suez# psna# oosm# kpee#

$P^\star$ = acb$$ bar$$ rda$$ aab$$

Figure 3 Example of rearranging the text and the pattern with $\ell = 4$.

are $\mathcal{O}(m/k)$ such positions, and we can safely discard all the others. Then, we test every such position using the "kangaroo jumps" technique of Landau and Vishkin [13], using $\mathcal{O}(k)$ constant-time operations per position, in total $\mathcal{O}(m)$ time.

Small $8k$-period. Let $\ell \leq k$ be any $8k$-period of the pattern. For a string S and $1 \leq i \leq \ell$, let $\{S\}_{\ell,i} = S[i]S[i+\ell]S[i+2\ell]\dots$ up until end of S. We denote by $\{S\}_\ell$ an ℓ-encoding of S, that is the string $\{S\}_{\ell,1}\{S\}_{\ell,2}\dots\{S\}_{\ell,\ell-1}, \{S\}_{\ell,\ell}$. Let runs$(S)$ be the number of runs in S. Denote $\text{runs}_\ell(S) = \sum_{i=1}^{\ell} \text{runs}(\{S\}_{\ell,i})$, and observe that it upperbounds the number of runs in $\{S\}_\ell$.

▶ **Lemma 7** (Lemma 6.1 in [7]). *If P has an $8k$-period ℓ for $\ell \leq k$, then* $\text{runs}_\ell(P) \leq 9k$.

We proceed with the kernelization argument. Let T_L be the longest suffix of $T[1,m]$ such that $\text{runs}_\ell(T_L) \leq 11k$. Similarly, let T_R be the longest prefix of $T[m+1,2m]$ such that $\text{runs}_\ell(T_R) \leq 11k$. Let $T' = T_L T_R$. Obviously, $\text{runs}_\ell(T') \leq 22k$.

▶ **Lemma 8** (Lemma 6.2 in [7]). *Every $T[i,i+m-1]$ that is an occurrence of P with k mismatches is fully contained in T'.*

Thus we see that k-mismatch pattern matching is reduced to a kernel where the ℓ-encoding of both the text and the pattern have few runs, that is, they both compress well with RLE.

From now on assume that both T' and P are of lengths divisible by ℓ. If this is not the case, we can pad them separately with at most $\ell - 1 < k$ characters each, not changing the complexity of our solution. Let m_1 and m_2 be integers such that $m_1 \cdot \ell = |T'|$ and $m_2 \cdot \ell = |P|$, $m_1 \geq m_2$.

We rearrange both P and T' to take advantage of their regular structure. That is, we define $T^\star = \{T'\}_\ell\{T''\}_\ell$, where $T'' = T'[\ell+1, m_1 \cdot \ell]\ \#^\ell$. Observe that $T^\star$ is a string of length $2m_1 \cdot \ell$, composed first of m_1 blocks of the form $T'[i]T'[i+\ell]\dots T'[i+(m_1-1)\ell]$ for $1 \leq i \leq \ell$, and then of m_1 blocks of the form $T'[i+\ell]\dots T'[i+(m_1-1)\ell]\ \#$. Similarly, we define $P^\star = \{P\ \$^{(m_1-m_2)\ell}\}_\ell$. Again we observe that $P^\star$ is the string of length $m_1 \cdot \ell$, composed of blocks of the form $P[i]P[i+\ell]\dots P[i+(m_2-1)\ell]\ \$^{m_1-m_2}$ for $1 \leq i \leq \ell$. An example of this reduction is presented in Figure 3.

Next we show that $T^\star$ and $P^\star$ preserve the Hamming distance between P and any m-substring of T' in the following sense:

▶ **Lemma 9.** *For any integer $0 \leq \alpha \leq (m_1 - m_2)\ell$, let $x = \lfloor \alpha/\ell \rfloor$ and $y = \alpha \bmod \ell$. Let $\beta = x + y \cdot m_1$. Then*

$$\text{Ham}(T'[\alpha+1, \alpha+m_2\cdot\ell], P) = \text{Ham}(T^\star[\beta+1, \beta+m_1\cdot\ell], P^\star) - (m_1 - m_2)\cdot\ell.$$

Proof. Observe that

$$\mathrm{Ham}(T'[\alpha+1,\alpha+m_2\cdot\ell],P)=\sum_{i=0}^{m_2-1}\sum_{j=1}^{\ell}\delta(T'[x\ell+y+i\ell+j],P[i\ell+j]), \tag{1}$$

where δ is indicator of character inequality. Observe that $P[i\ell+j])=P^\star[i+j\cdot m_1]$, for $1\le j\le \ell-y$ there is $T'[x\ell+y+i\ell+j]=T^\star[(x+i)+(y+j)m_1]$, and for $\ell-y<j\le\ell$ there is $T'[x\ell+y+i\ell+j]=T''[(x+i)\ell+(y+j-\ell)]=T^\star[(x+i)+(y+j-\ell)m_1+\ell m_1]=T^\star[(x+i)+(y+j)m_1]$. Additionally, for $m_2\le i<m_1$, $P^\star[i+j\cdot m_1]=\$$, which always generates a mismatch with any character in $T^\star$. Thus

$$(1)=\sum_{i=0}^{m_2-1}\sum_{j=1}^{\ell}\delta(T^\star[(x+i)+(y+j)m_1],P^\star[i+j\cdot m_1])=$$
$$=-(m_1-m_2)\ell+\sum_{i=0}^{m_1-1}\sum_{j=1}^{\ell}\delta(T^\star[(x+i)+(y+j)m_1],P^\star[i+j\cdot m_1]).$$ ◀

We see that it is enough to find all occurrences of $P^\star$ in $T^\star$ with $(k+(m_1-m_2)\cdot\ell)$ mismatches, where $k+(m_1-m_2)\cdot\ell\le 2k$, $|P^\star|=|T'|\le m$ and $|T^\star|=2|T'|\le 2m$. Additionally, $\mathrm{runs}(P^\star)\le 9k+\ell\le 10k$ and $\mathrm{runs}(T^\star)\le 22k+\ell\le 23k$. This gives us the following theorem.

▶ **Theorem 10.** *k-bounded text-to-pattern Hamming distance reduces in $\mathcal{O}(n\log^3 m)$ time to $\mathcal{O}(n/m)$ instances of $\mathcal{O}(k)$-RLE text-to-pattern Hamming distance on inputs of length $\mathcal{O}(m)$.*

Now we describe how to solve an instance of $\mathcal{O}(k)$-RLE Hamming distance.

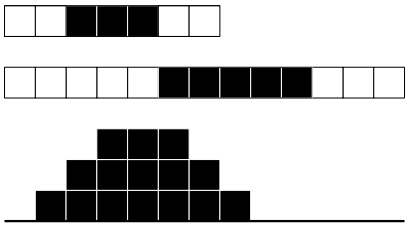
▶ **Lemma 11.** *There is a deterministic algorithm that outputs k-RLE Hamming text-to-pattern distance on inputs of length $\mathcal{O}(m)$ in time $\mathcal{O}(m+k\sqrt{m\log m})$.*

Proof. Consider a letter $c\in\Sigma$. For a string S, we denote by $\mathrm{runs}(S,c)$ the number of runs in S consisting of occurrences of c. Fix a parameter t. Call a letter c such that $\mathrm{runs}(P^\star,c)>t$ a heavy letter, and otherwise call it light. Now we describe how to count the number of mismatches for each type of letters. This is reminiscent to a trick originally used by Abrahmson [1] and later refined by Amir et al. [2].

Heavy letters. For every heavy letter c separately we use a convolution scheme. Since both $P^\star$ and $T^\star$ are of size $\mathcal{O}(m)$, this takes time $\mathcal{O}(m\log m)$ per every such letter. Since $\sum_{c\in\Sigma}\mathrm{runs}(P^\star,c)=\mathrm{runs}(P^\star)\le 10k$, there are $\mathcal{O}(k/t)$ heavy letters, making the total time $\mathcal{O}(mk/t\cdot\log m)$.

Light letters. First, we preprocess $P^\star$, and for every light letter c we compute a list of runs consisting of occurrences of c. Our goal is to compute the array $A[0,|T^\star|-|P^\star|]$, where $A[i]$ counts the number of matching occurrences of light letters in $T^\star[i+1,i+|P^\star|]$ and $P^\star$.

We scan $T^\star$, and for every run of a particular light letter, we iterate through the precomputed list of runs of this letter in $P^\star$. Observe that, given a run of the same letter in $P^\star$ and in $T^\star$, denoted $T^\star[u,v]$ and $P^\star[y,z]$, respectively, their corresponding matches can be seen as a piecewise linear function. More precisely, for all integers $u\le i\le v$ and $y\le j\le z$, we need to increase $A[i-j]$ by one. To see that we can process pair of runs in constant time, we work with discrete derivative, instead of original arrays.



$DA =$ | 1 | 0 | 0 | -1 |

$DB =$ | 1 | 0 | 0 | 0 | 0 | -1 |

$D^2(A \cdot B) =$ | 1 | 0 | 0 | -1 | 0 | -1 | 0 | 0 | 1 |

Figure 4 Left: a run in the pattern and a run in the text (both represented by black boxes) consisting of the same character and a histogram of the matches they generate. Right: first derivatives of the indicator arrays and second derivative of the match array, without padding zeroes.

Given a sequence F, we define its discrete derivative DF as follows: $(DF)[i] = F[i] - F[i-1]$. Correspondingly, if we consider a generating function $F(x) = \sum_i F[i]x^i$, then $(DF)(x) = F(x) \cdot (1-x)$ (for convenience, we assume that arrays are indexed from $-\infty$ to ∞).

Now consider indicator sequences $T_{u,v}[i] = \mathbf{1}(u \le i \le v)$ and $P_{y,z}[j] = \mathbf{1}(-z \le j \le -y)$. To perform the update, we set $A[i+j]$ += $T_{u,v}[i] \cdot P_{y,z}[j]$ for all i, j, or simpler using generating functions:

$$A(x) \mathrel{+}= T_{u,v}(x) \cdot P_{y,z}(x), \tag{2}$$

where $T_{u,v}(x) = \sum_{i=u}^{v} x^i$ and $P_{y,z}(x) = \sum_{j=y}^{z} x^{-j}$. However, we observe that $DT_{u,v}$ and $DP_{y,z}$ have particularly simple forms: $DT_{u,v}(x) = x^u - x^{v+1}$ and $DP_{y,z}(x) = x^{-z} - x^{-y+1}$. Thus it is easier to maintain second derivative of A, and (2) becomes:

$$D^2A(x) \mathrel{+}= x^{u-z} - x^{v-z+1} - x^{u-y+1} + x^{v-y+2}.$$

All in all, we can maintain D^2A in constant time per pair of runs, or in $\mathcal{O}(k \cdot t)$ total time, since every list of runs is of length at most t, and there are at most $23k$ runs in $T^\star$. Additionally, in $\mathcal{O}(m)$ time we can compute $A[0]$ and $A[1]$, allowing us to recover all other $A[i]$s from the formula $A[i] = (D^2A)[i] + 2A[i-1] - A[i-2]$.

Setting $t = \sqrt{m \log m}$ gives the total running time $\mathcal{O}(k\sqrt{m \log m})$ in both cases as claimed. ◀

Combining the reduction from Theorem 10 with Lemma 11 gives us Theorem 1.

4 Lower Bound for k-bounded Hamming Distance

In this section, we present a conditional lower bound for computing k-bounded Hamming distance. The main idea expands upon the proof attributed to Indyk [6], except that we use rectangular matrices instead of square, and use the padding accordingly. We pad using the same character in both text and pattern, increasing the number of mismatches only by a factor of 2.

Recall the combinatorial matrix multiplication conjecture stating that, for any $\varepsilon > 0$, there is no *combinatorial*[1] algorithm for multiplying two $n \times n$ boolean matrices working in time $\mathcal{O}(n^{3-\varepsilon})$. The following formulation is equivalent to this conjecture.

[1] It is not clear what does combinatorial mean *precisely*. However, FFT and so boolean convolution often used in algorithms on strings are considered not to be combinatorial.

▶ **Conjecture 12** (Combinatorial matrix multiplication). *For any $\alpha, \beta, \gamma, \varepsilon > 0$, there is no combinatorial algorithm for multiplying an $n^\alpha \times n^\beta$ matrix with an $n^\beta \times n^\gamma$ matrix in time $\mathcal{O}(n^{\alpha+\beta+\gamma-\varepsilon})$.*

The equivalence can be seen by simply cutting the matrices into square blocks (in one direction) or in rectangular blocks (in the other direction).

Now, consider two boolean matrices, A of dimension $M' \times N$ and B of dimension $N \times M$, for $M' \geq M \geq N$. We encode A as a text T by writing down its elements row-by-row and adding some padding. Namely:

$$T = \#^{M^2} r_1 \ \#^{M-N+1} \ r_2 \ \#^{M-N+1} \ \dots \ \#^{M-N+1} \ r_{M'} \#^{M^2}$$

where $r_i = r_{i,1} \dots r_{i,N}$ and $r_{i,j} = 0$ when $A_{i,j} = 0$ and $r_{i,j} = j$ when $A_{i,j} = 1$. Similarly, we encode B as a pattern P by writing down its elements column-by-column, except that here the padding is shorter by one character:

$$P = c_1 \ \#^{M-N} \ c_2 \ \#^{M-N} \ \dots \ \#^{M-N} \ c_M$$

where $c_j = c_{1,j} \dots c_{N,j}$ and $c_{i,j} = 0'$ when $B_{i,j} = 0$ and $c_{i,j} = i$ when $B_{i,j} = 1$.

Observe that, since we encode 0s from A and B using different symbols, and encoding of 1s is position-dependent, r_i and c_j will generate a match only if they are perfectly aligned and there is k such that $r_{i,k} = c_{k,j}$, or equivalently $A_{i,k} = B_{k,j} = 1$. Since each block (encoded row plus following padding) is either of length $N+1$ for rows or N for columns, there will be at most one aligned row-column pair for each pattern-text alignment.

The total number of mismatches, for each alignment, is at most $2NM$ (since there are at most MN non-# text characters that are aligned with pattern, and at most MN non-# pattern characters). We can determine whether any given entry of $A \cdot B$ is a 1, since if so the number of mismatches for the corresponding alignment is decreased by 1.

We have $|T| = \Theta(M'M)$ and $|P| = \Theta(M^2)$. By setting $M = \sqrt{m}$, $M' = \frac{n}{\sqrt{m}}$ and $N = \frac{k}{\sqrt{m}}$ we obtain Theorem 2.

If we denote by $\omega(\alpha, \beta, \gamma)$ the exponent of fastest algorithm to multiply a matrix of dimension $n^\alpha \times n^\beta$ with a matrix of dimension $n^\beta \times n^\gamma$, we also have:

▶ **Corollary 13.** *For any positive $\varepsilon, \alpha, \kappa$, such that $\frac{1}{2}\alpha \leq \kappa \leq \alpha \leq 1$ there is no algorithm solving pattern matching with $\Theta(n^\kappa)$ mismatches in time $\mathcal{O}(n^{\omega(2-\alpha, 2\kappa-\alpha, \alpha)/2-\varepsilon})$ for a text of length n and a pattern of length $\Theta(n^\alpha)$.*

5 $(1 \pm \varepsilon)$-approximation of L_1 Distance

In this section we prove Theorem 3. We use a procedure `generalized_weighted_matching` $(T, P, \texttt{score})$ that computes, for a text $T = t_1 t_2 \dots t_n$ and a pattern $P = p_1 p_2 \dots p_m$ and an arbitrary weight function $\sigma : \Sigma \times \Sigma \to \mathbb{Z}$, the array S_σ such that $S[i] = \sum_{j=1}^{m} \sigma(t_{i+j}, p_j)$ in in $\mathcal{O}(|\Sigma| n \log m)$ time.

Let $\delta = \frac{\varepsilon}{24 \cdot (3 + \log M)} = \Theta(\varepsilon / \log n)$, and let b be the smallest positive integer such that $2^b \geq 1/\delta$. We claim that with such parameters, Algorithm 1 outputs the desired $(1 \pm \varepsilon)$-approximation in the claimed time. Let S_ε be its output.

▶ **Theorem 14.** *For any i, $S[i] \cdot (1 - \varepsilon) \leq S_\varepsilon[i] \leq S[i] \cdot (1 + \varepsilon)$ with probability at least $2/3$.*

Proof. Consider first $x = x_a$ and $y = y_b$, two characters of the input. We analyze how well Algorithm 1 approximates $|x - y| = \operatorname{sgn}(x - y) \cdot (x - y)$ in the consecutive calls of `generalized_weighted_matching`. First, fix value of Δ and consider the binary representations of $x' = x + \Delta$ and $y' = y + \Delta$. More precisely, let $x' = \sum_i 2^i \cdot \alpha_i$ and $y' = \sum_i 2^i \cdot \beta_i$ for

Algorithm 1: $(1 \pm \varepsilon)$-approximation of text-to-pattern L_1 distance.

Input: Integer strings T and P.
Output: Score vector S_ε.

```
def score(x, y):
    x_0 ← x mod 2
    y_0 ← y mod 2
    if x_0 = y_0 then
        return 0
    else if sgn(x − y) = sgn(x_0 − y_0) then
        return 1
    else
        return −1

def approximate(T, P):
    Δ ← u.a.r. integer from 0 to 2^⌈log M⌉ − 1
    T' ← T + Δ
    P' ← P + Δ
    S_ε ← [0 ... 0]
    for i ← 0 to ⌈log M⌉ do
        T'' ← ⌊T'/2^i⌋ mod 2^b
        P'' ← ⌊P'/2^i⌋ mod 2^b
        S ← generalized_weighted_matching(T'', P'', score)
        S_ε ← S_ε + S · 2^i
    return S_ε
```

some $\alpha_i, \beta_i \in \{0, 1\}$. Algorithm 1 in essence estimates $|x-y| = \operatorname{sgn}(x-y)\sum_i 2^i(\alpha_i - \beta_i)$ with $C = \sum_i 2^i\gamma_i$ where $\gamma_i \in \{-1, 0, 1\}$ is the estimation of a contribution of $(\alpha_i - \beta_i) \cdot \operatorname{sgn}(x - y)$ to $(x' - y')$ and depends only on values of $\alpha_j - \beta_j \in \{-1, 0, 1\}$ for $i \le j < i + b$ in the following way:

- If, for every $i \le j < i + b$ we have $\alpha_j = \beta_j$, then $\gamma_i = 0$.
- Otherwise, let j' be the largest j such that $i \le j < i+b$ and $\alpha_j \neq \beta_j$. If $\alpha_{j'} - \beta_{j'} = 1$, then the local estimation is that $x' > y'$ and so $\gamma_i = \alpha_i - \beta_i$, and otherwise $\gamma_i = -1 \cdot (\alpha_i - \beta_i)$.

Consider $c = \max\{i : \alpha_i \neq \beta_i\}$ and $d = \max\{i : 2^i \le (x' - y')\}$, that is c is the position of the highest bit on which x' and y' differ, and d is the position of the highest bit of $x' - y'$. In general, $c \ge d$, and we say that pair x', y' is *t-bad*, if $c - d = t$.

We first observe that for a x, y pair to be at least t-bad, a following condition must be met: $\lfloor x'/2^{d+t} \rfloor \neq \lfloor y'/2^{d+t} \rfloor$. Since Δ is chosen uniformly at random from a large enough range of integers, there is

$$\sum_{\tau \ge t} \Pr(x', y' \text{ is } \tau\text{-bad} \mid x, y) \le |x - y|/2^{d+t} \le 2^{-t+1}.$$

We also observe following: for any pair x', y', in C, all the coefficients $\gamma_c, \gamma_{c-1}, \ldots, \gamma_{c-b+1}$ are computed correctly, since for any j such that $c \ge j \ge c - b + 1$ there is $j' = c$, and then $\gamma_j = (\alpha_j - \beta_j) \cdot \operatorname{sgn}(x - y)$. Therefore

$$\big|C - |x' - y'|\big| = \left|\sum_{i \le c-b} 2^i(\gamma_i - (\alpha_i - \beta_i) \cdot \operatorname{sgn}(x - y))\right| \le \sum_{i \le c-b} 2 \cdot 2^i < 2 \cdot 2^{c-b+1}.$$

If a pair x', y' is t-bad, it immediately follows that the absolute error of estimation is at most $2^{c-b+2} = 2^{d+t-b+2} \leq |x'-y'|2^{t+2}\delta$.

We now estimate expected error in estimation based on choice of Δ. If a particular pair x, y is t-bad, then $t \leq 1 + \lceil \log M \rceil$. Using the previous observations, we have

$$\mathbb{E}\Big[\big|C - \big|x'-y'\big|\big| \;\Big|\; x,y\Big] = \sum_t \Pr(x',y' \text{ is } t\text{-bad} \mid x,y) \cdot \mathbb{E}\Big[\big|C - \big|x'-y'\big|\big| \;\Big|\; x',y' \text{ is } t\text{-bad}\Big]$$
$$\leq \sum_{t=0}^{1+\lceil \log M \rceil} 2^{-t+1}|x-y|2^{t+2}\delta = (3+\log M)8\delta|x-y| = \frac{\varepsilon}{3}|x-y|.$$

By linearity of expectation $\mathbb{E}\Big[\big|S_\varepsilon[i] - S[i]\big|\Big] \leq \frac{\varepsilon}{3}S[i]$, and by Markov's inequality the claim follows. ◀

Now, a standard amplification technique applies: it is enough to repeat Algorithm 1 independently p times and take the median value from $S_\varepsilon^{(1)}[i], S_\varepsilon^{(2)}[i], \ldots, S_\varepsilon^{(p)}[i]$ as the final estimate $\widehat{S}_\varepsilon[i]$. Taking $p = \Theta(\log n)$ to be large enough makes the final estimate good with high probability, and by the union bound whole $\widehat{S}_\varepsilon$ is a good estimate of S.

The complexity of Algorithm 1 is dominated by `generalized_weighted_matching` being invoked $\mathcal{O}(\log n)$ times on alphabet of size $2^b = \Theta(\varepsilon^{-1}\log n)$. Each such invocation takes $\mathcal{O}(2^b n \log m) = \mathcal{O}(\varepsilon^{-1} n \log n \log m)$, and Algorithm 1 takes $\mathcal{O}(\varepsilon^{-1} n \log^2 n \log m)$ time and the total time for computing $(1 \pm \epsilon)$-approximation is $\mathcal{O}(\varepsilon^{-1} n \log^3 n \log m)$.

6 Reductions

In this section we design a series of reductions to complete the picture from Figure 2. We start with an L_1 version of Theorem 10.

▶ **Theorem 15.** *k-bounded text-to-pattern L_1 distance problem reduces in $\mathcal{O}(n \log^3 m)$ time to $\mathcal{O}(n/m)$ instances of $\mathcal{O}(k)$-RLE text-to-pattern L_1 distance on inputs of length $\mathcal{O}(m)$, where both the pattern and the text might contain wildcards.*

Proof. As in the proof of Theorem 10, we can assume that the length of T is $2m$. We observe that if the L_1 distance of an m-substring of T is at most k, so must be its Hamming distance. Therefore, we can use exactly the same filtering approach. In the case of no small $4k$-period, after filtering m-substrings with more than $2k$ mismatches we are left with $\mathcal{O}(m/k)$ possibilities, which are then verified using the "kangaroo jumps" technique of Landau and Vishkin [13]. The only required modification is that now for every found mismatch we calculate the corresponding increase in the L_1 distance. In the case of a small $8k$-period, we use the same transformation, except that all special characters become identical wildcards $*$ with L_1 distance 0 to every letter. This preserves the L_1 distance by the same argument. ◀

Then, instead of designing an algorithm for computing k-RLE L_1 distance, we apply the following reduction.

▶ **Corollary 16** (Theorem 2.1, Theorem 2.2 and Lemma A.1 in [8]). *For any $M \geq 0$, there is a linearity preserving reduction from L_1 distance between integers from $[M]$ to $\mathcal{O}(\log^2 M)$ instances of Hamming distance. There is a converse reduction from Hamming distance to $\mathcal{O}(1)$ instances of L_1 distance. Both reductions allow for wildcards in the input and output wildcard-less instances.*

By inspecting the proof of the above reduction we see that it does not create any new runs, that is, allows us to reduce k-RLE L_1 distance to $\mathcal{O}(\log^2 M)$ instances of k-RLE Hamming distance. Therefore, together with Theorem 15 and Lemma 11, we obtain an $\mathcal{O}((m + k\sqrt{m \log m}) \cdot \log^2 n + n \log^3 m)$ time algorithm for k-bounded L_1 distance as claimed in Theorem 4.

To complete the picture, we show a reduction from k-RLE Hamming distance to $2k$-bounded Hamming distance, that is, a converse to Theorem 10.

▶ **Theorem 17.** *k-RLE text-to-pattern Hamming distance on inputs of length $\mathcal{O}(m)$ reduces to $\mathcal{O}(1)$ instances of $2k$-bounded Hamming distance on inputs of length $\mathcal{O}(m)$.*

Proof. We proceed similarly as in Lemma 11. We observe that it is enough to compute the second discrete derivative of the output array S, that is D^2S defined as $(D^2S)[i] = S[i+2] - 2S[i+1] + S[i]$, since D^2S and two initial values of S (computed naively in time $\mathcal{O}(m)$) are enough to recover S. For any two blocks $t_u t_{u+1} \dots t_{v-1} t_v$ and $p_y p_{y+1} \dots p_{z-1} p_z$ of the same letter, D^2S needs to be updated in only 4 places, that is $D^2S[u - z] += 1$, $D^2S[v - z + 1] -= 1$, $D^2S[u - y + 1] -= 1$ and $D^2S[v - y + 2]$. We now explain how to deal with the first kind of updates, with the other three being implemented similarly.

We first reduce the problem to *k-sparse text-to-pattern Hamming distance*, where the text and the pattern are of length $\mathcal{O}(m)$ and have each at most k regular characters, with every other character being wildcard $*$ (special character having distance 0 to every other character). We construct sparse instance as follows: for every position t_u in T that starts a block, we set $T_{\text{sparse}}[u] = t_u$, and similarly in pattern for a position p_z (that ends a block), we set $P_{\text{sparse}}[z] = p_z$. Observe, that if $t_u \neq p_z$, then in the answer there is $S_{\text{sparse}}[u - z] += 1$, and if $t_u = p_z$ then S_{sparse} remains unchanged. That is, the answer counts mismatches, while we want to count matches. To invert the answer, we create T_{bin} such that $T_{\text{bin}}[i] = 1$ iff $T_{\text{sparse}}[i] \neq *$ and $T_{\text{bin}}[i] = 0$ otherwise, and P_{bin} in an analogous manner. Convolving T_{bin} and P_{bin} gives us, for every alignment, the total number of non-special text characters aligned with non-special pattern characters, and we obtain the answer with a single subtraction.

To reduce from k-sparse instances of Hamming distance to $2k$-bounded Hamming distance, we follow an analogous reduction from Lemma A.1 in [8] that reduces Hamming distance on $\mathbb{N}^+ + \{*\}$ to Hamming distance on $\mathbb{N}$. First, create a new text T_1 such that $T_1[i] = T[i]$ iff $T[i] \neq *$ and $T_1[i] = 0$ iff $T[i] = *$, and similarly to obtain P_1. Second, create a new text T_2 such that $T_2[i] = 1$ iff $T_2[i] \neq *$ and $T_2[i] = 0$ iff $T[i] = *$, and similarly to obtain P_2. Now we observe that $\text{Ham}(T[i], P[j]) = \text{Ham}(T_1[i], P_1[j]) - \text{Ham}(T_2[i], P_2[j])$, thus it is enough to compute exact Hamming text-to-pattern distances on these two instances and subtract them. However, we observe that in both of them, there are in total at most $2k$ characters different than 0, thus $2k$-bounded Hamming distance works just as fine. ◀

7 Conclusion and Open Problems

Showing a reduction from either *bounded RLE* L_1 distance, or *sparse* L_1 distance to k-approximated L_1 distance would suffice in completing a converse to Theorem 5, and prove that complexity of k-bounded L_1 and Hamming distances is the same, up to poly-logarithmic factors.

References

1 Karl R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987.

2 Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004.

3 Amihood Amir, Ohad Lipsky, Ely Porat, and Julia Umanski. Approximate matching in the L_1 metric. In *CPM*, pages 91–103, 2005.

4 Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of gap-hamming-distance. *SIAM J. Comput.*, 41(5):1299–1317, 2012.

5 Peter Clifford, Raphaël Clifford, and Costas S. Iliopoulos. Faster algorithms for δ,γ-matching and related problems. In *CPM*, pages 68–78, 2005.

6 Raphaël Clifford. Matrix multiplication and pattern matching under Hamming norm. `http://www.cs.bris.ac.uk/Research/Algorithms/events/BAD09/BAD09/Talks/BAD09-Hammingnotes.pdf`, 2009. Retrieved March 2017.

7 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. The k-mismatch problem revisited. In *SODA*, pages 2039–2052, 2016.

8 Daniel Graf, Karim Labib, and Przemyslaw Uznański. Hamming distance completeness and sparse matrix multiplication. *CoRR*, abs/1711.03887, 2017. `arXiv:1711.03887`.

9 T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of hamming distance. *Theory of Computing*, 4(1):129–135, 2008.

10 Howard J. Karloff. Fast algorithms for approximately counting mismatches. *Inf. Process. Lett.*, 48(2):53–60, 1993.

11 Tsvi Kopelowitz and Ely Porat. Breaking the variance: Approximating the hamming distance in $1/\epsilon$ time per alignment. In *FOCS*, pages 601–613, 2015.

12 Tsvi Kopelowitz and Ely Porat. A simple algorithm for approximating the text-to-pattern hamming distance. In *SOSA*, pages 10:1–10:5, 2018.

13 Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theor. Comput. Sci.*, 43:239–249, 1986.

14 Ohad Lipsky and Ely Porat. L_1 pattern matching lower bound. *Inf. Process. Lett.*, 105(4):141–143, 2008.

15 Ohad Lipsky and Ely Porat. Approximate pattern matching with the L_1, L_2 and L_∞ metrics. *Algorithmica*, 60(2):335–348, 2011.

16 David P. Woodruff. Optimal space lower bounds for all frequency moments. In *SODA*, pages 167–175, 2004.

A Faster Construction of Greedy Consensus Trees

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Gad M. Landau
University of Haifa, Israel
landau@cs.haifa.ac.il

Wing-Kin Sung
National University of Singapore, Singapore
ksung@comp.nus.edu.sg

Oren Weimann[1]
University of Haifa, Israel
oren@cs.haifa.ac.il

Abstract

A consensus tree is a phylogenetic tree that captures the similarity between a set of conflicting phylogenetic trees. The problem of computing a consensus tree is a major step in phylogenetic tree reconstruction. It is also central for predicting a species tree from a set of gene trees, as indicated recently in [Nature 2013].

This paper focuses on two of the most well-known and widely used consensus tree methods: the greedy consensus tree and the frequency difference consensus tree. Given k conflicting trees each with n leaves, the previous fastest algorithms for these problems were $\mathcal{O}(kn^2)$ for the greedy consensus tree [J. ACM 2016] and $\widetilde{\mathcal{O}}(\min\{kn^2, k^2n\})$ for the frequency difference consensus tree [ACM TCBB 2016]. We improve these running times to $\widetilde{\mathcal{O}}(kn^{1.5})$ and $\widetilde{\mathcal{O}}(kn)$ respectively.

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

Keywords and phrases phylogenetic trees, greedy consensus trees, dynamic trees

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.63

1 Introduction

A *phylogenetic tree* describes the evolutionary relationships among a set of n species called taxa. It is an unordered rooted tree whose leaves represent the taxa and whose inner nodes represent their common ancestors. Each leaf has a distinct label from $[n]$. The inner nodes are unlabelled and have at least two children.

Numerous phylogenetic trees, reconstructed from data sources like fossils or DNA sequences, have been published in the literature since the early 1860s. However, the phylogenetic trees obtained from different data sources or using different reconstruction methods result in conflicts (similar though not identical phylogenetic trees over the same set $[n]$ of leaf labels). The conflicts between phylogenetic trees are usually measured by their difference in *signatures*: The signature of a phylogenetic tree T is the set $\{\mathsf{L}(u) : u \in T\}$ where $\mathsf{L}(u)$ denotes the set of labels of all leaves in the subtree rooted at node u of T (the set $\mathsf{L}(u)$ is sometimes called a *cluster*). To deal with the conflicts between k phylogenetic trees in a

[1] Supported in part by Israel Science Foundation grant 592/17

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 63; pp. 63:1–63:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

systematic manner, the concept of a *consensus tree* was invented. Informally, the consensus tree is a single phylogenetic tree that summarizes the branching structure (signatures) of all the conflicting trees. That is, given a collection of k phylogenetic trees with the same set of leaf labels $[n]$, we would like to build a single phylogenetic tree that captures as much of their structure as possible (in practice, we might want to relax this assumption and only require that each set of leaf labels is a subset of $[n]$, but we assume that it is exactly $[n]$ as in the previous theoretical work). Of course, there are many possibilities of how this single phylogenetic tree should be chosen.

Many different types of consensus trees have been proposed in the literature. For almost all of them, optimal or near-optimal $\widetilde{\mathcal{O}}(kn)$ time constructions are known. These include Adam's consensus tree [1], strict consensus tree [27], loose consensus tree [4, 13], majority-rule consensus tree [17, 13], majority-rule (+) consensus tree [11], and asymmetric median consensus tree [20, 21][2]. Two of the most notable exceptions are the frequency difference consensus tree [10] and the greedy consensus tree [5, 9] whose running time remains quadratic in either k or n. In particular, the former can be constructed in $\widetilde{\mathcal{O}}(\min\{kn^2, k^2n\})$ time [11] and the later in $\mathcal{O}(kn^2)$ time [13]. For more details about different consensus trees and their advantages and disadvantages see the survey in [5], Chapter 30 in [8], and Chapter 8.4 in [31].

In this paper we propose novel worst-case efficient algorithms for the frequency difference consensus tree problem and the greedy consensus tree problem.

First, we present an $\mathcal{O}(kn \log^2 n)$ time deterministic labeling method. The labeling method counts the frequency (number of occurrences) of every cluster S in the input trees. Based on this labeling method, we obtain an $\mathcal{O}(kn \log^2 n)$ time construction of the frequency difference consensus tree. Then, for the greedy consensus tree, we present our main technical contribution: a method that uses micro-macro decomposition to verify if a cluster S is compatible with a tree T in $\mathcal{O}(n^{0.5} \log n)$ time and, if so, modify T to include S in $\mathcal{O}(n^{0.5} \log n)$ amortized time. Using this procedure, we obtain an $\mathcal{O}(kn^{1.5} \log n)$ time construction of the greedy consensus tree.

The frequency difference consensus tree. The frequency $f(S)$ of a cluster S (a set of labels of all leaves in some subtree) is the number of trees that contain S. A cluster is said to be *compatible* with another cluster if they are either disjoint or one is included in the other. A *frequent* cluster is a cluster that occurs in more trees than any of the clusters that are incompatible with it. The frequency difference consensus tree is a tree whose signature is exactly all the frequent clusters. Such a tree always exists because, for any pair of incompatible clusters, at most one will be included, and so all included clusters are pairwise compatible.

The frequency difference consensus tree was initially proposed by Goloboff et al. [10], and its relationship with other consensus trees was studied in [7]. In particular, it can be seen as a refinement of the majority-rule consensus tree [17, 13]. Moreover, it is known to give less noisy branches than the greedy consensus tree defined below. Steel and Velasco [30] concluded that "the frequency difference method is worthy of more widespread usage and serious study". A naive construction of the frequency difference consensus tree takes $\mathcal{O}(k^2n^2)$ time. The free software TNT [10] has implemented a heuristics method to construct it more efficiently. However, its time complexity remains unknown.

[2] Constructing the asymmetric median consensus tree was proven to be NP-hard for $k > 2$ [20] and solvable in $\widetilde{\mathcal{O}}(n)$ time for $k = 2$ [21].

Recently, Jansson et al. [11] presented an $\mathcal{O}(\min\{kn^2, k^2n + kn\log^2 n\})$ time construction (implemented in the FACT software package [12]). Their algorithm first computes the frequency $f(S)$ of every cluster S with non-zero frequency. This is done in total $\mathcal{O}(\min\{kn^2, k^2n\})$ time. They then show that given these computed frequencies, the frequency difference consensus tree can be computed in additional $\mathcal{O}(kn\log^2 n)$ time. In Section 2 we show how to compute all frequencies in total $\mathcal{O}(kn\log^2 n)$ time leading to the following theorem:

▶ **Theorem 1.** *The frequency difference consensus tree of k phylogenetic trees $T_1, T_2, \ldots, T_k$ on the same set of leaves $[n]$ can be computed in $\mathcal{O}(kn\log^2 n)$ time.*

To prove the above theorem, we first develop an $\mathcal{O}(kn\log^2 n)$ time algorithm for assigning a number $\mathsf{id}(u) \in [kn]$ to every $u \in T_i$ such that $\mathsf{id}(u) = \mathsf{id}(u')$ iff $\mathsf{L}(u) = \mathsf{L}(u')$. With these numbers in hand, we can then compute the frequencies of all clusters in $\mathcal{O}(kn)$ time using counting sort (since there are only kn clusters with non-zero frequencies, and each was assigned an integer bounded by kn). Notice that this also generates a sorted list of all clusters with non-zero frequencies.

The greedy consensus tree. We say that a given collection $\mathcal{C}$ of subsets of $[n]$ is *consistent* if there exists a phylogenetic tree T such that the signature of T is exactly $\mathcal{C}$. The greedy consensus tree is defined by the following procedure: We begin with an initially empty $\mathcal{C}$ and then consider all clusters S in decreasing order of their frequencies. In this order, for every S, we check if $\mathcal{C} \cup \{S\}$ is consistent, and if so we add S to $\mathcal{C}$.

The greedy consensus tree is one of the most well-known consensus trees. It has been used in numerous papers such as [6, 23, 14, 18, 2, 24, 29, 19, 3, 15, 16, 26, 33] to name a few. For example, in a recent landmark paper in Nature [23], it was used to construct the species tree from 1000 gene trees of yeast genomes, and in [6] it was asserted that *"The greedy consensus tree offers some robustness to gene-tree discordance that may cause other methods to fail to recover the species tree. In addition, the greedy consensus method outperformed our other methods for branch lengths outside the too-greedy zone."*.

The greedy consensus tree is a refinement of the majority-rule consensus tree, and is sometimes called the extended majority-rule consensus (eMRC) tree. It is implemented in popular phylogenetics software packages like PHYLIP [9], PAUP* [32], MrBayes [22], and RAxML [28]. A naive construction of the greedy consensus tree requires $\mathcal{O}(kn^3)$ time [5]. To speed this up, these software packages often use hashing to improve the running time. Thus, if one is interested in analyzing worst-case complexity of the algorithms used in these packages, it would be necessary to allow randomization, as otherwise there is no guarantee on the efficiency of hashing. Even with randomization, the worst-case time complexities of these solutions are not known. Recently, Jansson et al. [13] gave the best known provable construction with an $\mathcal{O}(kn^2)$ deterministic running time (their implementation is also part of the FACT package). In Section 3 we present our main contribution, a deterministic $\widetilde{\mathcal{O}}(kn^{1.5})$ construction as stated by the following theorem:

▶ **Theorem 2.** *The greedy consensus tree of k phylogenetic trees $T_1, T_2, \ldots, T_k$ on the same set of leaves $[n]$ can be computed in $\mathcal{O}(kn^{1.5}\log n)$ time.*

To prove the above theorem, we develop a generic procedure that takes any ordered list of clusters $S_1, S_2, \ldots, S_\ell \subseteq [n]$ and tries adding them one-by-one to the current solution $\mathcal{C}$. We assume that every cluster S_i is specified by providing a tree T_i and a node $u_i \in T_i$ such that $S_i = \mathsf{L}(u_i)$. Our procedure requires $\mathcal{O}(n^{0.5}\log n)$ time per cluster (to add this cluster to $\mathcal{C}$ or assert that it cannot be added) and needs not to assume anything about the order of the clusters. In particular, it does not rely on the clusters being sorted by frequencies.

2 Computing the Identifiers

We process the nodes of every T_i in a bottom-up order. For every node $u \in T_i$, we compute the identifier $\mathsf{id}(u)$ by updating the following structure called the *dynamic set equality structure*:

▶ **Lemma 3** (the dynamic set equality structure)**.** *There exists a data structure that maintains a set of integers under the following operations: (1) create a new empty set in constant time, (2) add $x \in [n]$ to the set in $\mathcal{O}(\log^2 n)$ time, (3) return the identifier of the set in constant time, and (4) list all ℓ elements of the set in $\mathcal{O}(\ell)$ time. The structure ensures that the identifiers are bounded by the total number of update operations performed so far, and that two sets are equal iff their identifiers are equal.*

Proof. To allow for listing all elements of the current set S, we store them in a list. Before adding the new element x to the list, we need to check if $x \in S$. This will be done using the representation described below.

Conceptually, we work with a complete binary tree B on n leaves labelled with $0, 1, \ldots, n-1$ when read from left to right (without losing generality, n is a power of 2), where every node u corresponds to a set $D(u) \subseteq [n]$ defined by the leaves in its subtree (note that $D(u) = \{i, i+1, \ldots, j\}$, where $0 \leq i \leq j < n$). Now, any set S is associated with a binary tree B, where we write 1 in a leaf if the corresponding element belongs to S and 0 otherwise. Then, for every node we define its characteristic vector by writing down the values written in the leaves of its subtree in the natural order (from left to right). Clearly, the vector of an inner node is obtained by concatenating the vector of its children. We want to maintain identifiers of all nodes, so that the identifiers of two nodes are equal iff their characteristic vectors are identical. If we can keep the identifiers small, then the identifier of the current set can be computed as the identifiers of the root of B.

Assume that we have already computed the identifiers of all nodes in B and now want to add x to S. This changes the value in the leaf u corresponding to x and, consequently, the characteristic vectors of all ancestors of u. However, it does not change the characteristic vectors of any other node. Therefore, we traverse the ancestors of u starting from u and recompute their identifiers. Let v be the current node. If we have never seen the characteristic vector of v before, we can set the identifier of v to be the largest already used identifier plus one. Otherwise, we have to set the identifier of v to be the same as the one previously used for a node with such a characteristic vector. As mentioned above, the characteristic vector of an inner node v is the concatenation of the characteristic vectors of its children v_ℓ and v_r. We maintain a dictionary mapping a pair consisting of the identifier of v_ℓ and the identifier of v_r to the identifier of v. The dictionary is global, that is, shared by all instances of the structure. Then, assuming that we have already computed the up-to-date identifiers of v_ℓ and v_r, we only need to query the dictionary to check if the identifier of v should be set to the largest already used identifier plus one (which is exactly when the dictionary does not contain the corresponding pair) or retrieve the appropriate identifier. Therefore, adding x to B reduces to $\log n$ queries to the dictionary. By implementing the dictionary with balanced search trees, we therefore obtain the claimed $\mathcal{O}(\log^2 n)$ time for adding an element.

We are not completely done yet, because creating a new complete binary tree B takes $\mathcal{O}(n)$ time and therefore the initialization time is not constant yet. However, we can observe that it does not make sense to explicitly maintain a node u of B such that $S \cap D(u) = \emptyset$, because we can assume that the identifier of such an u is 0. In other words, we can maintain only the part of B induced by the leaves corresponding to S. Adding an element $x \in S$ is implemented as above, except that we might need to create (at most $\mathcal{O}(\log n)$) new nodes

on the leaf-to-root path corresponding to x (if such a leaf already exists, we terminate the procedure as $x \in S$ already) and then recompute the identifiers on the whole path as described above. ◀

Armed with Lemma 3, we process every T_i bottom-up. Consider an inner node $v \in T_i$ and let $v_1, v_2, \ldots, v_d$ be its children ordered so that $|\mathsf{L}(v_1)| = \max_j |\mathsf{L}(v_j)|$, that is, the subtree rooted at v_1 is the largest. Assuming that we have already stored every $\mathsf{L}(v_j)$ in a dynamic set equality structure, we construct a dynamic set equality structure storing $\mathsf{L}(v)$ by simply inserting all elements of $\mathsf{L}(v_2) \cup \mathsf{L}(v_3) \cup \cdots \cup \mathsf{L}(v_d)$ into the structure of $\mathsf{L}(v_1)$. This takes $\mathcal{O}(\log^2 n)$ time per element. Then, we set $\mathsf{id}(u)$ to be the identifier of the obtained structure. By a standard argument (heavy path decomposition), every leaf of T_i is inserted into at most $\log n$ structures and therefore the whole T_i is processed in $\mathcal{O}(n \log^3 n)$ time. This gives us the claimed $\mathcal{O}(kn \log^3 n)$ total time.

We now proceed with a faster $\mathcal{O}(kn \log^2 n)$ total time solution. While this is irrelevant for our $\mathcal{O}(kn^{1.5} \log n)$ time construction of the greedy consensus tree, it implies a better complexity for constructing the frequency difference consensus tree.

We start with a high-level intuition. Lemma 3 is, in a sense, more than we need, as it is not completely clear that we need to immediately compute the identifier of the current set. Indeed, applying heavy path decomposition we can partially delay computing the identifiers by proceeding in $\mathcal{O}(\log n)$ phases. In each phase, we can then replace the dynamic dictionary used to store the mapping with a radix sort. Intuitively, this shaves one log from the time complexity. We proceed with a detailed explanation.

▶ **Theorem 4.** *The numbers $\mathsf{id}(u)$ can be found for all nodes of the k phylogenetic trees $T_1, T_2, \ldots, T_k$ in $\mathcal{O}(kn \log^2 n)$ total time.*

Proof. For a node $v \in T_i$, define its level $\mathsf{level}(v)$ to be ℓ, such that $2^\ell \leq |\mathsf{L}(v)| < 2^{\ell+1}$. Thus, the levels are between 0 and $\log n$, level of a node is at least as large as the levels of its children, and a node on level ℓ has at most one child on the same level. We work in phases $\ell = 0, 1, \ldots, \log n$. In phase ℓ, we assume that the numbers $\mathsf{id}(v)$ are already known for all nodes v, such that $\mathsf{level}(v) < \ell$, and want to assign these numbers to all nodes v, such that $\mathsf{level}(v) = \ell$. We will show how to achieve this in $\mathcal{O}(kn \log n)$ time, thus proving the theorem.

Consider all nodes v, such that $\mathsf{level}(v) = \ell$. Because every such v has at most one child at the same level, all level-ℓ nodes in T_i can be partitioned into maximal paths of the form $p = v_1 - v_2 - \ldots - v_s$, where the level of the parent of v_1 is larger than ℓ (or v_1 is the root of T_i), and the levels of all children of v_j (except for v_{j+1}, if defined) are smaller than ℓ. v_1 is called the head of p and denoted $\mathsf{head}(p)$. Now, our goal is to find $\mathsf{id}(v_j)$ with the required properties for every $j = 1, 2, \ldots, s$. We will actually achieve a bit more. The sets $\mathsf{L}(\mathsf{head}(p))$ are disjoint in every tree T_i, and thus we can define, for every i, a partition $\mathcal{P}_i = \{P_i(1), P_i(2), \ldots, P_i(t_i)\}$ of the set of leaves $[n]$, where every $P_i(z)$ corresponds to a level-ℓ path $p = v_1 - v_2 - \ldots - v_s$ in T_i, such that $\mathsf{L}(\mathsf{head}(p)) = P_i(z)$. The elements of $P_i(z)$ are then ordered, and we think that $P_i(z)$ is a sequence of length $|P_i(z)|$. The ordering is chosen so that, for every $j = 1, 2, \ldots, s$, the set $\mathsf{L}(v_j)$ corresponds to some prefix of $P_i(z)$. $P_i(z)[1..r]$ denotes the prefix of $P_i(z)$ of length r. We will assign identifiers to all such prefixes $P_i(z)[1..r]$, for every $i = 1, 2, \ldots, k$, $z = 1, 2, \ldots, t_i$ and $r = 1, 2, \ldots, |P_i(z)|$, with the property that the identifiers of two prefixes are equal iff the sets of leaves appearing in both of them are equal. Then, we can extract the required $\mathsf{id}(v_j)$ in constant time each by taking the identifiers of some $P_i(z)[1..r]$.

Recall that in the slower solution we worked with a complete binary tree B on n leaves. For every set S in the collection and every $u \in B$, we computed an identifier of the set $S \cap D(u)$. This was possible, because if u_ℓ and u_r are the left and the right child of u,

respectively, then the identifier of $S \cap D(u)$ can be found using the identifiers of $S \cap D(u_\ell)$ and $S \cap D(u_r)$. We need to show that retrieving these identifiers can be batched.

Fix a node $u \in B$ and, for every $i = 1, 2, \ldots, k$ and $z = 1, 2, \ldots, t_i$, consider all prefixes $P_i(z)[1..r]$ for $r = 1, 2, \ldots, |P_i(z)|$. We create a *version* of u for every such prefix. The version corresponds to the set containing all elements of $D(u)$ occurring in the prefix $P_i(z)[1..r]$. We want to assign identifiers to all versions of u. First, observe that we only have to create a new version if $P_i(z)[r] \in D(u)$, as otherwise the set is the same as for $r-1$. Thus, the total number of required versions, when summed over all nodes $u \in B$ on the same depth in B, is only kn, as a leaf of T_i creates exactly one new version for some u. For every node $u \in B$, we will store a list of all its versions. A version consists of its identifier (such that the identifier of two versions is the same iff the corresponding sets are equal) together with the indices i, z and r. We describe how to create such a list for every node $u \in B$ at the same depth d given the lists for all nodes at depth $d+1$ next.

Let u_1 and u_2 be the left and the right child of $u \in B$, respectively. Then, we need to create a new version of u for every new version of u_1 and every new version of u_2, because for the set corresponding to u to change either the set corresponding to u_1 or the set corresponding to u_2 must change, and every change is adding one new element. Fix i and z and consider all versions of u_1 corresponding to i and z sorted according to r. Let the sorted list of their r's be $a_1 < a_2 < \ldots$. Similarly, consider all versions of u_2 corresponding to i and z sorted according to r, and let the sorted list of their r's be $b_1 < b_2 < \ldots$. For every $x \in \{a_1, a_2, \ldots\} \cup \{b_1, b_2, \ldots\}$, we create a new version of u corresponding to i, z, and r equal to x. This is done by retrieving the version of u_1 with r equal to a_p, such that $a_p \leq x$ and p is maximized, and the version of u_2 with r equal to b_q, such that $b_q \leq x$ and q is maximized. Then, the identifier of the new version of u can be constructed from the pair consisting of the identifiers of these versions of u_1 and u_2 (this is essentially the same reasoning as in the slower solution). We could now use a dictionary to map these pairs to identifiers. However, we can also observe that, in fact, we have reduced finding the identifiers of all versions of all nodes $u \in B$ at the same depth d to identifying duplicates on a list of kn pairs of numbers from $[kn]$. This can be done by radix sorting all pairs in linear time (more precisely, $\mathcal{O}(kn)$ time and $\mathcal{O}(kn)$ space), and then sweeping through the sorted list while assigning the identifiers. This takes only $\mathcal{O}(kn)$ time for every depth d, so $\mathcal{O}(kn \log n)$ for every level as claimed. ◀

The proof of Theorem 1 follows immediately from Theorem 4.

3 Simulating the Greedy Algorithm

We consider k trees $T_1, \ldots, T_k$ on the same set of leaves $[n]$, and assume that every node u has an identifier $\mathsf{id}(u)$ such that $\mathsf{id}(u) = \mathsf{id}(u')$ iff $\mathsf{L}(u) = \mathsf{L}(u')$. We next develop a general method for maintaining a solution $\mathcal{C}$ (i.e., a set of compatible identifiers) so that, given any node $u \in T_i$, we are able to efficiently check if $\mathsf{L}(u)$ is compatible with $\mathcal{C}$, meaning that $\mathcal{C} \cup \mathsf{L}(u)$ is consistent, and if so add $\mathsf{L}(u)$ to $\mathcal{C}$. Our method does not rely on the order in which the sets arrive and in particular can be used to run the greedy algorithm.

We represent $\mathcal{C}$ with a phylogenetic tree T_c such that $\mathcal{C} = \{\mathsf{L}(u) : u \in T_c\}$. T_c is called the *current consensus tree*. By Lemma 2.2 of [13], S is compatible with $\mathcal{C}$ iff the node $v \in T_c$ defined as the lowest common ancestor of all leaves with labels from S has the property that, for every child v' of v, either $\mathsf{L}(v') \cap S = \emptyset$ or $\mathsf{L}(v') \subseteq S$. Recall that the lowest common ancestor (lca) of u and v is the deepest node w that is an ancestor of both u and v. Also, adding $\mathsf{L}(u)$ to $\mathcal{C}$ can be done by creating a new child w of v and reconnecting every original child v' of v such that $\mathsf{L}(v') \subseteq S$ to the new w. This is illustrated in Figure 1 (left).

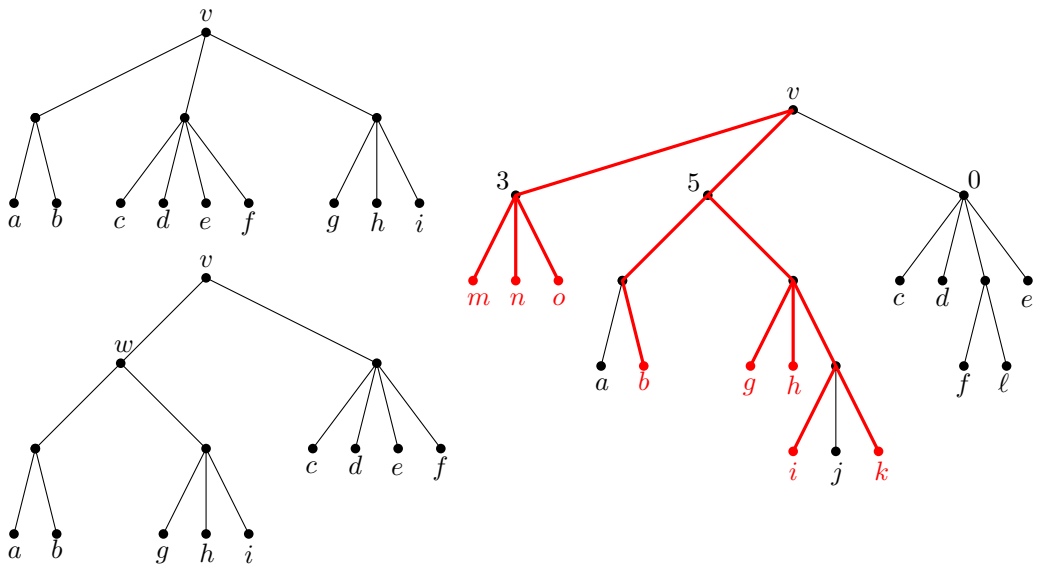


Figure 1 Left: adding $\{a, b, g, h, i\}$ to S. Right: checking if $S = \{m, n, o, b, g, h, i, k\}$ is compatible with T_c. Leaves corresponding to the elements of S are shown in red and their lca is v. S is not compatible with T_c because the counter of the middle child of v is equal to 5 yet there are 7 leaves in its subtree.

Initially, T_c consists only of n leaves attached to the common root (which corresponds to $\mathcal{C} = \{\{x\} : x \in [n]\}$). Our goal is to maintain some additional information so that given any node $u \in T_i$, we can check if $\mathsf{L}(u)$ is compatible with $\mathcal{C}$ in $\mathcal{O}(n^{0.5} \log n)$ time. After adding $\mathsf{L}(u)$ to $\mathcal{C}$ the information will be updated in amortized $\mathcal{O}(kn^{0.5} \log n)$ time. To explain the intuition, we first show how to check if $\mathsf{L}(u)$ is compatible with $\mathcal{C}$ in roughly $\mathcal{O}(|\mathsf{L}(u)|)$ time.

Let $\mathsf{L}(u) = \{\ell_1, \ell_2, \ldots, \ell_s\}$ and let u_i be the leaf of T_c labelled with ℓ_i. Let v be the lowest common ancestor of $u_1, u_2, \ldots, u_s$ found by asking $s - 1$ lca queries: we start with u_1 and then iteratively jump to the lca of the current node and u_i. Assuming that we represent T_c in such a way that an lca query can be answered efficiently, this takes roughly $\mathcal{O}(s)$ time. Then, we need to decide if for every child v' of v it holds that $\mathsf{L}(v') \subseteq \mathsf{L}(u)$ or $\mathsf{L}(v') \cap \mathsf{L}(u) = \emptyset$. This can be done by computing, for every such v', how many u_i's belong to the subtree rooted at v', and then checking if this number is either 0 or $|\mathsf{L}(v')|$. To compute these numbers, we maintain a counter for every v'. Then, for every u_i we retrieve the child v' of v such that u_i belongs to the subtree rooted at v' and increase the counter of v'. Assuming that we represent T_c so that such v' can be retrieved efficiently, this again takes roughly $\mathcal{O}(s)$ time. Finally, we iterate over all u_i again, retrieve the corresponding v' and check if its counter is equal to $|\mathsf{L}(v')|$ (so our representation of T_c should also allow retrieving the number of leaves in a subtree). If not, then $\mathsf{L}(u)$ is not compatible with $\mathcal{C}$, see Figure 1 (right). Otherwise, we create the new node w and reconnect to w all children v' of v, such that the counter of v' is equal to $|\mathsf{L}(v')|$.

We would like to avoid explicitly iterating over all elements of $\mathsf{L}(u)$. This will be done by maintaining some additional information, so that we only have to iterate over up to $n^{0.5}$ elements. To explain what is the additional information we need the (standard) notion of a *micro-macro decomposition*. Let b be a parameter and consider a binary tree on n nodes. We want to partition it into $\mathcal{O}(n/b)$ node-disjoint subtrees called *micro trees*. Each micro tree is of size at most b and contains at most two *boundary nodes* that are adjacent to nodes

in other micro trees. One of these boundary nodes, called the top boundary node, is the root of the whole micro tree, and the other is called the bottom boundary node. Such a partition is always possible and can be found in $\mathcal{O}(n)$ time.

We binarize every T_i to obtain T_i' (this could be avoided by working with edge-disjoint subtrees in the decomposition, but we find node-disjoint subtrees easier to think about; binarization adds a number of artificial nodes u for which we do not check if $\mathsf{L}(u)$ is compatible with $\mathcal{C}$)). Then, we find a micro-macro decomposition of T_i' with $b = n^{0.5}$ (where b has been chosen as to minimize the total running time). By properties of the decomposition we have the following:

▶ **Proposition 5.** *For any $u \in T_i$ such that $|\mathsf{L}(u)| > n^{0.5}$, there exists a boundary node $v \in T_i'$ such that $\mathsf{L}(u)$ can be obtained by adding at most $n^{0.5}$ elements to $\mathsf{L}(v)$. Furthermore, v and these up to $n^{0.5}$ elements can be retrieved in $\mathcal{O}(n^{0.5})$ time after $\mathcal{O}(n)$ preprocessing.*

The total number of boundary nodes is only $\mathcal{O}(kn^{0.5})$. For each such boundary node u, we maintain a pointer to a node $\mathsf{finger}(u) \in T_c$ called the *finger* of u, defined as the lowest common ancestor in T_c of all leaves with labels belonging to $\mathsf{L}(u)$. Additionally, the children of $\mathsf{finger}(u)$ are partitioned into three groups: (1) v_i such that $\mathsf{L}(v_i) \subseteq \mathsf{L}(u)$, (2) v_i such that $\mathsf{L}(v_i) \cap \mathsf{L}(u) = \emptyset$, and (3) the rest. We call them full, empty, and mixed, respectively (with respect to u). For each group we maintain a list storing all nodes in the group, every node knows its group, and the group knows it size. Additionally, every group knows the total number of leaves in all subtrees rooted at its nodes.

We also need to augment the representation T_c to allow for efficient *extended lca queries*. An extended lca query, denoted $\mathsf{lca\_ext}(u, v)$, returns the first edge on the path from the lca of u and v to u, and -1 if u is an ancestor of v. For example, in Figure 1 (right), $\mathsf{lca\_ext}(v, k) = -1$ whereas $\mathsf{lca\_ext}(h, k)$ is the edge between v and its leftmost child. The following lemma follows by slightly tweaking the link/cut trees of Sleator and Tarjan [25].

▶ **Lemma 6.** *We can maintain a collection of rooted trees under: (1) create a new tree consisting of a single node, (2) make the root of one tree a child of a node in another tree, (3) delete an edge from a node to its parent, (4) count leaves in the tree containing a given node, and (5) extended lca queries, all in $\mathcal{O}(\log n)$ amortized time, where n is the total size of all trees in the collection.*

We next show how to efficiently check for any u if $\mathsf{L}(u)$ is compatible with $\mathcal{C}$. By the following lemma, this can be done in $\mathcal{O}(n^{0.5} \log n)$ time, assuming we have stored the required additional information. Recall that this additional information includes:

1. The tree T_c maintained using Lemma 6.
2. For every boundary node w, we store $\mathsf{finger}(w)$.
3. For every boundary node w, we store three lists containing the full, the mixed, and the empty children of $\mathsf{finger}(w)$ respectively. Each list also stores the total number of leaves in all subtrees rooted at its nodes.

▶ **Lemma 7.** *Assuming access to the above additional information, given any node $u \in T_i$ we can check if $\mathsf{L}(u)$ is compatible with $\mathcal{C}$ in $\mathcal{O}(n^{0.5} \log n)$ time.*

Proof. By Lemma 2.2 of [13], to check if $\mathsf{L}(u)$ is compatible with $\mathcal{C}$ we need to check if, for a node v defined as the lowest common ancestor of all leaves with labels belonging to $\mathsf{L}(u)$, it holds that for every child v' of v either $\mathsf{L}(v') \cap \mathsf{L}(u) = \emptyset$ or $\mathsf{L}(v') \subseteq \mathsf{L}(u)$. By properties of the micro-macro decomposition, we can retrieve a boundary node w and a set S of up to $n^{0.5}$ labels such that $\mathsf{L}(u) = \mathsf{L}(w) \cup S$ (if $|\mathsf{L}(u)| < n^{0.5}$, there is no w). See Figure 2. Then, the

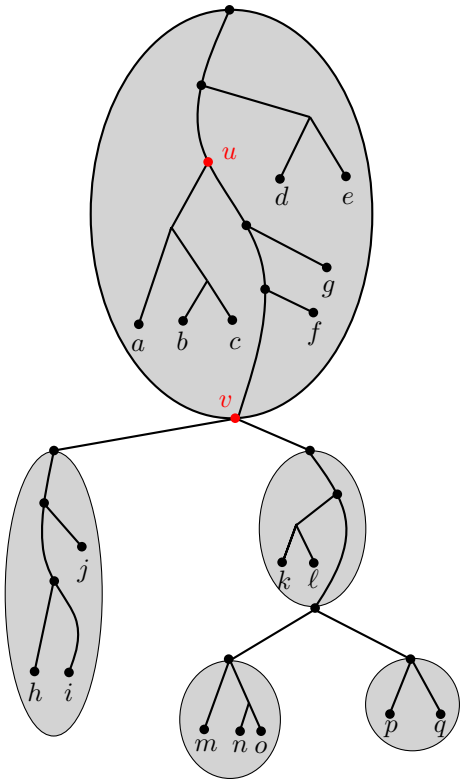


Figure 2 A schematic illustration of the micro-macro decomposition. v is a boundary node and $\mathsf{L}(v) = \{h, i, j, k, \ell, m, n, o, p, q\}$. Then, $\mathsf{L}(u) = \{a, b, c, f, g, h, i, j, k, \ell, m, n, o, p, q\}$ so $\mathsf{L}(u) = \mathsf{L}(v) \cup \{a, b, c, f, g\}$.

lowest common ancestor of all leaves with labels belonging to $\mathsf{L}(u)$ is the lowest common ancestor of $\mathsf{finger}(w)$ and all leaves with labels belonging to S. Therefore, v can be found with $|S|$ lca queries in $\mathcal{O}(n^{0.5} \log n)$ time. Second, to check if $\mathsf{L}(v_i) \cap \mathsf{L}(u) = \emptyset$ or $\mathsf{L}(v_i) \subseteq \mathsf{L}(u)$ for every child v_i of v we distinguish two cases:

(1) If v is a proper ancestor of $\mathsf{finger}(w)$ we can calculate $|\mathsf{L}(v_i) \cap \mathsf{L}(u)|$ for every v_i in $\mathcal{O}(|S| \log n) = \mathcal{O}(n^{0.5} \log n)$ time as follows. Every edge has its associated counter. We assume that all counters are set to zero before starting the procedure and will make sure that they are cleared at the end. First, we use an $\mathsf{lca\_ext}(w, v)$ query to access the edge leading to the subtree containing w and set its counter to $|\mathsf{L}(w)|$. Then, we iterate over all $\ell \in S$, retrieve the leaf u of T_c labelled with ℓ, and use an $\mathsf{lca\_ext}(u, v)$ query to access the edge leading to the subtree of v containing u and increase its counter by one. Additionally, whenever we access an edge for the first time (in this particular query), we add it to a temporary list Q. After having processed all $\ell \in S$, we iterate over $(v, v_i) \in Q$ and check if the counter of (v, v_i) is equal to the number of leaves in the subtree rooted at v_i (which requires retrieving the number of leaves). If this condition holds for every $(v, v_i) \in Q$ then $\mathsf{L}(u)$ is compatible with $\mathcal{C}$ and furthermore, the nodes v_i such that $(v, v_i) \in Q$ are exactly the ones that should be reconnected. Finally, we iterate over the edges in Q again and reset their counters.

(2) If $v = \mathsf{finger}(w)$ the situation is a bit more complicated because we might not have enough time to explicitly iterate over all children of v that should be reconnected. Nevertheless, we can use a very similar method. Every edge has its associated counter (again,

we assume that the counter are set to zero before starting the procedure and will make sure that they are cleared at the end). We also need a global counter g, that is set to the total number of leaves in all subtrees rooted at either full or mixed children of v decreased by $|\mathsf{L}(w)|$. g can be initialized in constant time in the first step of the procedure due to the additional information stored with every list of children. Intuitively, g is how many leaves not belonging to $\mathsf{L}(w)$ we still have to see to conclude that indeed $\mathsf{L}(v_i) \cap \mathsf{L}(u) = \emptyset$ or $\mathsf{L}(v_i) \subseteq \mathsf{L}(u)$ for every child v_i of v. We iterate over $\ell \in S$ and access the edge (v, v_i) leading to the subtree containing u labelled with ℓ. We decrease g by one and, if v_i is an empty child of v and this is the first time we have seen v_i (in this query) then we add the number of leaves in the subtree rooted at v_i to g. If, after having processed all $\ell \in S$, $g = 0$ then we conclude that $\mathsf{L}(u)$ is compatible with $\mathcal{C}$. The whole process takes $\mathcal{O}(|S| \log n) = \mathcal{O}(n^{0.5} \log n)$ time. ◀

Before explaining the details of how to update the additional information, we present the intuition. Recall that adding $\mathsf{L}(u)$ to $\mathcal{C}$ is done by creating a new child v' of v and reconnecting some children of v to v'. Let the set of all children of v be C and the set of children that should be reconnected be C_r. Note that if $|C_r| = 1$ or $|C| = |C_r|$ then we do not have to change anything in T_c. Otherwise, updating T_c can be implemented using two different methods:

1. Delete edges from nodes in C_r to v. Create a new tree consisting of a single node v' and make it a child of v. Then, make all nodes in C_r children of v'.
2. Delete edges from nodes in $C \setminus C_r$ to v. Delete the edge from v to its parent w. Create a new tree consisting of a single node v' and make it a child of w. Then, make v a child of v' and also make all nodes in $C \setminus C_r$ children of w. See Figure 3.

Thus, by using C_r or $C \setminus C_r$, the number of operations can be either $\mathcal{O}(|C_r|)$ or $\mathcal{O}(|C| - |C_r|)$. We claim that by choosing the cheaper option we can guarantee that the total time for modifying the link-cut tree representation of T_c is $\mathcal{O}(n \log^2 n)$. Intuitively, every edge of the final consensus tree participates in $\mathcal{O}(\log n)$ operations, and there are at most n such edges. This is formalized in the following lemma.

▶ **Lemma 8.** $\min\{|C_r|, |C| - |C_r|\}$ *summed over all updates of* T_c *is* $n \log n$.

Proof. We assume that $2 \leq |C_r| < |C|$ in every update, as otherwise there is nothing to change in T_c. Then, there are at most n updates, as each of them creates a new inner node and there are never any nodes with degree 1 in T_c.

We bound the sum of $\min\{|C_r|, |C| - |C_r|\}$ by assigning credits to inner nodes of T_c. During the execution of the algorithm, a node u with b siblings should have $\log b$ credits. Thus, whenever we create a new inner node we need at most $\log n$ new credits, thus the total number of allocated credits is $n \log n$. It remains to argue that, whenever we create a new child v' of v and reconnect some of its children, the original credits of v can be used to pay for the update and make sure that all children of v and v' have enough credits after the update.

Denoting $x = |C_r|$ and $y = |C| - |C_r|$, the cost of the update is $\min\{x, y\}$. The total number of credits of all children of v before the update is $(x + y) \log(x + y - 1)$. After the update, the number of credits of all children of v is $(y + 1) \log y \leq y \log y + \log n$ and the number of credits of all children of v' is $x \log(x - 1)$. Ignoring the $\log n$ new credits allocated to v', the number of available credits is thus:

$$(x + y) \log(x + y - 1) - y \log y - x \log(x - 1) = x \log(1 + y/(x - 1)) + y \log(1 + (x - 1)/y)$$

which is at least $\min\{x, y\}$ for $x \geq 2$, so enough to pay $\min\{|C_r|, |C| - |C_r|\}$ for the update. Hence, the sum is at most $n \log n$. ◀

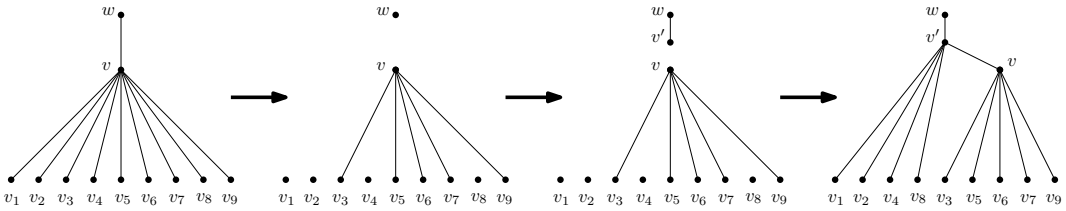


Figure 3 Reconnecting children v_3, v_5, v_6, v_7, v_9 of v using the second method.

Before presenting the whole update procedure, we need one more technical lemma.

▶ **Lemma 9.** *The procedure for checking if* $\mathsf{L}(u)$ *is compatible with* $\mathcal{C}$ *can be requested to return* C_r *in* $\mathcal{O}(|C_r| + n^{0.5})$ *time or* $C \setminus C_r$ *in* $\mathcal{O}(|C| - |C_r| + n^{0.5})$ *time.*

Proof. By inspecting the proof of Lemma 7, we see that there are two cases depending on whether v is a proper ancestor of $\mathsf{finger}(w)$ or not.

1. If v is a proper ancestor of $\mathsf{finger}(w)$ then C_r can be obtained from Q. More precisely, for every $(v, v_i) \in Q$ we add v_i to C_r in $\mathcal{O}(|C_r|)$ total time. We can also obtain $C \setminus C_r$ in $\mathcal{O}(|C|) = \mathcal{O}(|C \setminus C_r| + |S|) = \mathcal{O}(|C| - |C_r| + n^{0.5})$ time.
2. If $v = \mathsf{finger}(w)$ then, while iterating over $\ell \in S$, if this is the first time we have seen v_i then we add v_i to C_r. Additionally, we add all full children of v to C_r. Thus, C_r can be generated in $\mathcal{O}(|C_r|)$ time. Similarly, $C \setminus C_r$ consists of all empty children of v without the nodes v_i seen when iterating over $\ell \in S$, and so can be generated in $\mathcal{O}(|C \setminus C_r| + |S|) = \mathcal{O}(|C| - |C_r| + n^{0.5})$ time.

Thus, we can always generate C_r in $\mathcal{O}(|C_r| + n^{0.5})$ time and $C \setminus C_r$ in $\mathcal{O}(|C| - |C_r| + n^{0.5})$ time. ◀

To add $\mathsf{L}(u)$ to $\mathcal{C}$, we will need to iterate over either C_r or $C \setminus C_r$ (depending on which is smaller). After paying additional $\mathcal{O}(n^{0.5})$ time we can assume that we have access to a list of the elements in the appropriate set. The additional time sums up to $\mathcal{O}(n^{1.5})$, because there can be only n distinct new sets added to $\mathcal{C}$.

▶ **Lemma 10.** *If* $\mathsf{L}(u)$ *is compatible with* $\mathcal{C}$ *then, after adding* $\mathsf{L}(u)$ *to* $\mathcal{C}$ *and modifying* T_c *we can update all additional information in amortized* $\mathcal{O}(kn^{0.5} \log n)$ *time assuming that we add* n *such sets.*

Proof. Recall that T_c is maintained using the data structure from Lemma 6, and adding $\mathsf{L}(u)$ to $\mathcal{C}$ is implemented by creating a new child v' of v and reconnecting some of the children of v to v'. C is the set of all children of v and C_r is the set of children of v that are reconnected to v'. If $|C_r| \leq |C| - |C_r|$ we iterate over C_r and reconnect them one-by-one. If $|C_r| > |C| - |C_r|$ we iterate over $C \setminus C_r$ and reconnect them to a new node w that is inserted between v and its parent. To iterate over either C_r or $C \setminus C_r$, we extend the query procedure as explained in Lemma 9. This adds $\mathcal{O}(n^{0.5})$ to the time complexity, but then we can assume that the requested set can be generated in time proportional to its size. To unify the case of $|C_r| \leq |C| - |C_r|$ and $|C_r| > |C| - |C_r|$, we think that v is replaced with two nodes v' and v'', where v' is the parent of v''. All nodes in C_r become children of v'' while all nodes of $C \setminus C_r$ become children of v' after iterating over either C_r or $C \setminus C_r$, depending on which set is smaller, so by Lemma 8 in the whole process we iterate over sets of total size $n \log n$, so only amortized $\log n$ assuming that we add n sets $\mathsf{L}(u)$.

Consider a boundary node u. If $\mathsf{finger}(u) \neq v$ then there is no need to update the additional information concerning u. If $\mathsf{finger}(u) = v$ then we need to decide if the finger of u should be set to v' or v'' and update the partition of the children of $\mathsf{finger}(u)$ accordingly.

$\mathsf{finger}(u)$ should be set to v' exactly when, for any $w \in C \setminus C_r$, $\mathsf{L}(w) \cap \mathsf{L}(u) = \emptyset$ or, in other words, all nodes in $C \setminus C_r$ are empty with respect to u. The groups should be updated as follows:

1. If $\mathsf{finger}(u)$ is set to v'' then we should remove all nodes in $C \setminus C_r$ from the list of empty nodes with respect to u (as they are no longer children of $\mathsf{finger}(u)$). Other groups remain unchanged.
2. If $\mathsf{finger}(u)$ is set to v' then we should remove all nodes in C_r from the lists. Additionally, we need to insert v'' into the appropriate group: full if all nodes in C_r were full, empty if all nodes in C_r were empty, and mixed otherwise.

We need to show that all these conditions can be checked by either iterating over the nodes of C or over the nodes of $C \setminus C_r$, because we want to iterate over the smaller of these. This then guarantees that the amortized cost of updating the additional information for a boundary node is only $\mathcal{O}(\log n)$, so amortized $\mathcal{O}(kn^{0.5} \log n)$ overall.

To check if all nodes in $C \setminus C_r$ are empty with respect to u, we can either iterate over the nodes in $C \setminus C_r$ or iterate over all nodes in C_r and check if all nodes in C that are full or empty in fact belong to C_r (this is possible because we also keep the total number of full and empty nodes in C). Thus, we can check if $\mathsf{finger}(u)$ should be set to v'.

If $\mathsf{finger}(u)$ is set to v' we need to decide where to put v''. We only explain how to decide if all nodes in C_r are full, as the procedure for empty is symmetric. We can either iterate over all nodes in C_r and check that they are full or iterate over all nodes in $C \setminus C_r$ and check that all nodes in C that are empty or mixed in fact belong to $C \setminus C_r$ (and thus do not belong to C_r, so all nodes in C_r are full). Finally, we add the number of leaves in the subtree rooted at v'' (extracted in $\mathcal{O}(\log n)$ time) to the appropriate sum.

It remains to describe how to remove all unnecessary nodes from the lists. Here we do not worry about having to iterate over the smaller set, because there are only $\mathcal{O}(n)$ new edges created during the whole execution of the algorithm, so we can afford to explicitly iterate over the nodes that should be removed, that is, over C or $C \setminus C_r$. For every removed node, we also subtract the number of leaves in its subtree (extracted in $\mathcal{O}(\log n)$ time) from the appropriate sum. Overall, this adds $\mathcal{O}(n \log n)$ per boundary node to the time complexity, so only amortized $\mathcal{O}(kn^{0.5} \log n)$ overall. ◀

References

1 E. N. Adams III. Consensus techniques and the comparison of taxonomic trees. *Systematic Zoology*, 21(4):390–397, 1972.

2 Md. Shamsuzzoha Bayzid and Tandy J. Warnow. Naive binning improves phylogenomic analyses. *Bioinformatics*, 29(18):2277–2284, 2013.

3 M.S. Bayzid, S. Mirarab, B. Boussau, and T. Warnow. Weighted statistical binning: enabling statistically consistent genome-scale phylogenetic analyses. *PLOS One*, page e0129183, 2015.

4 K. Bremer. Combinable component consensus. *Cladistics*, 6(4):369–372, 1990.

5 D. Bryant. A classification of consensus methods for phylogenetics. In *Bioconsensuss*, volume 61 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 163–184. American Mathematical Society, 2003.

6 J. H. Degnan, M. DeGiorgio, D. Bryant, and N. A. Rosenberg. Properties of consensus methods for inferring species trees from gene trees. *Systematic Biology*, 58(1):35–54, 2009.

7 J. Dong, D. Fernández-Baca, F. R. McMorris, and R. C. Powers. Majority-rule (+) consensus trees. *Mathematical Biosciences*, 228(1):10–15, 2010.

8 J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., Sunderland, Massachusetts, 2004.

9 J. Felsenstein. PHYLIP, version 3.6. Software package, Department of Genome Sciences, University of Washington, Seattle, U.S.A., 2005.

10 P. A. Goloboff, J. S. Farris, and K. C. Nixon. TNT, a free program for phylogenetic analysis. *Cladistics*, 24(5):774–786, 2008.

11 Jesper Jansson, Ramesh Rajaby, Chuanqi Shen, and Wing-Kin Sung. Algorithms for the majority rule (+) consensus tree and the frequency difference consensus tree. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2016.

12 Jesper Jansson, Chuanqi Shen, and Wing-Kin Sung. Fast Algorithms for Consensus Trees (FACT). `http://compbio.ddns.comp.nus.edu.sg/~consensus.tree`, 2013.

13 Jesper Jansson, Chuanqi Shen, and Wing-Kin Sung. Improved algorithms for constructing consensus trees. *Journal of the ACM*, 63(3):1–24, 2016.

14 E. D. Jarvis et al. Whole-genome analyses resolve early branches in the tree of life of modern birds. *Science*, 346(6215):1320–1331, 2014.

15 Liang Liu, Lili Yu, and Scott Edwards. A maximum pseudo-likelihood approach for estimating species trees under the coalescent model. *BMC Evolutionary Biology*, 10:302, 2010.

16 Liang Liu, Lili Yu, Laura Kubatko, Dennis K. Pearl, and Scott V. Edwards. Coalescent methods for estimating phylogenetic trees. *Molecular Phylogenetics and Evolution*, 53(1):320–328, 2009.

17 T. Margush and F. R. McMorris. Consensus n-trees. *Bulletin of Mathematical Biology*, 43(2):239–244, 1981.

18 S. Mirarab, S. Bayzid, and T. Warnow. Evaluating summary methods for multilocus species tree estimation in the presence of incomplete lineage sorting. *Systematic Biology*, 65(3):366–380, 2016.

19 James B. Pease, David C. Haak, Matthew W. Hahn, and Leonie C. Moyle. Phylogenomics reveals three sources of adaptive variation during a rapid radiation. *PLOS Biology*, 14(2):1–24, 2016.

20 Cynthia Phillips and Tandy J. Warnow. The asymmetric median tree — a new model for building consensus trees. *Discrete Applied Mathematics*, 71(1-3):311–335, 1996.

21 Ramesh Rajaby and Wing-Kin Sung. Computing asymmetric median tree of two trees via better bipartite matching algorithm. In *IWOCA*, 2017.

22 F. Ronquist and J. P. Huelsenbeck. MrBayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics*, 19(12):1572–1574, 2003.

23 L. Salichos and A. Rokas. Inferring ancient divergences requires genes with strong phylogenetic signals. *Nature*, 497:327–331, 2013.

24 Leonidas Salichos, Alexandros Stamatakis, and Antonis Rokas. Novel information theory-based measures for quantifying incongruence among phylogenetic trees. *Molecular Biology and Evolution*, 31(5):1261–1271, 2014.

25 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

26 Jordan V. Smith, Edward L. Braun, and Rebecca T. Kimball. Ratite nonmonophyly: independent evidence from 40 novel loci. *Systematic Biology*, 62(1):35–49, 2013.

27 R. R. Sokal and F. J. Rohlf. Taxonomic congruence in the Leptopodomorpha re-examined. *Systematic Zoology*, 30(3):309–325, 1981.

28 A. Stamatakis. Raxml version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014.

29 Alexandros Stamatakis, Paul Hoover, Jacques Rougemont, and Susanne Renner. A rapid bootstrap algorithm for the raxml web servers. *Systematic Biology*, 57(5):758, 2008.

30 M. Steel and J. D. Velasco. Axiomatic opportunities and obstacles for inferring a species tree from gene trees. *Systematic Biology*, 63(5):772–778, 2014.

31 Wing-Kin Sung. *Algorithms in Bioinformatics: A Practical Introduction*. Chapman & Hall/CRC, Boca Raton, Florida, 2010.

32 D. L. Swofford. PAUP*, version 4.0. Software package, Sinauer Associates, Inc., Sunderland, Massachusetts, 2003.

33 Jimmy Yang and Tandy J. Warnow. Fast and accurate methods for phylogenomic analyses. *BMC Bioinformatics*, 12(S-9):S4, 2011.

A Faster FPTAS for #Knapsack

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Liran Markin[1]
University of Haifa, Israel
liran.markin@gmail.com

Oren Weimann[2]
University of Haifa, Israel
oren@cs.haifa.ac.il

Abstract

Given a set $W = \{w_1, \ldots, w_n\}$ of non-negative integer weights and an integer C, the #Knapsack problem asks to count the number of distinct subsets of W whose total weight is at most C. In the more general integer version of the problem, the subsets are multisets. That is, we are also given a set $\{u_1, \ldots, u_n\}$ and we are allowed to take up to u_i items of weight w_i.

We present a deterministic FPTAS for #Knapsack running in $O(n^{2.5}\varepsilon^{-1.5}\log(n\varepsilon^{-1})\log(n\varepsilon))$ time. The previous best deterministic algorithm [FOCS 2011] runs in $O(n^3\varepsilon^{-1}\log(n\varepsilon^{-1}))$ time (see also [ESA 2014] for a logarithmic factor improvement). The previous best randomized algorithm [STOC 2003] runs in $O(n^{2.5}\sqrt{\log(n\varepsilon^{-1})} + \varepsilon^{-2}n^2)$ time. Therefore, for the case of constant ε, we close the gap between the $\tilde{O}(n^{2.5})$ randomized algorithm and the $\tilde{O}(n^3)$ deterministic algorithm.

For the integer version with $U = \max_i\{u_i\}$, we present a deterministic FPTAS running in $O(n^{2.5}\varepsilon^{-1.5}\log(n\varepsilon^{-1}\log U)\log(n\varepsilon)\log^2 U)$ time. The previous best deterministic algorithm [TCS 2016] runs in $O(n^3\varepsilon^{-1}\log(n\varepsilon^{-1}\log U)\log^2 U)$ time.

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

Keywords and phrases knapsack, approximate counting, K-approximating sets and functions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.64

1 Introduction

Given a set $W = \{w_1, \ldots, w_n\}$ of non-negative integer weights and an integer C, the #Knapsack problem asks to count the number of distinct subsets of W whose total weight is at most C. This problem is the counting version of the well known Knapsack problem and is #P-hard. While there are many, celebrated, randomized polynomial-time algorithms for approximately counting #P-hard problems, the #Knapsack problem is one of the few examples where there is also a deterministic approximation algorithm (other notable examples are [8, 14, 1]).

From a geometric view, the #Knapsack problem is equivalent to finding the number of vertices of the n-dimensional hypercube that lie on one side of a given n-dimensional hyperplane. The problem is also related to pseudorandom generators for halfspaces (see

[1] Supported in part by Israel Science Foundation grant 592/17
[2] Supported in part by Israel Science Foundation grant 592/17

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 64; pp. 64:1–64:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

e.g. [2, 9, 11]) as these imply deterministic (though not polynomial-time) approximation schemes for #Knapsack by enumerating over all input seeds to the generator.

Approximately counting knapsack solutions. The #Knapsack problem can be solved with the following simple recursion: $S(i,j) = S(i-1,j) + S(i-1,j-w_i)$ where $S(i,j)$ is the number of subsets of $\{w_1, ..., w_i\}$ whose weight sums to at most j. This recurrence immediately implies a pseudo-polynomial $O(nC)$ time algorithm. More interestingly, this recurrence is the basis of all existing fully polynomial-time approximation schemes (FPTAS). That is, algorithms that for any $\varepsilon > 0$ estimate the number of solutions to within relative error $(1 \pm \varepsilon)$ in time polynomial in n and in $1/\varepsilon$.

Dyer et al. [4] were the first to show how to approximate this recurrence with random sampling. They gave a randomized sub-exponential $2^{O(\sqrt{n}\log^{2.5} n))}\epsilon^{-2}$ time algorithm. Using a more complicated random sampling (with a rapidly mixing Markov chain), Morris and Sinclair [10] obtained the first FPRAS (fully-polynomial *randomized* approximation scheme) running in $O(n^{4.5+\epsilon} + \epsilon^{-2}n^2)$ time. Dyer [3] further improved this to $O(n^{2.5}\sqrt{\log(n\varepsilon^{-1})} + \epsilon^{-2}n^2)$ by using a surprisingly simple sampling procedure (combined with randomized rounding). This to date is the fastest known randomized solution. As for deterministic solutions, the fastest solution to date is by Rizzi and Tomescu [12] and runs in $O(n^3\epsilon^{-1}\log\epsilon^{-1}/\log n)$ time. It is a logarithmic factor improvement (obtained by discretizing the recursion $S(i,j)$ with floating-point arithmetic) over the previous fastest $O(n^3\epsilon^{-1}\log(n\varepsilon^{-1}))$ time solutions of Gopalan et al. [6] (who used read-once branching programs inspired by related work on pseudorandom generators for halfspaces [9]) and of Štefankovič et al. [13] (who approximated a "dual" recursion $S^*(i,j)$ defined as the smallest capacity c such that there exist at least j subsets of $\{w_1, \ldots, w_i\}$ with weight c).

Approximately counting integer knapsack solutions. In the more general integer version of #Knapsack, the subsets are multisets. That is, in addition to $W = \{w_1, \ldots, w_n\}$ we are also given a set $\{u_1, \ldots, u_n\}$ and we are allowed to take up to u_i items of weight w_i.

The first (randomized) FPRAS for counting integer knapsack solution was given by Dyer [3] who presented a strongly polynomial $O(n^5 + n^4\varepsilon^{-2})$ time algorithm. A (deterministic) FPTAS for this problem was then given by Gopalan et al. [6] with a running time of $O(n^5\varepsilon^{-2}\log^2 U \log w)$ (see also [5]) where $U = \max_i\{u_i\}$ and $w = \sum_i w_i u_i + C$. The fastest solution to date is by Halman [7] with a running time of $O(n^3\varepsilon^{-1}\log(n\varepsilon^{-1}\log U)\log^2 U)$.

Our results. In this paper we present improved algorithms for both #Knapsack and its integer version. Our algorithms improve the previous best algorithms by polynomial factors. For constant ε, we close the gap between the $\tilde{O}(n^{2.5})$ randomized and the $\tilde{O}(n^3)$ deterministic running times. More formally, with the standard assumption of constant time arithmetics on the input numbers, we prove the following two theorems:

▶ **Theorem 1.** *There is a FPTAS running in $O(n^{2.5}\varepsilon^{-1.5}\log(n\varepsilon^{-1})\log(n\varepsilon))$ time and $O(n^{1.5}\varepsilon^{-1.5})$ space for counting knapsack solutions.*

▶ **Theorem 2.** *There is a FPTAS running in $O(n^{2.5}\varepsilon^{-1.5}\log(n\varepsilon^{-1}\log U)\log(n\varepsilon)\log^2 U)$ time and $O(n^{1.5}\varepsilon^{-1.5}\log U)$ space for counting integer knapsack solutions.*

Our algorithm is the first algorithm to deviate from the standard recursion. In particular, on large enough sets, instead of recursing on all but the last item, we recurse in the middle and use convolution to merge the two sub-solutions. This requires extending the recent technique of *K-approximation sets and functions* used by Halman [7] and introduced in [8].

Our extended technique (which we call *sum approximations*) is simple to state and leads to a surprisingly simple solution to #Knapsack with an improved running time. In a nutshell, for any function $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ (think of $f(x) =$ the number of subsets with total weight exactly x) let $f^{\leq}$ denote the function $f^{\leq}(x) = \sum_{y \leq x} f(y)$ (hence $f^{\leq}(x) =$ the number of subsets with total weight at most x). Then, in order to approximate the function $f^{\leq}$ it is enough to find any function F such that $F^{\leq}$ approximates $f^{\leq}$.

We examine the properties of such sum approximations F in Section 2, and introduce a number of useful computational primitives on sum approximations. With these primitives in hand, we give a simplified version of Halman's algorithm for #Knapsack in Section 3. Then, in Section 4 we present an improved divide and conquer algorithm based on convolutions of sum approximations. Finally, in Section 5 we adapt our algorithm to the integer version, where every item has a corresponding multiplicity. Instead of the *binding constraints* approach used by Halman [7], we show that it is enough to perform a single scan of a sum approximation using nothing more than a standard binary search tree.

2 Approximation of a Function

Consider the following two functions: $f(x) =$ the number of subsets with total weight exactly x, and $f^{\leq}(x) =$ the number of subsets with total weight at most x. More generally:

▶ **Definition 3.** Given a function $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ we define the function $f^{\leq}(x)$ as

$$f^{\leq}(x) = \sum_{y \leq x} f(y).$$

Our goal is to approximate $f^{\leq}(C)$ but we will actually approximate the entire function $f^{\leq}(x)$ for all x. We now describe what it means to approximate a function and present some properties of such approximations.

▶ **Definition 4** ($(1+\varepsilon)$-approximation of a function)**.** Given a function $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ and a parameter $\varepsilon > 0$, a function $F : \mathbb{Z}^+ \to \mathbb{Z}^+$ is a $(1+\varepsilon)$-*approximation of* f if for every x,

$$f(x) \leq F(x) \leq (1+\varepsilon) f(x).$$

The above definition is similar to the definition of K-approximation sets [7] for $K = (1+\varepsilon)$.

▶ **Definition 5** ($(1+\varepsilon)$-sum approximation of a function)**.** Given a function $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ and a parameter $\varepsilon > 0$, a function $F : \mathbb{Z}^+ \to \mathbb{Z}^+$ is a $(1+\varepsilon)$-*sum approximation of* f if $F^{\leq}$ is a $(1+\varepsilon)$-approximation of $f^{\leq}$.

We next examine some useful properties of sum approximations. For a function $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ define its *shift* by w as follows:

$$f|_w(x) = \begin{cases} f(x-w), & x \geq w, \\ 0 & x < w, \end{cases}$$

and for two functions $f, g : \mathbb{Z}^+ \to \mathbb{Z}^+$ define their *convolution* to be:

$$(f * g)(w) = \sum_{x+y=w} f(x) g(y).$$

The following lemma describes four operations on sum approximations. The first three are similar to the ones used in [7, Property 2.1]. The fourth operation (convolution) is novel.

▶ **Lemma 6** (operations on sum approximations). *Let F be a $(1+\varepsilon)$-sum approximation of f and G be a $(1+\varepsilon)$-sum approximation of g, then the following properties hold:*

Approximation: *A $(1+\delta)$-sum approximation of F is a $(1+\delta)(1+\varepsilon)$-sum approximation of f.*

Summation: *$(F+G)$ is a $(1+\varepsilon)$-sum approximation of $(f+g)$.*

Shifting: *$F|_w$ is a $(1+\varepsilon)$-sum approximation of $f|_w$ for any $w>0$.*

Convolution: *$(F*G)$ is a $(1+\varepsilon)^2$-sum approximation of $(f*g)$.*

Proof.

Approximation: Let F' be a $(1+\delta)$-approximation of F. For every x, $f^{\leq}(x) \leq F^{\leq}(x) \leq (1+\varepsilon)f^{\leq}(x)$ and $F^{\leq}(x) \leq F'^{\leq}(x) \leq (1+\delta)F^{\leq}(x)$. We therefore have that $f^{\leq}(x) \leq F'^{\leq}(x) \leq (1+\delta)(1+\varepsilon)f^{\leq}(x)$.

Summation: For every x we have that $f^{\leq}(x) \leq F^{\leq}(x) \leq (1+\varepsilon)f^{\leq}(x)$ and $g^{\leq}(x) \leq G^{\leq}(x) \leq (1+\varepsilon)g^{\leq}(x)$, adding these two equations we get $(f+g)^{\leq}(x) \leq (F+G)^{\leq}(x) \leq (1+\varepsilon)(f+g)^{\leq}(x)$.

Shifting: For $x<w$, $f|_w(x) = 0 = F|_w(x)$. For $x \geq w$ let $y = x-w$. Since $y \geq 0$ we have that $f^{\leq}(y) \leq F^{\leq}(y) \leq (1+\varepsilon)f^{\leq}(y)$ and therefore $f|_w^{\leq}(x) \leq F|_w^{\leq}(x) \leq (1+\varepsilon)f|_w^{\leq}(x)$.

Convolution: We first prove that $(F*G)^{\leq}(w) \geq (f*g)^{\leq}(w)$:

$$\begin{aligned}
(F*G)^{\leq}(w) &= \sum_{x+y\leq w} F(x)G(y) = \sum_{x\leq w}\sum_{y\leq w-x} F(x)G(y) = \sum_{x\leq w}\left[F(x)\sum_{y\leq w-x} G(y)\right] \\
&= \sum_{x\leq w} F(x)G^{\leq}(w-x) \geq \sum_{x\leq w} F(x)g^{\leq}(w-x) = \sum_{x\leq w} F(x)\sum_{y\leq w-x} g(y) \\
&= \sum_{x+y\leq w} F(x)g(y) = \sum_{y\leq w}\sum_{x\leq w-y} F(x)g(y) = \sum_{y\leq w}\left[g(y)\sum_{x\leq w-y} F(x)\right] \\
&= \sum_{y\leq w} g(y)F^{\leq}(w-y) \geq \sum_{y\leq w} g(y)f^{\leq}(w-y) = \sum_{y\leq w} g(y)\sum_{x\leq w-y} f(x) \\
&= \sum_{x+y\leq w} f(x)g(y) = (f*g)^{\leq}(w).
\end{aligned}$$

Next we prove that $(F*G)^{\leq}(w) \leq (1+\varepsilon)^2(f*g)^{\leq}(w)$:

$$\begin{aligned}
(F*G)^{\leq}(w) &= \sum_{x+y\leq w} F(x)G(y) = \sum_{x\leq w}\sum_{y\leq w-x} F(x)G(y) = \sum_{x\leq w}\left[F(x)\sum_{y\leq w-x} G(y)\right] \\
&= \sum_{x\leq w} F(x)G^{\leq}(w-x) \leq \sum_{x\leq w} F(x)(1+\varepsilon)g^{\leq}(w-x) = \\
&= (1+\varepsilon)\sum_{x\leq w} F(x)\sum_{y\leq w-x} g(y) = (1+\varepsilon)\sum_{x+y\leq w} F(x)g(y) \\
&= (1+\varepsilon)\sum_{\leq w}\sum_{x\leq w-y} F(x)g(y) = (1+\varepsilon)\sum_{y\leq w}\left[g(y)\sum_{x\leq w-y} F(x)\right] \\
&= (1+\varepsilon)\sum_{y\leq w} g(y)F^{\leq}(w-y) \leq (1+\varepsilon)\sum_{y\leq w} g(y)(1+\varepsilon)f^{\leq}(w-y) = \\
&= (1+\varepsilon)^2\sum_{y\leq w} g(y)\sum_{x\leq w-y} f(x) = (1+\varepsilon)^2\sum_{x+y\leq w} f(x)g(y) \\
&= (1+\varepsilon)^2(f*g)^{\leq}(w).
\end{aligned}$$

◀

We next describe the way that we represent functions.

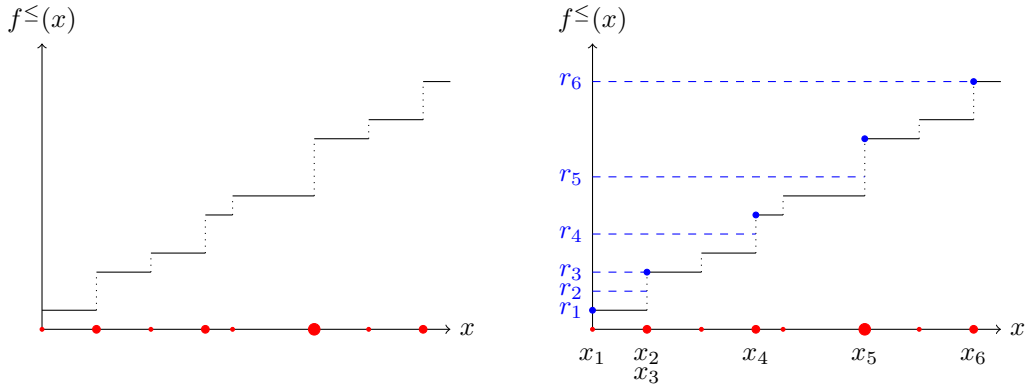


Figure 1 On the left, $f^{\leq}(x)$ compared to $f(x)$. The red point at position x is wider as $f(x)$ is larger. On the right, the blue points are the first entries that have value of at least r_i.

▶ **Definition 7** (a function representation)**.** Given a function $f : \mathbb{Z}^+ \to \mathbb{Z}^+$, the *representation of* f is defined to be a list of all the pairs $(x, f(x))$ where $f(x) > 0$. The list is kept sorted by the x value. The *size* of f (denoted by $|f|$) is the number of pairs in the representation of f. To simplify our presentation, we allow the representation to include multiple pairs with the same value of x. This can be easily fixed with a single scan over the representation.

In the following paragraphs we show how to efficiently implement the following operations on functions: *sparsification*, *summation*, *shifting*, *convolution*, and *query*. The output of each operation is a sum approximation.

Sparsification. Sparsification is the operation of constructing a $(1+\delta)$-sum approximation of f (see Definition 5). The input is a function $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ and a sparsification parameter $\delta > 0$, the output is a function $F : \mathbb{Z}^+ \to \mathbb{Z}^+$ that is a $(1+\delta)$-sum approximation of f. The goal is to construct a function F that has a compact representation (i.e. a small number of points with non-zero values). The general idea is based on the one in [7] (function *Compress*) but tailored towards our particular application. We partition the values of $f^{\leq}$ into segments with elements belonging to $[r_i, r_{i+1})$ (see Figure 1), where:

$$r_0 = 0,$$
$$r_{i+1} = \max\{r_i + 1, \lfloor (1+\delta) r_i \rfloor\}.$$

We call $x_i = \min_x \left\{ f^{\leq}(x) \geq r_i \right\}$ the i-th *breakpoint*. For any x, let $\mathrm{succ}(x)$ be the strict successor of x among $\{x_i\}$, i.e. $\mathrm{succ}(x) = \min_i \{x_i > x\}$ We define the function $\tilde{f^{\leq}}$ (see Figure 2) as:

$$\tilde{f^{\leq}}(x) = f^{\leq}(\mathrm{succ}(x) - 1),$$

where $\tilde{f^{\leq}}(x) = \lim_{x \to \inf} f^{\leq}(x)$ if $\mathrm{succ}(x) = \infty$.

▶ **Lemma 8.** *$\tilde{f^{\leq}}$ is a $(1+\delta)$-approximation of $f^{\leq}$.*

Proof. First observe that $f^{\leq}(x) \leq \tilde{f^{\leq}}(x)$ (since $\mathrm{succ}(x) > x$ and $f^{\leq}$ is monotone). Consider any x and let i be the unique index such that $r_i \leq f^{\leq}(x) < r_{i+1}$. If $\mathrm{succ}(x) = \infty$ then $\tilde{f^{\leq}}(x) = \lim_{x \to \inf} f^{\leq}(x) < r_{i+1}$. Otherwise, $x_{i+1} > x$ and $\tilde{f^{\leq}}(x) = f^{\leq}(x_{i+1} - 1) < r_{i+1}$. We need to consider two cases: If $r_{i+1} \leq (1+\delta) r_i$, then $\tilde{f^{\leq}}(x) < r_{i+1} \leq (1+\delta) f^{\leq}(x)$. If $r_{i+1} = r_i + 1$ and because the values of $\tilde{f^{\leq}}(x)$ are integer, $\tilde{f^{\leq}}(x) \leq r_{i+1} - 1 = r_i$. So in both cases $\tilde{f^{\leq}}(x) \leq (1+\delta) f^{\leq}(x)$. ◀

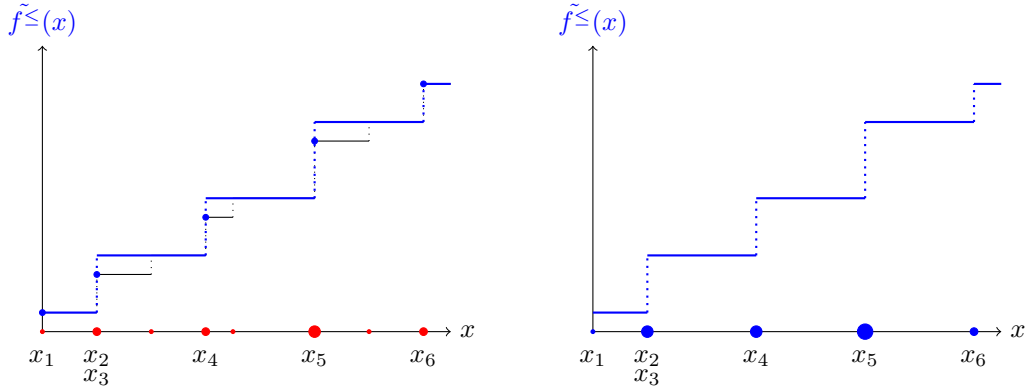


Figure 2 On the left, $\tilde{f^{\leq}}$ (in blue) is defined from $f^{\leq}$ and has the same value in any segment $[x_i, x_{i+1})$. On the right, the construction of $F(x)$. The blue points are only at positions x_i and are wider as $F(x_i)$ is larger.

Algorithm 1 SPARSIFY(f, δ).

Input: a function f represented by a sorted list of all pairs $(x, f(x))$ where $f(x) > 0$ and a sparsification parameter $\delta > 0$.

Output: a function F that is a $(1+\delta)$-sum approximation of f and is represented by a sorted list of at most $\log_{1+\delta} M$ pairs (where M is the maximum value of $f^{\leq}$).

initialize $r = accum = prevaccum = prevx = 0$
for every pair $(x, f(x))$ in sorted order **do**
  $r \leftarrow \max\{r+1, \lfloor(1+\delta)r\rfloor\}$
  while $accum < r$ **do**
    $accum \leftarrow accum + f(x)$
    get the next pair $(x, f(x))$ in the list
  end while
  add the pair $(prevx, accum - f(x) - prevaccum)$ to F
  $prevaccum \leftarrow accum - f(x)$
  $prevx \leftarrow x$
end for

We can now define the function F (the $(1+\delta)$-sum approximation of f). Observe that by Lemma 8, every F such that $F^{\leq} = \tilde{f^{\leq}}$ is a $(1+\delta)$-sum approximation. We define F as the discrete derivative of $\tilde{f^{\leq}}$. That is,

$$F(x) = \begin{cases} \tilde{f^{\leq}}(x) - \tilde{f^{\leq}}(x-1) & x > 0, \\ \tilde{f^{\leq}}(x) & x = 0. \end{cases}$$

It is easy to see that $F^{\leq} = \tilde{f^{\leq}}$. It is also easy to construct the representation of F in linear time with a single scan over the representation of f (see Algorithm 1). Let M be the maximum value of $f^{\leq}$. Notice that $\tilde{f^{\leq}}$ can have at most $|\{x_i\}| = |\{r_i\}| = \log_{1+\delta} M$ different values. This means that $|F|$ is at most $\log_{1+\delta} M$. The total running time is therefore $O(|f| + \log_{1+\delta} M)$.

Summation. Given two $(1+\varepsilon)$-sum approximations F and G of functions f and g respectively, we wish to construct the function $F + G$ (that is a $(1+\varepsilon)$-sum approximation of $f + g$

by Lemma 6). We construct $F+G$ naively by setting $(F+G)(x) = F(x)+G(x)$. The sorted list of $F+G$ can be obtained in linear time given two sorted lists of F and of G. The total space and time is therefore $O(|F|+|G|)$.

Shifting. Given a $(1+\varepsilon)$-sum approximation F of f, the function shifted by w, $F|_w$ is a $(1+\varepsilon)$-sum approximation of $f|_w$ by Lemma 6. In order to create $F|_w$, we take every pair in the representation of F, namely $(x, y = F(x))$ and change it to $(x+w, y)$. $F|_w(x)$ will be the sum of all the pairs where the first coordinate is x. The total space and time is $O(|F|)$.

Convolution. The convolution $F * G$ contains all the combinations of taking some x value from F and some y value from G. For every pair x, y such that $F(x) \neq 0$ and $G(x) \neq 0$ we add the value $F(x)G(y)$ to the value of $F * G$ at point $x+y$.

We sort these pairs in order to get the representation of $F * G$. For a certain y we have all the points $(x+y, F(x))$ sorted already, those are $|G|$ sorted sequences that we have to merge. The total space of the output $F * G$ is at most the number of such pairs x, y, that is $|F| \cdot |G|$. But it is possible to obtain a stream of the sorted pairs with their value using less space, by using a heap to merge the lists. Assuming without loss of generality that $|G| \leq |F|$, each list is a value y and a pointer to a point in F, and the heap extracts the minimum value of the sum of y and the value in the pointer. The total time to create $F * G$ is therefore $O(|F| \cdot |G| \cdot \log(\min\{|F|, |G|\}))$ and the space $O(|F|+|G|)$.

Query. Given a $(1+\varepsilon)$-sum approximation F of f and a point x, we can query the value $F^{\leq}(x)$ that satisfies $f^{\leq}(x) \leq F^{\leq}(x) \leq (1+\varepsilon)f^{\leq}(x)$ in time $O(|F|)$. This is because computing the function $F^{\leq}(x) = \sum_{y \leq x} F(y)$ takes $O(|F|)$ time by considering every y. Moreover, if we store the representation of F in a balanced binary search tree T then a query can be done in $O(\log |F|)$ time with a prefix sum query on T.

3 The Algorithm of Halman [7] (Simplified)

In this section we present a simplified version of the algorithm of Halman [7] for #Knapsack using sum approximations. The running time of this simple deterministic algorithm is $O(n^3\varepsilon^{-1})$ and the space is $O(n^2\varepsilon^{-1})$.

For a set of weights S, let $k_S(x)$ denote the number of subsets of S with total weight exactly x. The output of the algorithm is the function K_W that is a $(1+\varepsilon)$-sum approximation of k_W. The desired answer, $K_W^{\leq}(C)$, can then be easily obtained using the query operation.

Recall that $k_S|_w(x) = k_S(x-w)$ if $x \geq w$ and 0 otherwise. The algorithm is based on the following observation:

▶ **Lemma 9.** *Let S be a set of integer weights and w be an additional integer weight, then:*

$$k_{S\cup\{w\}} = k_S + k_S|_w$$

Proof. Any subset of $S \cup \{w\}$ with weight x either includes w (the number of such solutions is $k_S(x-w)$) or does not include w (the number of such solutions is $k_S(x)$). Since these options are disjoint, we have that $k_{S\cup\{w\}}(x) = k_S(x) + k_S|_w(x)$. ◀

The algorithm. The algorithm uses the above lemma to construct the set S by inserting one element at a time (until $S = W$), keeping K_S updated. The algorithm starts by setting $K_\emptyset(0) = 1$ and $K_\emptyset(x) = 0$ for any $x \neq 0$. In the i-th step, we compute the function

$K_{\{w_1,\dots,w_i\}}$ from $K_{\{w_1,\dots,w_{i-1}\}}$. Computing $K_{\{w_1,\dots,w_i\}}$ can be done with one shifting and one summation operation: $K_{\{w_1,\dots,w_i\}} = K_{\{w_1,\dots,w_{i-1}\}} + K_{\{w_1,\dots,w_{i-1}\}}|_{w_i}$. Notice that the size $|K_{\{w_1,\dots,w_i\}}| = 2|K_{\{w_1,\dots,w_{i-1}\}}|$ doubles from the summation operation. To overcome this blowup, at the end of each step of the algorithm, we sparsify with parameter $\delta = (1+\varepsilon)^{\frac{1}{n}} - 1$.

Correctness. From Lemmas 6 and 9, it follows that if K_S is a $(1+\alpha)$-sum approximation of k_S, then $K_S + K_S|_w$ is a $(1+\alpha)$-sum approximation of $k_{S\cup\{w\}}$. Furthermore, the approximation factor of $K_{\{w_1,\dots,w_i\}}$ after the sparsification is the approximation factor of $K_{\{w_1,\dots,w_{i-1}\}}$ multiplied by $(1+\delta)$. We get that K_W is a $((1+\varepsilon)^{\frac{1}{n}})^n$-sum approximation of k_W, as required.

Time complexity. The size of $K_{\{w_1,\dots,w_i\}}$ after the sparsification is bounded by $\log_{1+\delta} 2^i$. The time complexity is therefore:

$$\sum_{i=1}^{n} O\big(\log_{1+\delta} 2^i\big) = O\left(\sum_{i=1}^{n} \frac{i}{\log(1+\delta)}\right) = O\left(\frac{1}{\frac{1}{n}\log(1+\varepsilon)} \sum_{i=1}^{n} i\right) = O\big(n^3\varepsilon^{-1}\big)\,.$$

Space complexity. The space is $O(|K_W|) = O(\log_{1+\delta} 2^n) = O\Big(\frac{n}{\log(1+\delta)}\Big) = O\Big(\frac{n^2}{\log(1+\varepsilon)}\Big) = O(n^2\varepsilon^{-1})$, where we have used that $\ln(1+\varepsilon) \geq \varepsilon/2$ for $\varepsilon \in (0,1)$.

4 The Algorithm for Counting Knapsack Solutions

In this section we present a deterministic $O(n^{2.5}\varepsilon^{-1.5}\log(n\varepsilon^{-1})\log(n\varepsilon))$ time $O(n^{1.5}\varepsilon^{-1.5})$ space algorithm for counting knapsack solutions. The algorithm is based upon a similar observation to the one in Lemma 9:

▶ **Lemma 10.** *Let S and T be two sets of integer weights, then:*

$$k_{S\cup T} = k_S * k_T$$

Proof. A subset of $S\cup T$ of weight w must be obtained by taking a subset of weight w_S from S and a subset of weight w_T from T where $w_S + w_T = w$. Thus, $k_S(w_S)$ subsets of S of weight w_S and $k_T(w_T)$ subsets of T weight w_T generate $k_S(w_S)k_T(w_T)$ subsets of $S\cup T$ of weight $w_S + w_T$. Overall, we get that $k_{S\cup T}(w) = \sum_{w_S+w_T=w} k_S(w_S)k_T(w_T) = (k_S * k_T)(w)$. ◀

The algorithm. As in Section 3, our algorithm computes a $(1+\varepsilon)$-sum approximation K_W of k_W recursively. This time however, we do two things differently: (1) The value of the approximation factor is different for each recursion depth. In a recursive call of depth i, we are given a set S and we compute a $(1+\varepsilon_i)$-sum approximation K_S of k_S for some ε_i to be chosen later. (2) Given a set S we recurse differently depending on the size of S:

1. If $|S| > \sqrt{n/\varepsilon}$, then we partition the set S into two sets A and B each of size $|S|/2 = n/2^i$ and make two recursive calls: One computes a $(1+\varepsilon_{i+1})$-sum approximation K_A of k_A and the other computes a $(1+\varepsilon_{i+1})$-sum approximation K_B of k_B. We then find K_S by computing the convolution $K_A * K_B$ and sparsifying with parameter

$$\delta_i = \frac{\varepsilon^{3/4}}{2c\cdot 2^{i/2}\cdot n^{1/4}},$$

where ε is the original approximation parameter and $c = \frac{\sqrt{2}}{\sqrt{2}-1}$.

2. If $|S| \leq \sqrt{n/\varepsilon}$, then we apply the algorithm from Section 3 on the set S with parameter $\delta_{\log(\sqrt{n\varepsilon})} = \Omega(\sqrt{\varepsilon/n})$. Observe that in such a case the recursion depth is at least $\log(\sqrt{n\varepsilon})$.

Correctness. From Lemmas 6 and 10, it follows that if K_A is a $(1+\varepsilon_{i+1})$-sum approximation of k_A and K_B is a $(1+\varepsilon_{i+1})$-sum approximation of k_B, then K_S is a $(1+\varepsilon_{i+1})^2(1+\delta_i)$-sum approximation of k_S. This means that ε_i satisfies the following relation:

$$(1+\varepsilon_i) = (1+\varepsilon_{i+1})^2(1+\delta_i)$$

From the above equation and since on the bottom of the recursion with $\delta_i = \delta_{\log(\sqrt{n\varepsilon})}$, the final approximation factor of K_W is:

$$(1+\varepsilon_0) = (1+\delta_{\log(\sqrt{n\varepsilon})})^{\sqrt{n\varepsilon}} \prod_{i=0}^{\log(\sqrt{n\varepsilon})-1} (1+\delta_i)^{2^i} = \prod_{i=0}^{\log(\sqrt{n\varepsilon})} (1+\delta_i)^{2^i}$$

We need to prove that the above product is not larger than $(1+\varepsilon)$. Since $\sum_{i=0}^{x} 2^{i/2} < \frac{\sqrt{2}}{\sqrt{2}-1} \cdot 2^{x/2} = c \cdot 2^{x/2}$ and $(1+\delta_i)^{2^i} = (1+\delta_i)^{1/\delta_i \cdot 2^i \delta_i} \leq e^{2^i\delta_i} = e^{\frac{\varepsilon^{3/4}}{2c} n^{-1/4} 2^{i/2}}$ we obtain:

$$\prod_{i=0}^{\log(\sqrt{n\varepsilon})} (1+\delta_i)^{2^i} \leq e^{\frac{\varepsilon^{3/4}}{2c} n^{-1/2} \sum_{i=0}^{\log(\sqrt{n\varepsilon})} 2^{i/2}} < e^{\frac{\varepsilon^{3/4}}{2} n^{-1/4} 2^{\log(\sqrt{n\varepsilon})/2}} = e^{\frac{\varepsilon}{2}} \leq (1+\varepsilon),$$

where the last inequality follows from $\ln(1+\varepsilon) \geq \varepsilon/2$ for $\varepsilon \in (0,1)$. Moreover, since the recursion changes at depth $\log(\sqrt{n\varepsilon})$ we further need to assume that $\varepsilon \geq 1/n$. These assumptions are without loss of generality since for $\varepsilon > 1$ we could simply use $\varepsilon = 1$ and for $\varepsilon < 1/n$ the previous algorithms are faster.

Time complexity. We analyze the time complexity of every recursion depth i. For depth $i = \log(\sqrt{n\varepsilon})$, we apply the simple algorithm from Section 3 $\lfloor\sqrt{n\varepsilon}\rfloor$ times on sets of size $\Theta(\sqrt{n/\varepsilon})$ with $\delta_{\log(\sqrt{n\varepsilon})} = \Omega(\sqrt{\varepsilon/n})$, the running time is therefore $O(n^{2.5}\varepsilon^{-1.5})$.

For depth $i < \log(\sqrt{n\varepsilon})$, we apply 2^i convolutions and sparsifications. The time of the convolutions is dominant. The total running time is therefore:

$$\begin{aligned}\sum_{i=0}^{(\log\sqrt{n\varepsilon})-1} 2^i \left(\frac{n}{2^{i+1}\cdot\delta_{i+1}}\right)^2 \log\left(\frac{n}{2^{i+1}\cdot\delta_{i+1}}\right) &\leq \frac{1}{2}\sum_{i=1}^{\log\sqrt{n\varepsilon}} 2^i \left(\frac{n}{2^i\cdot\delta_i}\right)^2 \log(n\varepsilon^{-1}) \\ &= O\left(\sum_{i=1}^{\log\sqrt{n\varepsilon}} \frac{n^{2.5}}{\varepsilon^{1.5}} \log(n\varepsilon^{-1})\right) \\ &= O(n^{2.5}\varepsilon^{-1.5}\log(n\varepsilon^{-1})\log(n\varepsilon)).\end{aligned}$$

Space complexity. Since each recursive call makes at most two recursive calls, we do not need to keep more than two representation of sum approximations on every level of the recursion. We have seen in Section 2 that it is possible to construct sparsification of convolution in linear space. The space complexity of all recursive calls of depth $i < \log(\sqrt{n\varepsilon})$ is therefore:

$$\sum_{i=0}^{\log(\sqrt{n\varepsilon})} \left(2\cdot\frac{n}{2^i\cdot\delta_i}\right) = O\left(\sum_{i=0}^{\log(\sqrt{n\varepsilon})} \frac{n^{1.25}}{2^{i/2}\cdot\varepsilon^{0.75}}\right) = O\left(n^{1.25}\varepsilon^{-0.75}\right)$$

One call to the algorithm from Section 3 uses $O\left(\frac{\left(\sqrt{n/\varepsilon}\right)^2}{\delta_{\log(\sqrt{n\varepsilon})}}\right) = O\left(\frac{\left(\sqrt{n/\varepsilon}\right)^2}{\sqrt{\varepsilon/n}}\right) = O(n^{1.5}\varepsilon^{-1.5})$ space, and therefore the total space complexity is $O(n^{1.5}\varepsilon^{-1.5})$.

5 The Algorithm for Counting Integer Knapsack Solutions

In this section we show how to generalize the algorithms of Sections 3 and 4 to the integer version of counting knapsack solutions.

5.1 Generalizing the algorithm of Section 3

In Section 3 we showed how to insert into a set S a single item with weight w. We now need to show how to insert a single item with weight w and multiplicity u. The proof of the following lemma is similar to that of Lemma 9.

▶ **Lemma 11.** *Let S be a set of integer pairs representing weights and multiplicity of items, and let (w, u) be the weight and multiplicity of an additional item, then:*

$$k_{S\cup\{(w,u)\}} = k_S + k_S|_w + k_S|_{2w} + \ldots + k_S|_{u\cdot w}$$

We will describe a new operation on sum approximations that creates the sparsification of $G = (K_S + K_S|_w + K_S|_{2w} + \ldots + K_S|_{u\cdot w})$ without actually computing G, i.e. without actually computing all the points with non-zero value.

Events. Observe that a point (x, y) with non-zero value $y = K_S(x)$ implies $u + 1$ points in the above sum: $(x, y), (x + w, y), (x + 2w, y), \ldots, (x + u \cdot w, y)$. We call the first point (x, y) a *start event* (with position x and value y) and the last point $(x + u \cdot w, y)$ an *end event* (with position $x + u \cdot w$ and value y). Overall, for every x with non-zero value $y = K_S(x)$ there are two events, a total of $2|K_S|$ events. It is possible to sort in linear time the sequence of events by their positions $\{x\}_i \cup \{x + u \cdot w\}_i$ because K_S is given sorted. We call the sorted list of events the *event list*.

Similarly to Algorithm 1, we could construct the sparsification of $G = (K_S + K_S|_w + K_S|_{2w} + \ldots + K_S|_{u\cdot w})$ by scanning all points in G. This however would be too costly. Instead, we next present a new operation that constructs the sparsification of G while only scanning the $O(|K_S|)$ events in the event list.

InsertAndSparsify. While scanning the event list, when we see a start event (x, y) then we say that this event is an *active event* and that all the points $(x, y), (x + w, y), (x + 2w, y), \ldots, (x + u \cdot w, y)$ are *active points*. These points will become inactive when we will see the end event $(x + u \cdot w, y)$. As in Algorithm 1, we would like to accumulate the values of all points seen so far. When the accumulator is larger than r, we introduce a new breakpoint (i.e., output a new point to the sparsification of G) and set the new r to be $\max\{r + 1, \lfloor(1 + \delta)r\rfloor\}$. During the scan, apart from the accumulator, we also maintain a value Y = the sum of values of all currently active events.

When we scan a start event (x, y), we add the value y to both the accumulator and to Y. This is not enough. We also need to add to the accumulator the values of all active points whose position is x. These are precisely the points whose start events were at positions $x - w, x - 2w, \ldots, x - u \cdot w$. Notice that all these points have the same start position modulo w. For this reason, we maintain a balanced binary search tree T that stores all active events keyed by their start position modulo w. Each node v in T with key $r \in \{0, \ldots, w - 1\}$ stores a field Y_v = the sum of values of all active points whose start position is x such that $x = r \pmod w$. When we see a start event (x, y), we search in T for the node v with key $x \bmod w$ (or create one if no such node exists), we increase the node's Y_v field by y, and increase the accumulator by y. If (x, y) is an end event we subtract y from Y_v and from Y.

After processing event (x, y) as explained above, we want to process the next event (x', y') in the event list. Before doing so, we need to: (1) increase the accumulator by the total value of all points in the segment $[x, x')$, and (2) if the updated accumulator is larger than r, find and output the (possibly many) new breakpoints whose positions are between x and x'.

(1) We partition the segment $[x, x')$ into three segments: $[x, k_1w), [k_1w, k_2w), [k_2w, x')$ such that the lengths of the first and last segments are smaller than w, and the length of the middle segment is a multiple of w. Notice that every segment of length w contains an active point from every active event exactly once. Therefore, to obtain the total value of points in the segment $[x, x')$ we query T for the total value in segment $[x, k_1w)$ (with a *suffix sum* query on the Y_v values) and in segment $[k_2w, x')$ (with a *prefix sum* query on the Y_v values), and add to it $Y \cdot (k_2 - k_1)$ (the total value in segment $[k_1w, k_2w)$).

(2) After increasing the accumulator by the above total value, if the accumulator becomes larger than r, then we will find all the new breakpoints in $[x, x')$ in $O(\log |K_S|)$ time per breakpoint. Suppose the accumulated value at x was *prevaccum*. To find the first breakpoint in the segment $[x, k_1w)$ we query T for the the first node after $x \bmod w$ such that the sum of Y_v values between $x \bmod w$ and that node is at least $r - prevaccum$ (we call this a *succeeding sum* query on T, and we symmetrically define a *preceding sum* query in which we seek the first node before $x \bmod w$ rather than after). We then output this breakpoint, set *prevaccum* to be the accumulated value in this breakpoint, set $r = \max\{r + 1, \lfloor (1 + \delta)r \rfloor\}$, and continue in the same way to find the next breakpoint in the segment $[x, k_1w)$. Next, we find the breakpoints in the segment $[k_1w, k_2w)$. Since this segment is composed of $(k_2 - k_1)$ subsegments of length w, and since each of these subsegments contributes exactly Y to the accumulator, it is easy (in $O(1)$ time) to find which subsegment contains the next breakpoint. On this subsegment we proceed similarly as on $[x, k_1w)$. Finally, we need to find the breakpoints in segment $[k_2w, x')$, again similarly as in $[x, k_1w)$. See Algorithm 2.

Time and space complexity. Each operation on the binary search tree T takes $O(\log |K_S|)$ time thus the total time for InsertAndSparsify is $O\big((|K_{S\cup\{(w,u)\}}| + |K_S|) \cdot \log |K_S|\big)$. If $S = \{w_1, \ldots, w_i\}$ then $|K_S| = \log_{1+\delta} U^i$ and $|K_{S\cup\{(w,u)\}}| = O\big(\log_{1+\delta} U^i\big)$. The total time complexity of the algorithm is therefore:

$$\begin{aligned}
\sum_{i=1}^{n} O\big(\log_{1+\delta} U^i \cdot \log(\log_{1+\delta} U^i)\big) &= O\left(\sum_{i=1}^{n} \frac{i \log U}{\log(1+\delta)} \cdot \log\left(\frac{i \log U}{\log(1+\delta)}\right)\right) \\
&= O\left(\frac{\log U}{\frac{1}{n}\log(1+\varepsilon)} \cdot \log\left(\frac{n \log U}{\frac{1}{n}\log(1+\varepsilon)}\right) \sum_{i=1}^{n} i\right) \\
&= O\big(n^3 \varepsilon^{-1} \log U \log\left(n\varepsilon^{-1} \log U\right)\big)\,.
\end{aligned}$$

The space complexity is $O(|K_W|) = O\big(\log_{1+\delta} U^n\big) = O\Big(\frac{n \log U}{\log(1+\delta)}\Big) = O(n^2\varepsilon^{-1} \log U)$.

5.2 Generalizing the algorithm of Section 4

The only change to the algorithm of Section 4 is that when the set size is small (i.e. when we call the algorithm of Section 3) we use InsertAndSparsify as in the previous subsection.

Time and space complexity. We observe that the size of the representation of a $(1+\varepsilon)$-sum approximation has been changed to $|K_S| = \log_{1+\delta} U^{|S|} = \frac{|S| \log U}{\delta}$. As in Section 4, we

Algorithm 2 InsertAndSparsify(K_S, w, u, δ).

Input: a sum approximation K_S of k_S, a new item with weight of w and multiplicity u, and a sparsification parameter δ.

Output: a function G that is a $(1+\delta)$-sum approximation of $k_{S\cup(w,u)}$.

initialize T as an empty binary search tree of pairs (x, y) indexed by $(x \bmod w)$
initialize $Y = x = accum = prevaccum = 0$, and $r = 1$
for every event (x', y') in sorted order **do**
  $k_1 \leftarrow \lceil \frac{x}{w} \rceil$, $k_2 \leftarrow \lfloor \frac{x'}{w} \rfloor$
  while $accum + T.\mathsf{suffixSum}(x) + Y \cdot (k_2 - k_1) + T.\mathsf{prefixSum}(x' - 1) \geq r$ **do**
    if $accum + T.\mathsf{suffixSum}(x) \geq r$ **then**
      $bp \leftarrow T.\mathsf{succeedingSum}(x, r - accum)$
      $accum \leftarrow accum + T.\mathsf{suffixSum}(x) - T.\mathsf{suffixSum}(bp)$
    else
      $k_3 \leftarrow k_1 + \lfloor (r - accum - T.\mathsf{suffixSum}(x))/Y \rfloor$
      $bp \leftarrow T.\mathsf{precedingSum}(x', r - (accum + T.\mathsf{suffixSum}(x) + Y \cdot (k_3 - k_1)))$
      $accum \leftarrow accum + T.\mathsf{suffixSum}(x) + Y \cdot (k_3 - k_1) + T.\mathsf{prefixSum}(bp - 1)$
    end if
    add the pair $(x', accum - prevaccum - T[bp \bmod w])$ to G
    $prevaccum \leftarrow accum$, $x \leftarrow bp$, $k_1 \leftarrow \lceil \frac{x}{w} \rceil$, $r \leftarrow \max\{r+1, \lfloor (1+\delta)r \rfloor\}$
  end while
  if (x', y') is a start event **then**
    $Y \leftarrow Y + y'$, $T[x' \bmod w] \leftarrow T[x' \bmod w] + y'$
  else
    $Y \leftarrow Y - y'$, $T[x' \bmod w] \leftarrow T[x' \bmod w] - y'$
  end if
  $accum \leftarrow accum + T.\mathsf{suffixSum}(x) + Y \cdot (k_2 - k_1) + T.\mathsf{prefixSum}(x' - 1)$
  $x \leftarrow x'$
end for

calculate the total time complexity for depth $\log(\sqrt{n\varepsilon})$:

$$O\left(\sqrt{n\varepsilon} \cdot \left(\left(\sqrt{n/\varepsilon}\right)^3 \left(\sqrt{\varepsilon/n}\right)^{-1} \log U \log\left(\sqrt{n\varepsilon}\left(\sqrt{\varepsilon/n}\right)^{-1} \log U\right)\right)\right)$$

which is $O\big(n^{2.5}\varepsilon^{-1.5} \log U \log(n\varepsilon^{-1} \log U)\big)$. The total time for depths $i < \log(\sqrt{n\varepsilon})$ is:

$$\begin{aligned}
&\sum_{i=0}^{(\log\sqrt{n\varepsilon})-1} 2^i \left(\frac{n \log U}{2^{i+1} \cdot \delta_{i+1}}\right)^2 \log\left(\frac{n \log U}{2^{i+1} \cdot \delta_{i+1}}\right) \\
&\leq \frac{1}{2} \sum_{i=1}^{\log\sqrt{n\varepsilon}} 2^i \left(\frac{n \log U}{2^i \cdot \delta_i}\right)^2 \log(n\varepsilon^{-1} \log U) \\
&= O\left(\sum_{i=1}^{\log\sqrt{n\varepsilon}} \frac{n^{2.5} \log^2 U}{\varepsilon^{1.5}} \log(n\varepsilon^{-1} \log U)\right) \\
&= O(n^{2.5}\varepsilon^{-1.5} \log(n\varepsilon^{-1} \log U) \log(n\varepsilon) \log^2 U).
\end{aligned}$$

The space complexity is dominated by the space used for InsertAndSparsify on a set of size $\sqrt{n/\varepsilon}$ and $\delta = \sqrt{\varepsilon/n}$, that is $O(n^{1.5}\varepsilon^{-1.5} \log U)$.

References

1 Mohsen Bayati, David Gamarnik, Dimitriy Katz, Chandra Nair, and Prasad Tetali. Simple deterministic approximation algorithms for counting matchings. In *STOC*, pages 122–127, 2007.

2 Ilias Diakonikolas, Parikshit Gopalan, Ragesh Jaiswal, Rocco A. Servedio, and Emanuele Viola. Bounded independence fools halfspaces. In *FOCS*, pages 171–180, 2009.

3 Martin Dyer. Approximate counting by dynamic programming. In *STOC*, pages 693–699, 2003.

4 Martin Dyer, Alan Frieze, Ravi Kannan, Ajai Kapoor, Ljubomir Perkovic, and Umesh Vazirani. A mildly exponential time algorithm for approximating the number of solutions to a multidimensional knapsack problem. *Combinatorics, Probability and Computing*, 2:271–284, 1993.

5 Parikshit Gopalan, Adam Klivans, and Raghu Meka. Polynomial-time approximation schemes for knapsack and related counting problems using branching programs. *arXiv 1008.3187*, 2010.

6 Parikshit Gopalan, Adam Klivans, Raghu Meka, Daniel Štefankovic, Santosh Vempala, and Eric Vigoda. An FPTAS for #Knapsack and related counting problems. In *FOCS*, pages 817–826, 2011.

7 Nir Halman. A deterministic fully polynomial time approximation scheme for counting integer knapsack solutions made easy. *Theoretical Computer Science*, 645:41–47, 2016.

8 Nir Halman, Diego Klabjan, Mohamed Mostagir, Jim Orlin, and David Simchi-Levi. A fully polynomial-time approximation scheme for single-item stochastic inventory control with discrete demand. *Mathematics of Operations Research*, 34(3):674–685, 2009.

9 Raghu Meka and David Zuckerman. Pseudorandom generators for polynomial threshold functions. In *STOC*, pages 427–436, 2010.

10 Ben Morris and Alistair Sinclair. Random walks on truncated cubes and sampling 0-1 knapsack solutions. *SIAM journal on computing*, 34(1):195–226, 2004. Preliminary version in FOCS 1999.

11 Yuval Rabani and Amir Shpilka. Explicit construction of a small epsilon-net for linear threshold functions. In *STOC*, pages 649–658, 2009.

12 Romeo Rizzi and Alexandru I. Tomescu. Faster FPTASes for counting and random generation of knapsack solutions. In *ESA*, pages 762–773, 2014.

13 Daniel Štefankovič, Santosh Vempala, and Eric Vigoda. A deterministic polynomial-time approximation scheme for counting knapsack solutions. *SIAM Journal on Computing*, 41(2):356–366, 2012.

14 Dror Weitz. Counting independent sets up to the tree threshold. In *STOC*, pages 140–149, 2006.

Towards Optimal Approximate Streaming Pattern Matching by Matching Multiple Patterns in Multiple Streams*

Shay Golan
Bar Ilan University, Ramat Gan, Israel
golansh1@cs.biu.ac.il

Tsvi Kopelowitz
Bar Ilan University, Ramat Gan, Israel
kopelot@gmail.com

Ely Porat
Bar Ilan University, Ramat Gan, Israel
porately@cs.biu.ac.il

Abstract

Recently, there has been a growing focus in solving approximate pattern matching problems in the streaming model. Of particular interest are the pattern matching with k-mismatches (KMM) problem and the pattern matching with w-wildcards (PMWC) problem. Motivated by reductions from these problems in the streaming model to the *dictionary matching problem*, this paper focuses on designing algorithms for the dictionary matching problem in the *multi-stream model* where there are several independent streams of data (as opposed to just one in the streaming model), and the memory complexity of an algorithm is expressed using two quantities: (1) a read-only *shared memory* storage area which is shared among all the streams, and (2) local *stream memory* that each stream stores separately.

In the dictionary matching problem in the multi-stream model the goal is to preprocess a dictionary $D = \{P_1, P_2, \ldots, P_d\}$ of $d = |D|$ patterns (strings with maximum length m over alphabet Σ) into a data structure stored in shared memory, so that given multiple independent streaming texts (where characters arrive one at a time) the algorithm reports occurrences of patterns from D in each one of the texts as soon as they appear.

We design two efficient algorithms for the dictionary matching problem in the multi-stream model. The first algorithm works when all the patterns in D have the same length m and costs $O(d \log m)$ words in shared memory, $O(\log m \log d)$ words in stream memory, and $O(\log m)$ time per character. The second algorithm works for general D, but the time cost per character becomes $O(\log m + \log d \log\log d)$. We also demonstrate the usefulness of our first algorithm in solving both the KMM problem and PMWC problem in the streaming model. In particular, we obtain the first almost optimal (up to poly-log factors) algorithm for the PMWC problem in the streaming model. We also design a new algorithm for the KMM problem in the streaming model that, up to poly-log factors, has the same bounds as the most recent results that use different techniques. Moreover, for most inputs, our algorithm for KMM is significantly faster on average.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Streaming approximate pattern matching, Dictionary matching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.65

* This work is supported in part by ISF grant 1278/16. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 683064).

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 65; pp. 65:1–65:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the popular *streaming* model [2, 48] the input is given as a sequence of elements (the data stream) that may be scanned only once, the storage space is limited, and the amount of time spent on each element needs to be minimized. In many problems there is also a preprocessing phase involved. For example, in the basic streaming pattern matching problem, the goal is to find occurrences of a given pattern (to be preprocessed) of size m in the data stream [52, 18]. The preprocessing phase receives the pattern and creates a sub-linear sized data structure that is used to locate the pattern in streaming input. Following the breakthrough result of Porat and Porat [52], there has recently been a rising interest in solving pattern matching problems in the streaming model [18, 30, 47, 19, 42, 23, 24, 37, 38].

Approximate Streaming Pattern Matching. While Porat and Porat [52] and Breslauer and Galil [18] addressed the exact match case, which is the purest form of streaming pattern matching, several papers have focused on *approximate* versions in the streaming model. The term *approximate pattern matching* refers to any pattern matching problem that is not exact matching. Examples include pattern matching with up to k-mismatches [46, 53, 11, 22, 21, 27, 24, 26, 28], pattern matching with w-wildcards [33, 49, 41, 43, 29, 20, 37], pattern matching with up to e-edits [56], parameterized pattern matching [6, 12, 42, 15, 16, 17, 39], function pattern matching [13, 4], swapped matching [10, 3, 5] and many more.

Remarkably, recent results in the streaming model for both the pattern matching with k-mismatches (KMM) problem [24] and the pattern matching with w-wildcards (PMWC) problem [37] (both formally defined below) use a similar approach which, in particular, reduce the approximate pattern matching problem in the streaming model that is being solved to the *dictionary matching* (DM) problem in the streaming model. In the dictionary matching problem ([23, 31, 8, 9, 45, 34, 14, 35, 32, 7, 38]) the goal is to preprocess a *dictionary* $D = \{P_1, P_2, \ldots, P_d\}$ of $d = |D|$ patterns (strings over alphabet Σ) so that given a text T we quickly report all of the occurrences of patterns from D in T. In the streaming model [23, 38] the text T arrives online, one character at a time, and the goal is to report, for each arriving character, the id of the longest pattern ending at this character[1]. Moreover, a pattern must be reported as soon as it appears.

The motivation for this paper is due to realizing that the reductions to the dictionary matching problem mentioned above all suffer from an inefficiency that is due reducing a single stream approximate pattern matching problem to several instances of the dictionary matching problem, where all of the instances use the same dictionary but have different input texts. The results in [24] and [37] use a separate block of space for each instance of the dictionary matching, even though the dictionaries are the same. If it would be possible to share the space usage representing the dictionary among all of the instances then that would imply an immediate improvement in the total space usage. More formally, we introduce the *dictionary matching in the multi-stream model* that captures this challenge.

Dictionary matching in the multi-stream model. In the dictionary matching in the multi-stream (DMMS) problem the input is a dictionary D to be processed, there are s independent input streams of text $T_1, T_2, \ldots, T_s$, and the goal is to report all of the occurrences of

[1] This is a common simplification in which one must only report the longest pattern that has arrived (if several patterns end at the same text location), since converting such a solution to one that reports all the patterns is straightforward, and this way the focus is on the time cost that is independent from the output size.

patterns from D in any T_i, for $1 \leq i \leq s$, as soon as the occurrence arrives. An algorithm for the DMMS problem is allowed to set up a read-only block of *shared memory* during a preprocessing phase, whose contents depend solely on D, and s blocks of *stream memory*, one for each text stream, to be used privately for each text stream as the text is being processed. Notice that it is enough to describe an algorithm that works on one stream, as long as the description details which data is stored in each type of memory. Also notice that a naïve algorithm would be to use a separate solution for dictionary matching in one stream for each one of the text streams, where each instance is stored completely in stream memory and there is no use of the shared memory. The most efficient algorithm for dictionary matching in the streaming model is due to Golan and Porat [38] using $O(d \log m)$ words and the time per character is $O(\log\log|\Sigma|)$, which could be as large as $O(\log\log(m \cdot d))$. All of these complexities are in the worst-case, and their algorithm is correct with high probability. With the naïve method, the algorithm of [38] implies a solution for DMMS that uses a total of $O(s \cdot d \log m)$ words. This space complexity is inherent in the algorithm of [38] since, in particular, their algorithm always stores the last $\Theta(d \log m)$ text characters in each stream, which does not benefit from shared memory. Thus, algorithms are only of interest if they can beat this naïve method.

1.1 Our Results

We introduce a new algorithm for the dictionary matching problem in the multi-stream model, which is summarized in the following theorem.

▶ **Theorem 1.** *There exists an algorithm for the multi-stream dictionary matching problem where each pattern has length m that uses $O(d \log m)$ words of shared memory, $O(\log m \log d)$ words of stream memory, and $O(\log m)$ time per character. All of these complexities are in the worst-case, and the algorithm is correct with high probability.*

Notice that we focus on the case where all of the patterns in D have the same length m, since this case suffices for our applications. Due to space limitations, the following extension of the results to different length patterns is left for the full version.

▶ **Theorem 2.** *There exists an algorithm for the multi-stream dictionary matching problem that uses $O(d \log m)$ words of shared memory, $O(\log m \log d)$ words of stream memory, and $O(\log m + \log d \log\log d)$ time per character, where $m = m_D$ is the length of the longest pattern in D. All of these complexities are in the worst-case, and the algorithm is correct with high probability.*

Thus, if there are s streams of data, the total space usage becomes $O(d \log m + s \log m \log d)$ words, which is substantially less than the total space usage of the naïve method. By using the algorithm of Theorem 1 we are able to reduce the space usage for solving several approximate streaming pattern matching problems, as we discuss next.

Streaming pattern matching with w wildcards. A *wildcard* character, denoted by $'?' \notin \Sigma$, is a special character that matches every character in Σ. In the streaming PMWC problem the goal is to preprocess a pattern $P[1..m]$ that contains w wildcard characters, so that given a streaming text T the algorithm reports, for each arriving character, whether the current text suffix of length m matches the pattern. The most efficient known algorithm for the PMWC problem was given in [37] where we introduced a trade-off algorithm, which for every $0 \leq \delta \leq 1$ uses $\tilde{O}(w^{1-\delta})$ amortized time per character and $\tilde{O}(w^{1+\delta})$ words of space, where $\tilde{O}$ hides poly-logarithmic factors. Our results use a reduction from the PMWC problem to the DMMS problem. Using Theorem 1 we are able to obtain the following result which is optimal, up to $polylog(m, w)$ factors.

▶ **Theorem 3.** *There exists a randomized Monte Carlo algorithm for the pattern matching with w-wildcards problem in the streaming model that succeeds with high probability, uses $\tilde{O}(w)$ words of space and spends $\tilde{O}(1)$ time per arriving text character. Moreover, any algorithm which solves the streaming pattern matching with w wildcards with high probability must use $\Omega(w)$ bits of space.*

The proof of the upper bound for Theorem 3 is obtained by directly plugging the algorithm from Theorem 1 into the reduction of [37]. The lower bound is based on a straightforward reduction from the communication complexity INDEXING problem the proof of Theorem 3 is left for the full version.

Streaming pattern matching with k-mismatches. Another application is the KMM problem in the streaming model. In this problem the goal is to preprocess a pattern $P[1..m]$ so that given a streaming text T the algorithm reports for each arriving character whether the number of mismatches between P and the current text suffix of length m is at most k, and if so then the algorithm also reports the number of mismatches. The most efficient algorithm currently published for this problem is by Clifford et al. [24]. This algorithm uses $O(k^2 \operatorname{polylog} m)$ words of space and takes $O(\sqrt{k} \log k + \operatorname{polylog} m)$ time per character. Their results use a reduction from the streaming KMM problem to the DMMS problem. Using Theorem 1 we are able to obtain the following result.

▶ **Theorem 4.** *There exists a randomized Monte Carlo algorithm for the streaming k-mismatch problem that succeeds with high probability, uses $\tilde{O}(k)$ words of space and spends $\tilde{O}(k)$ time per arriving text character.*

A proof of Theorem 4 is obtained by plugging the algorithm from Theorem 1 into the reduction of [24] (with some minor details) which uses group testing techniques [54, 53, 51, 50, 36]. The space usage of this algorithm is optimal up to $\operatorname{polylog} m$ factors [40]. Clifford, Kociumaka and Porat [26] recently posted another algorithm that obtains the same complexities (ignoring poly-logarithmic factors) but using different techniques. Nevertheless, we provide a second stronger result.

k-mismatches and periodicity. Clifford et al. [24] introduced the notion of x-period which captures the generalization of periodicity to work with a bounded number of mismatches. The number of mismatches between two equal length strings S and S' is known as the Hamming Distance and is denoted by $\mathsf{Ham}(S, S')$. The x-period of a string P of length m is the smallest integer $\pi > 0$ such that $\mathsf{Ham}(P[1+\pi..m], P[1..m-\pi]) \leq x$. Notice that for small x, most strings have a high x-period.

▶ **Theorem 5.** *There exists a randomized Monte Carlo algorithm for the streaming k-mismatch problem that succeeds with high probability, uses $\tilde{O}(k)$ words of space and spends $\tilde{O}(k)$ time per arriving text character. Moreover, if the $4k$-period of P is $\Omega(k)$, then the algorithm spends an average of $\tilde{O}(1)$ time per character.*

Since the typical assumption is that k is fairly small, the algorithm of Theorem 5 spends an average of $\tilde{O}(1)$ time per character for most patterns. Notice that Theorem 5 immediately implies Theorem 4. Due to space considerations, the proof of Theorem 5 is left for the full version.

Organization. In the rest of this paper we give an overview focusing on intuition and the general ideas of how to prove Theorem 1. The missing proofs and details are left for the full version.

1.2 Related Work

As mentioned above, the current most efficient algorithm for DM in the streaming model is due to Golan and Porat [38] where the space usage is $O(d \log m)$ words and the time per character in T is $O(\log \log |\Sigma|)$. Another relevant result is that of Clifford et al. [25], which deals with pattern matching (one pattern) in multiple streams, but not in the classic streaming model (since the space usage is not sublinear). They show how for s streams and a pattern of size m, one can report occurrences of the pattern in all of the streams, concurrently, using $O(m + s)$ words of space.

We emphasize that the Aho-Corasick automata [1] is not a multi-stream solution for the dictionary matching problem since the shared memory usage is not sub-linear. However, the stream memory usage is $O(1)$ words.

2 Preliminaries

A string S of length $|S| = \ell$ is a sequence of characters $S[1]S[2] \dots S[\ell]$ over alphabet Σ. A *substring* of S is denoted by $S[x..y] = S[x]S[x+1] \dots S[y]$ for $1 \leq x \leq y \leq \ell$. If $x = 1$ the substring is called a *prefix* of S, and if $y = \ell$, the substring is called a *suffix* of S.

A prefix of S of length $y \geq 1$ is called a *period* of S if and only if $S[i] = S[i+y]$ for all $1 \leq i \leq \ell - y$. The shortest period of S is called *the principal period* of S, and its length is denoted by ρ_S. If $\rho_S \leq \frac{|S|}{2}$ we say that S is *periodic*.

Fingerprints For a natural number n we denote $[n] = \{1, 2, \dots, n\}$. For the following let $u, v \in \bigcup_{i=0}^{n} \Sigma^i$ be two strings of size at most n. Porat and Porat [52] and Breslauer and Galil [18] extended the fingerprint method of Karp and Rabin [44], and proved that for every constant $c > 1$ there exists a *fingerprint function* $\phi : \bigcup_{i=0}^{n} \Sigma^i \to [n^c]$, such that:

1. If $|u| = |v|$ and $u \neq v$ then $\phi(u) \neq \phi(v)$ with high probability (at least $1 - \frac{1}{n^{c-1}}$).
2. *The sliding property:* Let w=uv be the concatenation of u and v. If $|w| \leq n$ then given the length and the fingerprints of any two strings from u,v and w, one can compute the fingerprint of the third string in constant time.

For two strings u and v and a fingerprint function ϕ, we say that the *fingerprint concatenation* of $\phi(u)$ and $\phi(v)$ is $\phi(uv)$. If we are given the lengths of u and v then computing the fingerprint concatenation takes constant time due to the sliding property.

Remark. Our algorithm often uses fingerprints in order to quickly test if two strings are equal or not. To ease presentation, in the rest of the paper we assume that fingerprints never give false positives. This assumption is acceptable since the algorithm is allowed to fail with small probability.

3 Same Length Patterns – Proof of Theorem 1

Throughout the paper, let q denote the current index of the last character in T. The algorithm initially considers every text location as a *candidate* for an occurrence of a pattern. Conceptually, a text location c is considered to be a candidate until the algorithm encounters proof that there cannot be any pattern in D that appears at location c. This leads to a naïve solution which stores all of the candidates, and each time a new character arrives the candidates are tested to see if they are still candidates. Regardless of the method of testing, this solution is too expensive, in terms of both space and time, since the number of candidates could be $\Omega(m)$.

In order to reduce the time cost per character, we borrow a technique introduced in [52] which considers prefixes of the dictionary strings of exponentially growing length. This technique has been extensively used for streaming pattern matching algorithms [30, 18, 23, 24, 37, 38, 42, 26, 55], but in our case the details are more delicate than usual. Our algorithm makes use of an increasing sequence of $O(\log m)$ shift values $\Delta = (\delta_0, \delta_1, \ldots, \delta_{|\Delta|-1})$ where

$$\delta_k = \begin{cases} 25 & \text{if } k = 0 \\ \min(5\lfloor \frac{6}{25}\delta_{k-1} \rfloor, m) & \text{otherwise.} \end{cases}$$

Notice that for $1 \leq k < |\Delta| - 1$ we have $\delta_k - \delta_{k-1} \leq \delta_k/5$. Denote $\ell_k = \delta_k/5$. For each $0 \leq k \leq |\Delta| - 1$ let $D_k = \{P[1..\delta_k] | P \in D\}$ be the set of prefixes of patterns from D of length δ_k. Let $F_k = \{\phi(P) | P \in D_k\}$ be the set of fingerprints of patterns in D_k.

The intuition behind the sets D_k for $\delta_k \in \Delta$ is that our algorithm first finds occurrences of patterns from D_{k-1}, and those occurrences are then used for finding occurrences of patterns from D_k. Notice that D_0 contains only constant sized prefixes of patterns (of length 25). It is straightforward to find occurrences of these prefixes using $O(d)$ words in shared memory, $O(1)$ words in stream memory and $O(1)$ time per text character. Also notice that $D_{|\Delta|-1} = D$, and so once a pattern from $D_{|\Delta|-1}$ is found, it is reported immediately. Most of the technical work is on patterns in D_k for $1 \leq k < |\Delta| - 1$. Thus, the rest of the discussion primarily focuses on $\delta_1, \delta_2, \ldots, \delta_{|\Delta|-2}$.

Testing candidates. Testing whether a candidate c is still a candidate takes place only when $q = c + \delta_k - 1$ for some $\delta_k \in \Delta$ (so there are only $O(\log m)$ tests per each arrival of a text character). At this point in time, the suffix of T starting at location c is of length δ_k. Notice that for c to end up being an occurrence of some pattern it must be that $T[c..q] \in D_k$. The *text fingerprint* $\phi(T[1..q])$ is the fingerprint of the text up to the last character that has arrived, and is maintained with $O(1)$ space and in $O(1)$ time per character. The *candidate fingerprint* $\phi(T[1..c-1])$ is the fingerprint of the text prefix up to location c. With access to the candidate fingerprint (which we describe below) and the text fingerprint, the algorithm uses the sliding property to compute in constant time the fingerprint $\phi(T[c..q])$, and then tests whether $\phi(T[c..q]) \in F_k$ (thereby testing whether $T[c..q] \in D_k$) via a static hash table.

One almost trivial way of providing access to the candidate fingerprints is to store (in local stream memory) the candidates via a linked list together with some additional $O(1)$ information per candidate to help compute the candidate fingerprint. Unfortunately, the number of candidates could be as large as $\Omega(m)$, and so we cannot afford to store explicit information for each candidate. Instead, we devise a new method for implicitly storing the candidates so that whenever a new text character arrives we can quickly infer which candidates need to be tested (if any), and then quickly extract the candidate fingerprints of the tested candidates. This task is accomplished with the aid of *guiding graphs*.

The guiding graph. For each D_k the algorithm stores a directed edge-weighted graph G_k in shared memory, called a *guiding graph*. In order to simplify the presentation, we focus on a simplified version of the guiding graph. A *pseudo-forest* is an undirected graph where each connected component contains at most one cycle. G_k is a directed pseudo-forest in which the out-degree of each vertex is at most 1. Each edge e in G_k has a weight $w(e)$ and label $\lambda(e)$ such that there exists a non-empty *edge string* S_e where $\lambda(e) = \phi(S_e)$ and $w(e) = |S_e| \geq 1$. For each $P \in D_k$ there is an associated vertex $v_P \in G_k$. The total size of G_k is $O(|D_k|) = O(d)$.

The algorithmic usefulness of G_k is due to a *directed path* (DP) property which captures the combinatorial guarantees given by the guiding graph. We first state a stronger version of the DP property, which unfortunately we do not know how to guarantee as stated. In Section 4 we give a weaker version, which we do know how to guarantee, but the weaker version introduces an extra $O(\log d)$ factor in stream memory.

▶ **Property 1** (Strong Directed Path Property). *Let S be a string where $\delta_k \leq |S| < \delta_{k+1}$, such that the prefix and suffix of S of length δ_k are $P_b, P_e \in D_k$, respectively. Then there exists a single directed path π in G_k from v_b to v_e (which may contain cycles) such that the concatenation of the edge strings for the sequence of edges on π is exactly $S[1..|S| - \delta_k]$, which is the prefix of S until the occurrence of the suffix P_e.*

If $P \in D_k$ is a substring *of S at location h, then the path starting from v_b with total edge weight $h - 1$ must exist and end at v_P, so $v_P \in \pi$[2]. Moreover, for any prefix of π, with total edge weight w, the concatenation of the edge strings on this path prefix is $S[1..w]$.*

The intuition behind the usage of the guiding graph is that the guiding graph cleverly represents all possible linked lists of candidates in the *text interval* $I_k = (q-\delta_{k+1}+1, q-\delta_k+1]$ that can ever be encountered by the algorithm. For a location c in I_k, let the *entrance prefix* of c be $T[c..c + \delta_k - 1]$. Notice that c is a candidate if and only if the entrance prefix of c is some pattern $P \in D_k$. For a candidate c with entrance prefix $P \in D_k$, we denote $v_c = v_P$. Thus, all of the candidates in I_k are the candidates c for which the last verification took place when the character $T[c + \delta_k - 1]$ arrived, and so the entrance prefixes of candidates in I_k are patterns from D_k. Let L be the list of candidates in I_k. Let c_b (c_e) be the first (last) candidate in L, and let $P_b \in D_k$ ($P_e \in D_k$) be the string of length δ_k occurring at location c_b (c_e) in T. Both c_b and c_e are in I_k, and so $c_e - c_b < \delta_{k+1} - \delta_k$. Since $|P_b| = |P_e| = \delta_k$, the substring $S = T[c_b..c_e + |P_e| - 1]$ has P_b and P_e as its prefix and suffix, respectively, and $\delta_k \leq |S| < \delta_{k+1}$. Thus, by the strong DP property, there exists a path π_L in G_k from v_{c_b} to v_{c_e}, and the concatenation of edge strings on π_L is exactly $S[1..|S| - \delta_k] = T[c_b..c_e - 1]$. Moreover, for any candidate c in L we have $v_c \in \pi_L$ and the concatenation of the edge strings on a prefix π_c of π_L from v_{c_b} to v_c is exactly $S[1..c - c_b] = T[c_b..c - 1]$. Being that $T[1..c - 1]$ is the concatenation of $T[1..c_b - 1]$ and $T[c_b..c - 1]$, the candidate fingerprint of c is derivable from the candidate fingerprint of c_b and concatenation of the edge strings on π_c. Thus, if G_k is stored in shared memory, then we are able to recover the candidate fingerprints of all of the candidates in L by storing, in stream memory, a pointer to the beginning and of π_L together with locations c_b, c_e[3], and the fingerprint candidate of c_b. In some sense this feature allows us to access information about the history of the text stream, although this information is not stored explicitly.

Phantom Candidates. While the strong DP property guarantees that every candidate c in L has a corresponding vertex in π_L, the property does not guarantee that every vertex in π_L corresponds to a candidate in L. In particular, let $\pi_L = (v_1, v_2, \ldots, v_x)$ and notice that π_L may contain duplicate vertices (since π_L may contain a cycle). By the strong DP property, if the ordered list of candidates in L is $(c_1, c_2 \ldots, c_y)$ then there exist indices $1 = i_1 < i_2 < \cdots < i_y = x$ such that for all $1 \leq j \leq y$, v_{i_j} corresponds to c_j. However, for

[2] Notice that if π contains a cycle, then there may be several prefixes of π ending at v_P but only one of them can have a specific weight.

[3] The reason for storing c_e is in order to know where π_L ends. This is particularly important when G_k contains a cycle.

every $1 \leq z \leq x$ such that for every $1 \leq j \leq y$, $z \neq i_j$, we have that v_z at location z in the list[4] does not correspond to a candidate in L. This means that the implicit representation of L through π_L may contain irrelevant information.

To overcome this issue, we allow the vertices on π_L that do not correspond to candidates to be considered *as if* they are candidates, which we call *phantom* candidates. A phantom candidate is a text location that failed a test in the past, but now is implicitly considered again because its corresponding vertex lies on a directed path that is implicitly stored via the path's endpoints. A crucial aspect of phantom candidates is that they do *not* affect the complexity bounds or correctness of our algorithm. This will be made clear in the complexity analysis. If c is a phantom candidate due to vertex $v_i \in \pi_L$, we say that $v_c = v_i$.

Updating π_L. If L is empty and a new candidate c enters I_k, then π_L becomes the single vertex v_c. If L is not empty, let c_b and c_e be the first and last candidates in L, respectively. At this time it is possible that c_b is a phantom candidate, but we guarantee that c_e is not. Assume by induction that π_L is currently the path from v_{c_b} to v_{c_e} where the sum of the weights on edges of π_L is $c_e - c_b$.

If a new candidate c enters I_k then by the strong DP property there must exist a path from v_{c_e} to v_c. Moreover, by the strong DP property, the concatenation of edge strings (as seen when traversing π_L) from v_{c_e} to v_c is $T[c_e..c-1]$. Thus, the only change in memory needed for storing π_L is changing the stored location of the last candidate in I_k to be c, which is not a phantom candidate.

If c_b leaves I_k, then v_{c_b} is removed from the beginning of π_L. If c_b was the only candidate in I_k then π_L becomes empty and we are back to the base case. Otherwise, the new first candidate c in I_k is reached by following the single outgoing edge $e = (v_{c_b}, v_c)$ in G_k. By the strong DP property, $c = c_b + w(e)$, and $\phi(T[1..c-1])$ is the fingerprint concatenation of $\phi(T[1..c_b - 1])$ and $\lambda(e)$.

Information stored in shared memory. For each D_k the data structure stores the guiding graph G_k. In addition the data structure stores the fingerprints in F_k via a perfect hash table that maps each fingerprint $\phi(P)$ to the id of P. Since the space usage per each $\delta \in \Delta$ is $O(d)$ words, the total space used in shared memory is $O(d \log m)$ words.

Information stored in stream memory. The algorithm maintains the text fingerprint of the entire text which is the fingerprint of the text up to the last character that has arrived. This information is updated in constant time per each new character arrival, using the sliding property of ϕ.

The algorithm uses a separate data structure for each of the $O(\log m)$ text intervals. The data structure for text interval I_k maintains all the candidates in I_k by storing a pointer to the beginning of π_L, the locations of the first and last candidates in I_k, and the candidate fingerprint of the first candidate.

Thus, the total space usage per each text interval I_k is $O(1)$, and the total amount of stream memory used is $O(\log m)$.

Character Arrival. We now describe the fairly straightforward processing of a new text character. In particular, we show that the algorithm spends $O(1)$ time per text interval each time a text character arrives, for a total of $O(\log m)$ time per character. Let $T[q]$ be

[4] The reason for addressing the index of v_z directly is due to the possibility of cycles.

the new character that arrives to the stream of text T. The algorithm first updates the text fingerprint of T in the stream memory, and if $\phi(T[q-24..q]) \in F_0$ then the algorithm inserts $q-24$ as a candidate into the first text interval (recall that the last 25 characters in the text are dealt with via maintaining their fingerprint for this purpose). The candidate fingerprint for $q-24$ is computed via the sliding window property from the text fingerprint and $\phi(T[q-24..q])$.

For $\delta_k \in \Delta$ the algorithm checks for the existence of a candidate $c = q - \delta_k + 1$ by probing the head of the path for I_k. If so, the algorithm has to (1) test c, (2) remove c from I_k, and (3) if c is still a candidate and $k \leq |\Delta| - 3$ then add c to the data structure for I_{k+1}, while if $k = |\Delta| - 2$ then use $F_{|\Delta|-1}$ to report the id of a pattern occurrence.

To test a candidate $c = q - \delta_k + 1$ the algorithm uses the sliding property of ϕ to compute $\phi(T[c..q])$ from the candidate fingerprint of c (which is stored in stream memory) and the text fingerprint. This takes constant time. We assume from now that c is a candidate. The process for removing a candidate from I_k and adding a candidate to I_{k+1} is described above, and costs $O(1)$ time.

Phantom candidates do not affect complexities and correctness. The treatment of phantom candidates is exactly the same as the treatment of non-phantom candidates since in our algorithm we cannot distinguish between the two. In particular, a location $c = q - \delta_{k+1} + 1$ that was a *phantom* candidate before the last character arrived is tested to see if c is an occurrence of a pattern in D_{k+1}. Since c is a phantom candidate this test **must** fail and c will not be added to the data structure of D_{k+1} (but c could potentially become a phantom candidate again later on). Thus, allowing for phantom candidates does not affect the correctness of the algorithm. Notice that allowing for phantom candidates to exist does not increase the space or time complexities: the space usage is unaffected since the phantom candidates are maintained implicitly within the directed paths, and the time complexity is unaffected since for each arriving character and for each $\delta \in \Delta$, at most one candidate (phantom or not) needs to be considered.

4 Constructing Guiding Graphs

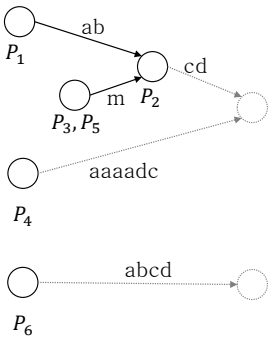
The main technical difficulty is in constructing the guiding graphs. We focus on the construction of the guiding graph for D_k. Recall that $\ell_k = \frac{\delta_k}{5}$ and that the length of every string in D_k is δ_k. A string $P \in D_k$ whose prefix of length $3\ell_k$ appears at least twice in P is said to be of type τ_{pr}, (the "pr" stands for "prefix repetition"). Since the goal here is to convey the main ideas and intuition of how to construct G_k we begin by making a simplifying assumption that there are no strings in D_k of type τ_{pr}. Removing this assumption requires introducing periodicity properties of strings, which we briefly address at the end of this section.

We define the *distance* between two strings $P, P' \in D_k$ to be the smallest possible distance between an occurrence of P and an occurrence of P' in any text. Notice that distances are never negative, and they do not define a metric (P may be at distance 1 from P' and P' may be at distance 1 from P'' but P and P'' might be at distance much larger than 2).

Intuition. The general idea is based on the following observation: if two strings in D_k are within distance at most ℓ_k in the text, then the two strings must share a common substring of length at least $4\ell_k$. However, the converse is not true. That is, not every two strings that share a common substring of length at least $4\ell_k$ have distance at most ℓ_k; see Figure 1.

P_1 = pref**sharedsubstring**suf
P_2 = ef**sharedsubstring**suffi
P_3 = in**sharedsubstring**abcde
P_4 = min**sharedsubstring**abcd
P_5 = abcde**sharedsubstring**xy

Figure 1 An example of strings in the same cluster. Notice that P_1 and P_2 could occur in a text within distance 2. Similarly P_3 and P_4 could occur in a text within distance 1. However, every other pair of strings cannot appear in any text within distance less than 15.



P_1 = abcdxyzxyzxyzxyaab
P_2 = cdxyzxyzxyzxyaabbc
P_3 = mcdxyzxyzxyzxyaabx
P_4 = aaaadcxyzxyzxyzxyx
P_5 = mcdxyzxyzxyzxyxyx
P_6 = abcdxxxxxxxxxxxxba

Figure 2 Examples of tries for clusters of strings of type τ_{npr}.

Nevertheless, we would still benefit from clustering the strings in D_k in a way that guarantees the following two properties: (1) if two strings from D_k are at distance at most ℓ_k then these strings appear in the same cluster, and (2) all of the strings in the same cluster share a common substring (possibly at different locations) of length $3\ell_k$ which we call the *seed* of the cluster. Notice that a string in a cluster may contain more than one occurrence of the seed. In order for seeds to be useful, we require that for a given seed of a cluster and a string P in that cluster, the location of the seed in P is the location of the *first* occurrence of the seed in P.

Given such a clustering, for each cluster separately we construct part of the guiding graph as follows (for an example, see Figure 2). For any pair of strings in the same cluster with distance at most ℓ_k, the algorithm synchronizes the two strings, based on the position of the seed in each one of the strings, as follows. Consider the prefix of each such string up to the occurrence of the seed. Then one of the prefixes must be a suffix of the other prefix. Thus, we consider all of the prefixes of all of the strings in a cluster, where each prefix of a string ends right before the occurrence of the seed in that string. We then construct a compacted trie from the reversal of all of the prefixes and associate each string $P \in D_k$ with the vertex in the trie corresponding to the reverse prefix of P. Each vertex in the trie has a single outgoing edge e to its parent (the root has out-degree 0), and the edge string of e is exactly the string corresponding to that edge in the compacted trie. Let T be a text and let c and c' be any two non-phantom candidates in I_k. Let P and P' be the entrance prefixes of c and c', respectively. The distance between P and P' is at most $c' - c \leq \ell_k$, and so P and P' must be in the same cluster. Moreover, $T[c..(c'-1)]$ corresponds exactly to the concatenation of the labels in the compacted trie on the single path from v_P to $v_{P'}$[5].

[5] Notice that if we were to allow strings of type τ_{pr} then this statement would no longer be true.

4.1 Creating Clusters

We consider two separate cases for the strings in D_k that are not of type τ_{pr}. The first are strings in $P \in D_k$ that have a substring of length $3\ell_k$ that occurs at least twice, but since we do not consider strings from type τ_{pr} then the prefix of P of length $3\ell_k$ does not occur more than once in P. Such strings are said to be of type τ_{npr} (the "npr" stands for "non-prefix repetition"). The second are strings that do not have a substring of length $3\ell_k$ that occurs at least twice. Such strings are said to be of type τ_{nr} (the "nr" stands for "no repetition").

Clustering for type τ_{npr}. Notice that a string P of type τ_{npr} could potentially have several substrings of length $3\ell_k$ such that each one of them appears at least twice in P. In order to remove the ambiguity, we treat the leftmost repeated string of length $3\ell_k$ in P as the *only* one that counts, and call it the *base* of P.

We cluster the strings of type τ_{npr} according to their base. That is, all of the strings in the same cluster have the same base, and this base is the seed of the cluster. Thus, we only need to show that if two strings of type τ_{npr} are at distance at most ℓ_k then these strings appear in the same cluster. The proof that this property holds is based on periodicity properties and is left for the full version.

Clustering for type τ_{nr}. Unfortunately, for type τ_{nr} it is impossible to guarantee both desired clustering properties at the same time. To see this, consider $S_1, S_2, S_3, \ldots, S_7 \in D_k$ and a text that contains all of these 7 strings, where for every $1 \leq i \leq 6$, the occurrence of S_i is exactly $\ell_k - 1$ positions before the occurrence of S_{i+1}. Then based on the properties that we are aiming for, all of these strings must appear in the same cluster. However, it is straightforward to construct such an example in which S_1 and S_7 do not share a single common character.

To solve this problem we modify the definition of guiding graphs by generalizing the definition of v_P for $P \in D_k$, using a weaker version of the DP property, and refining the properties that we require from the clustering. Instead of requiring each $P \in D_k$ to have a single associated vertex v_P, we now allow P to be associated with a set of vertices V_P. Recall that it is possible for a vertex to be associated with more than one string.

▶ **Property 2** (Weak Directed Path Property). *Let S be a string where $\delta_k \leq |S| < \delta_k + \left\lfloor \frac{\ell_k}{\log d} \right\rfloor$, such that the prefix and suffix of S of length δ_k are $P_b, P_e \in D_k$, respectively, where P_b and P_e are of type τ_{nr}. Then there exists $v_b \in V_b$ to $v_e \in V_e$ such that there exists a single directed path π in G_k from v_b to v_e and the concatenation of the edge strings for the sequence of edges on π is exactly $S[1..|S| - \delta_k]$, which is the prefix of S until the occurrence of the suffix P_e.*

If $P \in D_k$ of type τ_{nr} is a substring *of S at location h, then the path starting from v_b with total edge weight $h - 1$ must exist and end at a vertex $v_P \in V_P$, so $v_P \in \pi$. Moreover, for any prefix of π with total edge weight w, the concatenation of the edge strings on this prefix is $S[1..w]$.*

The *strong* DP property refers to strings of distance up to $\delta_{k+1} - \delta_k$, but the *weak* DP property only refers to strings of distance up to $\left\lfloor \frac{\ell_k}{\log d} \right\rfloor \leq \left\lfloor \frac{\delta_{k+1} - \delta_k}{\log d} \right\rfloor$. Thus, the algorithm uses $O(\log d)$ separate paths in order to cover the candidates in I_k which is of length $\delta_{k+1} - \delta_k$. This increases the space usage by a $O(\log d)$ factor, but the time cost remains the same.

Finally, the two properties we require from the clustering for type τ_{nr} are: (1) if two strings of type τ_{nr} are at distance at most $\lfloor \frac{\ell_k}{\log d} \rfloor$ then these strings appear together in *some* cluster, and (2) all of the strings in a cluster share a common seed of length $3\ell_k$. The details for finding such a clustering are non-obvious and are left for the full version.

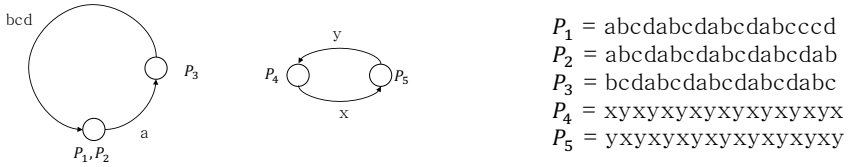


Figure 3 Example of connected components for clusters of pattern prefixes of type τ_{pr}.

Clustering for type τ_{pr}. The general idea for treating strings of type τ_{pr} is based on the following properties. Recall the definition of *base* from the clustering of τ_{npr}. Let $P \in D_k$ be of type τ_{pr}. Notice that Since P is of length $5\ell_k$ and the base of P is of length $3\ell_k$, then the two leftmost occurrences of the base in P (one of which is the prefix of P) must overlap. Denote the location of the second occurrence of the base by $r(P) + 1$. We prove (in the full version) that the prefix of P which ends after the *second* occurrence of the base in P (that is at location $r(P) + 3\ell_k$) must be periodic, and the principal period of this prefix is exactly the prefix $P[1..r(P)]$, which is the prefix of P up to the second occurrence of the base. Using periodicity techniques we are able to prove that for every two strings $P, P' \in D_k$ of type τ_{pr}, if the distance between the strings is at most $\ell_k \geq \delta_{k+1} - \delta_k$, then: (1) $r(P) = r(P')$, and (2) $P[1..r(P')]$ is a cyclic shift of $P'[1..r(P')]$. Thus, we cluster the strings in τ_{pr} such that for every two strings P and P' in the same cluster: (1) $r(P) = r(P')$, and (2) $P[1..r(P')]$ is a cyclic shift of $P'[1..r(P')]$. This will help us guarantee that the *strong* DP property holds. Finally, the cyclic shift naturally defines a directed cycle in G_k which captures the synchronization between the strings in a cluster, see Figure 3. Notice that it is possible that the same string will occur several times in a text at locations in a range shorter than ℓ_k. This case occurs only for strings that have a short period (less than ℓ_k), and is straightforward to show that all of these strings must be of type τ_{pr}. Thus, the cyclic graph description captures the relationship between possible occurrences of such a periodic string in the text, but also leads to the possibility of having a non-simple path represent many candidates.

Combining the three types. For type τ_{nr} we are able to guarantee the weak DP property, while for τ_{npr} and τ_{pr} we are able to guarantee the strong DP property, but for each type separately. That is, for two strings $P, P' \in D_k$ of different types, there is no path in G_k from a vertex corresponding to P to a vertex corresponding to P'. Thus, our algorithm creates a separate instance for each one of the three types, and runs them concurrently. Notice that it is possible for a candidate c to be of one type when c is in I_k and of a different type when c enters I_{k+1}. This is permissable since there are separate graphs for different values of k.

References

1 Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975. doi:10.1145/360825.360855.

2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. doi:10.1006/jcss.1997.1545.

3 Amihood Amir, Yonatan Aumann, Gad M. Landau, Moshe Lewenstein, and Noa Lewenstein. Pattern matching with swaps. *J. Algorithms*, 37(2):247–266, 2000. doi:10.1006/jagm.2000.1120.

4 Amihood Amir, Yonatan Aumann, Moshe Lewenstein, and Ely Porat. Function matching. *SIAM J. Comput.*, 35(5):1007–1022, 2006. doi:10.1137/S0097539702424496.

5 Amihood Amir, Estrella Eisenberg, and Ely Porat. Swap and mismatch edit distance. *Algorithmica*, 45(1):109–120, 2006. doi:10.1007/s00453-005-1192-8.

6 Amihood Amir, Martin Farach, and S. Muthukrishnan. Alphabet dependence in parameterized matching. *Inf. Process. Lett.*, 49(3):111–115, 1994. doi:10.1016/0020-0190(94)90086-8.

7 Amihood Amir, Tsvi Kopelowitz, Avivit Levy, Seth Pettie, Ely Porat, and B. Riva Shalom. Mind the gap: Essentially optimal algorithms for online dictionary matching with one gap. In *Proceedings of the 27th International Symposium on Algorithms and Computation, ISAAC*, pages 12:1–12:12, 2016. doi:10.4230/LIPIcs.ISAAC.2016.12.

8 Amihood Amir, Avivit Levy, Ely Porat, and B. Riva Shalom. Dictionary matching with one gap. In *Combinatorial Pattern Matching - 25th Annual Symposium, CPM*, pages 11–20, 2014.

9 Amihood Amir, Avivit Levy, Ely Porat, and B. Riva Shalom. Dictionary matching with a few gaps. *Theor. Comput. Sci.*, 589:34–46, 2015.

10 Amihood Amir, Moshe Lewenstein, and Ely Porat. Approximate swapped matching. *Inf. Process. Lett.*, 83(1):33–39, 2002. doi:10.1016/S0020-0190(01)00302-7.

11 Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004. doi:10.1016/S0196-6774(03)00097-X.

12 Amihood Amir and Gonzalo Navarro. Parameterized matching on non-linear structures. *Inf. Process. Lett.*, 109(15):864–867, 2009. doi:10.1016/j.ipl.2009.04.012.

13 Amihood Amir and Igor Nor. Generalized function matching. *J. Discrete Algorithms*, 5(3):514–523, 2007. doi:10.1016/j.jda.2006.10.001.

14 Tanver Athar, carl Barton, Widmer bland, Jia Gao, Costas S. Illopoulos, Chang Liu, and Solon P. Pissis. Fast circular dictionary-matching algorithm. *Mathematical Structures in Computer Science*, pages 1–14, 2015.

15 Brenda S. Baker. Parameterized pattern matching by boyer-moore-type algorithms. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 1995.*, pages 541–550, 1995. URL: http://dl.acm.org/citation.cfm?id=313651.313816.

16 Brenda S. Baker. Parameterized pattern matching: Algorithms and applications. *J. Comput. Syst. Sci.*, 52(1):28–42, 1996. doi:10.1006/jcss.1996.0003.

17 Brenda S. Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. *SIAM J. Comput.*, 26(5):1343–1362, 1997. doi:10.1137/S0097539793246707.

18 Dany Breslauer and Zvi Galil. Real-time streaming string-matching. *ACM Transactions on Algorithms*, 10(4):22:1–22:12, 2014. doi:10.1145/2635814.

19 Dany Breslauer, Roberto Grossi, and Filippo Mignosi. Simple real-time constant-space string matching. *Theor. Comput. Sci.*, 483:2–9, 2013. doi:10.1016/j.tcs.2012.11.040.

20 Peter Clifford and Raphaël Clifford. Simple deterministic wildcard matching. *Inf. Process. Lett.*, 101(2):53–54, 2007. doi:10.1016/j.ipl.2006.08.002.

21 Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. From coding theory to efficient pattern matching. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 778–784, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496855.

22 Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. Pattern matching with don't cares and few errors. *J. Comput. Syst. Sci.*, 76(2):115–124, 2010. doi:10.1016/j.jcss.2009.06.002.

23 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. Dictionary matching in a stream. In *23rd Annual European Symposium of Algorithms, ESA*, pages 361–372, 2015. doi:10.1007/978-3-662-48350-3_31.

24 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. The k-mismatch problem revisited. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2039–2052, 2016. doi:10.1137/1.9781611974331.ch142.

25 Raphaël Clifford, Markus Jalsenius, Ely Porat, and Benjamin Sach. Pattern matching in multiple streams. In *Proceedings of Combinatorial Pattern Matching - 23rd Annual Symposium, CPM 2012*, pages 97–109, 2012. doi:10.1007/978-3-642-31265-6_8.

26 Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming k-mismatch problem. *CoRR*, abs/1708.05223, 2017. arXiv:1708.05223.

27 Raphaël Clifford and Ely Porat. A filtering algorithm for k -mismatch with don't cares. In *String Processing and Information Retrieval, 14th International Symposium, SPIRE 2007, Santiago, Chile, October 29-31, 2007, Proceedings*, pages 130–136, 2007. doi:10.1007/978-3-540-75530-2_12.

28 Raphaël Clifford and Tatiana Starikovskaya. Approximate Hamming Distance in a Stream. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, pages 20:1–20:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.20.

29 Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 592–601, 2002. doi:10.1145/509907.509992.

30 Funda Ergün, Hossein Jowhari, and Mert Saglam. Periodicity in streams. In *Proceedings of Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM*, pages 545–559, 2010.

31 Guy Feigenblat, Ely Porat, and Ariel Shiftan. An improved query time for succinct dynamic dictionary matching. In *Combinatorial Pattern Matching - 25th Annual Symposium, CPM*, pages 120–129, 2014.

32 Guy Feigenblat, Ely Porat, and Ariel Shiftan. Linear time succinct indexable dictionary construction with applications. In *Data Compression Conference , DCC*, pages 13–23, 2016.

33 Michael J Fischer and Michael S Paterson. String-matching and other products. Technical report, DTIC Document, 1974.

34 Arnab Ganguly, Wing-Kai Hon, Kunihiko Sadakane, Rahul Shah, Sharma V. Thankachan, and Yilin Yang. Space-efficient dictionaries for parameterized and order-preserving pattern matching. In *27th Annual Symposium on Combinatorial Pattern Matching, CPM*, pages 2:1–2:12, 2016.

35 Arnab Ganguly, Wing-Kai Hon, and Rahul Shah. A framework for dynamic parameterized dictionary matching. In *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT*, pages 10:1–10:14, 2016.

36 Anna C. Gilbert, Hung Q. Ngo, Ely Porat, Atri Rudra, and Martin J. Strauss. 2/2-foreach sparse recovery with low risk. In *Proceedings of Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013*, pages 461–472, 2013. doi:10.1007/978-3-642-39206-1_39.

37 Shay Golan, Tsvi Kopelowitz, and Ely Porat. Streaming Pattern Matching with d Wildcards. In *24th Annual European Symposium on Algorithms (ESA)*, pages 44:1–44:16, 2016.

38 Shay Golan and Ely Porat. Real-time streaming multi-pattern search for constant alphabet. In *25th Annual European Symposium on Algorithms, ESA 2017*, pages 41:1–41:15, 2017. doi:10.4230/LIPIcs.ESA.2017.41.

39 Carmit Hazay, Moshe Lewenstein, and Dina Sokol. Approximate parameterized matching. *ACM Trans. Algorithms*, 3(3):29, 2007. doi:10.1145/1273340.1273345.

40 Wei Huang, Yaoyun Shi, Shengyu Zhang, and Yufan Zhu. The communication complexity of the hamming distance problem. *Inf. Process. Lett.*, 99(4):149–153, 2006. doi:10.1016/j.ipl.2006.01.014.

41 Piotr Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98*, pages 166–173, 1998. doi:10.1109/SFCS.1998.743440.

42 Markus Jalsenius, Benny Porat, and Benjamin Sach. Parameterized matching in the streaming model. In *Proceedings Symposium on Theoretical Aspects of Computer Science, STACS*, pages 400–411, 2013. doi:10.4230/LIPIcs.STACS.2013.400.

43 Adam Kalai. Efficient pattern-matching with don't cares. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2002*, pages 655–656, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545468.

44 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.

45 Tsvi Kopelowitz, Ely Porat, and Yaron Rozen. Succinct online dictionary matching with improved worst-case guarantees. In *27th Annual Symposium on Combinatorial Pattern Matching, CPM*, pages 6:1–6:13, 2016.

46 Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theor. Comput. Sci.*, 43:239–249, 1986. doi:10.1016/0304-3975(86)90178-7.

47 Lap-Kei Lee, Moshe Lewenstein, and Qin Zhang. Parikh matching in the streaming model. In *String Processing and Information Retrieval - 19th International Symposium, SPIRE 2012, Proceedings*, pages 336–341, 2012. doi:10.1007/978-3-642-34109-0_35.

48 S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005. doi:10.1561/0400000002.

49 S. Muthukrishnan and H. Ramesh. String matching under a general matching relation. In *Foundations of Software Technology and Theoretical Computer Science, 12th Conference, New Delhi, India, December 18-20, 1992, Proceedings*, pages 356–367, 1992. doi:10.1007/3-540-56287-7_118.

50 Hung Q. Ngo, Ely Porat, and Atri Rudra. Efficiently decodable error-correcting list disjunct matrices and applications - (extended abstract). In *Proceedings of Automata, Languages and Programming - 38th International Colloquium, ICALP 2011*, pages 557–568, 2011. doi:10.1007/978-3-642-22006-7_47.

51 Hung Q. Ngo, Ely Porat, and Atri Rudra. Efficiently decodable compressed sensing by list-recoverable codes and recursion. In *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012*, pages 230–241, 2012. doi:10.4230/LIPIcs.STACS.2012.230.

52 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009*, pages 315–323, 2009. doi:10.1109/FOCS.2009.11.

53 Ely Porat and Ohad Lipsky. Improved sketching of hamming distance with error correcting. In *Combinatorial Pattern Matching, 18th Annual Symposium, CPM 2007, London, Canada, July 9-11, 2007, Proceedings*, pages 173–182, 2007. doi:10.1007/978-3-540-73437-6_19.

54 Ely Porat and Amir Rothschild. Explicit nonadaptive combinatorial group testing schemes. *IEEE Trans. Information Theory*, 57(12):7982–7989, 2011. doi:10.1109/TIT.2011.2163296.

55 Jakub Radoszewski and Tatiana A. Starikovskaya. Streaming k-mismatch with error correcting and applications. In *2017 Data Compression Conference, DCC*, pages 290–299, 2017. doi:10.1109/DCC.2017.14.

56 Tatiana Starikovskaya. Communication and Streaming Complexity of Approximate Pattern Matching. In *28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)*, pages 13:1–13:11, 2017. doi:10.4230/LIPIcs.CPM.2017.13.

Gray Codes and Symmetric Chains

Petr Gregor
Department of Theoretical Computer Science and Mathematical Logic, Charles University, Prague, Czech Republic
gregor@ktiml.mff.cuni.cz

Sven Jäger
Institut für Mathematik, Technische Universität Berlin, Germany
jaeger@math.tu-berlin.de

Torsten Mütze
Institut für Mathematik, Technische Universität Berlin, Germany
muetze@math.tu-berlin.de

Joe Sawada
School of Computer Science, University of Guelph, Canada
jsawada@uoguelph.ca

Kaja Wille
Institut für Mathematik, Technische Universität Berlin, Germany
wille@math.tu-berlin.de

Abstract

We consider the problem of constructing a cyclic listing of all bitstrings of length $2n+1$ with Hamming weights in the interval $[n+1-\ell, n+\ell]$, where $1 \leq \ell \leq n+1$, by flipping a single bit in each step. This is a far-ranging generalization of the well-known middle two levels problem ($\ell = 1$). We provide a solution for the case $\ell = 2$ and solve a relaxed version of the problem for general values of ℓ, by constructing cycle factors for those instances. Our proof uses symmetric chain decompositions of the hypercube, a concept known from the theory of posets, and we present several new constructions of such decompositions. In particular, we construct four pairwise edge-disjoint symmetric chain decompositions of the n-dimensional hypercube for any $n \geq 12$.

2012 ACM Subject Classification Mathematics of computing → Combinatorics, Mathematics of computing → Graph theory

Keywords and phrases Gray code, Hamilton cycle, hypercube, poset, symmetric chain

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.66

Related Version A full version is available at `https://arxiv.org/abs/1802.06021`.

1 Introduction

Gray codes are named after Frank Gray, a researcher at Bell Labs, who described a simple method to generate all 2^n bitstrings of length n by flipping a single bit in each step [8], now known as the binary reflected Gray code. This code found widespread use, e.g., in circuit design and testing, signal processing and error correction, data compression, etc.; many more applications are mentioned in the survey [28]. The binary reflected Gray code is also implicit in the well-known *Towers of Hanoi* puzzle and the *Chinese ring* puzzle that date back to the 19th century. The theory of Gray codes has developed considerably in the last decades, and the term is now used more generally to describe an exhaustive listing of any class of combinatorial objects where successive objects in the list differ by a

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 66; pp. 66:1–66:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

small amount. In particular, such generation algorithms have been developed for several fundamental combinatorial objects of interest for computer scientists, such as bitstrings, permutations, partitions, trees, etc., all of which are covered in depth in the most recent volume of Knuth's seminal series *The Art of Computer Programming* [20].

Since the discovery of the binary reflected Gray code, there has been continued interest in developing Gray codes for bitstrings of length n that satisfy various additional constraints. For instance, a Gray code with the property that each bit is flipped (almost) the same number of times was first constructed by Tootill [35]. Goddyn and Gvozdjak constructed an n-bit Gray code in which any two flips of the same bit are almost n steps apart [7], which is best possible. These are only two examples of a large body of work on possible Gray code transition sequences; see also [3, 5, 34]. Savage and Winkler constructed a Gray code that generates all 2^n bitstrings such that all bitstrings with Hamming weight k appear before all bitstrings with weight $k+2$, for each $0 \leq k \leq n-2$ [29], where the *Hamming weight* of a bitstring is the number of its 1-bits. They used this construction to tackle the infamous *middle two levels problem*, which asks for a cyclic listing of all bitstrings of length $2n+1$ with weights in the interval $[n, n+1]$ by flipping a single bit in each step. This problem was raised in the 1980s and received considerable attention in the literature (a detailed historic account is given in [22]). A general existence proof for such a Gray code for any $n \geq 1$ has been found only recently [12, 22], and an algorithm for computing it using $\mathcal{O}(1)$ amortized time and $\mathcal{O}(n)$ space was subsequently presented in [23]. The starting point of this work is the following more general problem raised in [13, 27].

▶ **Problem M** (middle 2ℓ levels problem)**.** *For any $n \geq 1$ and $1 \leq \ell \leq n+1$, construct a cyclic listing of all bitstrings of length $2n+1$ with Hamming weights in the interval $[n+1-\ell, n+\ell]$ by flipping a single bit in each step.*

The special case $\ell = 1$ of Problem M is the middle two levels problem mentioned before. The case $\ell = n+1$ is solved by the binary reflected Gray code discussed in the beginning. Moreover, the cases $\ell = n$ and $\ell = n-1$ were settled in [6, 21] and [13], respectively.

A natural framework for studying such Gray code problems is the *n-dimensional hypercube* Q_n, or n-cube for short, the graph formed by all bitstrings of length n, with an edge between any two bitstrings that differ in exactly one bit. The 5-cube is illustrated in Figure 1 (a). The *kth level* of the n-cube is the set of all bitstrings with Hamming weight exactly k. In this terminology, Problem M asks for a Hamilton cycle in the subgraph of the $(2n+1)$-cube induced by the middle 2ℓ levels.

The most general version of this problem is whether the subgraph of the n-cube induced by all levels in an arbitrary weight interval $[a, b]$ has an (almost) Hamilton cycle. This was solved in [11] for all possible values of $n \geq 1$ and $0 \leq a \leq b \leq n$, except in the cases when the length n of the bitstrings is odd and the levels a and b are symmetric around the middle, which is exactly Problem M. For all other cases that paper provides algorithms that generate each bitstring in those Gray codes in constant time.

1.1 Our results

In this work we solve the case $\ell = 2$ of Problem M, i.e., we construct a cyclic listing of all bitstrings of length $2n+1$ with Hamming weights in the interval $[n-1, n+2]$.

▶ **Theorem 1.** *For any $n \geq 1$, the subgraph of the $(2n+1)$-cube induced by the middle four levels has a Hamilton cycle.*

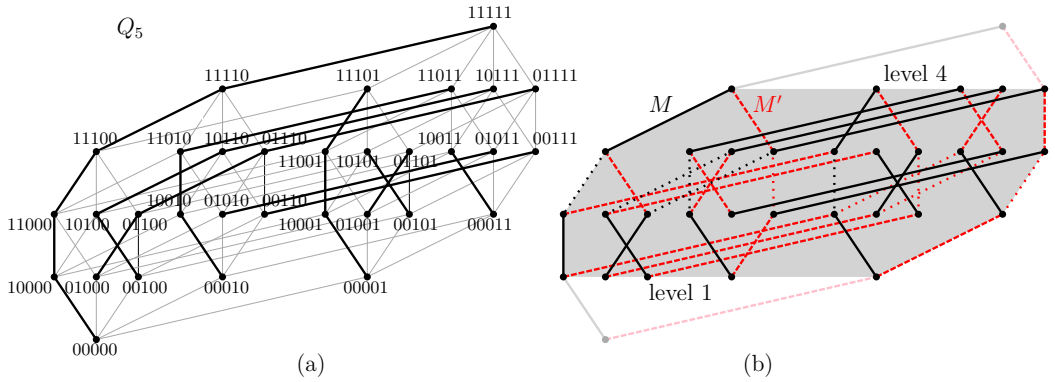


Figure 1 (a) The 5-cube with the (standard) symmetric chain decomposition $\mathcal{D}_0$, where the edges along the chains are highlighted by thick lines. (b) Building a cycle factor through the middle four levels of the 5-cube as explained in the proof of Theorem 2 with SCDs $\mathcal{D} := \mathcal{D}_0$ (black) and $\mathcal{D}' := \overline{\mathcal{D}_0}$ (red). The edges that are removed from $\mathcal{D}$ and $\mathcal{D}'$ are dotted, so the solid and dashed edges are the two matchings M and M' whose union forms the cycle factor. It has three cycles of lengths 4, 4 and 22, visiting all 30 bitstrings with Hamming weight in the interval $[1, 4]$.

Combining Theorem 1 with the results from [11] shows more generally that the subgraph of the n-cube induced by any four consecutive levels has an 'almost' Hamilton cycle.[1]

As another partial result towards Problem M, we show that the subgraph of the $(2n+1)$-cube induced by the middle 2ℓ levels has a cycle factor. A *cycle factor* is a collection of disjoint cycles which together visit all vertices of the graph. In particular, a Hamilton cycle is a cycle factor consisting only of a single cycle. Note here that the existence of a cycle factor for general values of ℓ is not an immediate consequence of Hall's theorem, which is applicable only for $\ell = 1$ and $\ell = n+1$, as only in those cases all vertices of the underlying graph have the same degree.

▶ **Theorem 2.** *For any $n \geq 1$ and $1 \leq \ell \leq n+1$, the subgraph of the $(2n+1)$-cube induced by the middle 2ℓ levels has a cycle factor.*

Our proof of Theorem 2 is concise and illustrative, and it motivates the subsequent discussion, so we present it right now. It uses a well-known concept from the theory of partially ordered sets (posets), a so-called symmetric chain decomposition. Here we define this term for the n-cube using graph-theoretic language. A *symmetric chain* in Q_n is a path $(x_k, x_{k+1}, \ldots, x_{n-k})$ in the n-cube where x_i is from level i for all $k \leq i \leq n-k$, and a *symmetric chain decomposition*, or SCD for short, is a partition of the vertices of Q_n into symmetric chains. For illustration, an SCD of Q_5 is shown in Figure 1 (a). We say that two SCDs are *edge-disjoint* if the corresponding paths in the graph Q_n are edge-disjoint, i.e., if there are no two consecutive vertices in one chain of the first SCD that are also contained in one chain of the second SCD. There is a well-known construction of two edge-disjoint SCDs in the n-cube for any $n \geq 1$ [31], which we will discuss momentarily.

[1] If the four levels are not symmetric around the middle, then this subgraph of the n-cube has two partition classes of different sizes, and thus cannot have a Hamilton cycle. However, it was shown in [11] that in those cases the graph has a cycle that visits all vertices in the smaller partition class, and also a cyclic listing of all vertices in which only few transitions flip two instead of one bit, where 'few' means only as many as the difference in size between the two partition classes. Both of these notions are natural generalizations of a Hamilton cycle.

Figure 2 The parenthesis matching approach for constructing the symmetric chain containing a bitstring x, yielding the symmetric chain decomposition $\mathcal{D}_0$. The highlighted bits are the leftmost unmatched 0 and the rightmost unmatched 1 in each bitstring.

Proof of Theorem 2. The proof is illustrated in Figure 1 (b). Consider two edge-disjoint SCDs $\mathcal{D}$ and $\mathcal{D}'$ in the $(2n+1)$-cube. Let $\mathcal{R}$ and $\mathcal{R}'$ be the chains obtained from $\mathcal{D}$ and $\mathcal{D}'$, respectively, by restricting them to the middle 2ℓ levels, so chains that are longer than $2\ell-1$ get shortened on both sides. As all chains in $\mathcal{R}$ and $\mathcal{R}'$ start and end at symmetric levels and the dimension $2n+1$ is odd, all these paths have odd length (possible lengths are $1, 3, \ldots, 2\ell-1$). Therefore, by taking every second edge on every path from $\mathcal{R}$ and $\mathcal{R}'$, we obtain two perfect matchings M and M' in the subgraph of the $(2n+1)$-cube induced by the middle 2ℓ levels. As the paths in $\mathcal{R}$ and $\mathcal{R}'$ are edge-disjoint, the matchings M and M' are also edge-disjoint. Therefore, the union of M and M' is the desired cycle factor. ◀

This proof motivates the search for a large collection of pairwise edge-disjoint SCDs in the n-cube. We can then use any two of them to construct a cycle factor as described in the previous proof, and use this cycle factor as a starting point for building a Hamilton cycle. This two-step approach of building a Hamilton cycle via a cycle factor proved to be very successful for such problems (see e.g. [15, 16, 17, 18, 22, 24, 30]). Consequently, for the rest of this section we focus on edge-disjoint SCDs in the n-cube.

There is a well-known construction of an SCD for the n-cube that is best described by the following parenthesis matching approach pioneered by Greene and Kleitman [9]; see Figure 2. For any vertex x of the n-cube, we interpret the 0s in x as opening brackets and the 1s as closing brackets. By matching closest pairs of opening and closing brackets in the natural way, the chain containing x is obtained by flipping the leftmost unmatched 0 to move up the chain, or the rightmost unmatched 1 to move down the chain, until no more unmatched bits can be flipped. It is easy to see that this indeed yields an SCD of the n-cube for any $n \geq 1$. We denote this standard SCD by $\mathcal{D}_0$; it is shown in Figure 1 (a) for $n = 5$.

By taking complements, we obtain another SCD, which we denote by $\overline{\mathcal{D}_0}$. It is not hard to see that $\mathcal{D}_0$ and $\overline{\mathcal{D}_0}$ are in fact edge-disjoint for any $n \geq 1$ [31]. Figure 1 (b) shows both SCDs for $n = 5$, and how they are used for building a cycle factor. Apart from this standard construction, we are not aware of any other construction of an SCD in the n-cube, even though there are several different ways to describe the same SCD (see e.g. [1, 4, 36]).

Our next result is a simple construction of another SCD in the n-cube for *even* values of $n \geq 2$, which we call $\mathcal{D}_1$. This construction is based on lattice paths and will be explained in Section 4 below. It has the additional feature that $\mathcal{D}_0$, $\overline{\mathcal{D}_0}$, $\mathcal{D}_1$ and $\overline{\mathcal{D}_1}$ are pairwise edge-disjoint for $n \geq 6$.

▶ **Theorem 3.** *For any even $n \geq 6$, the n-cube contains four pairwise edge-disjoint SCDs.*

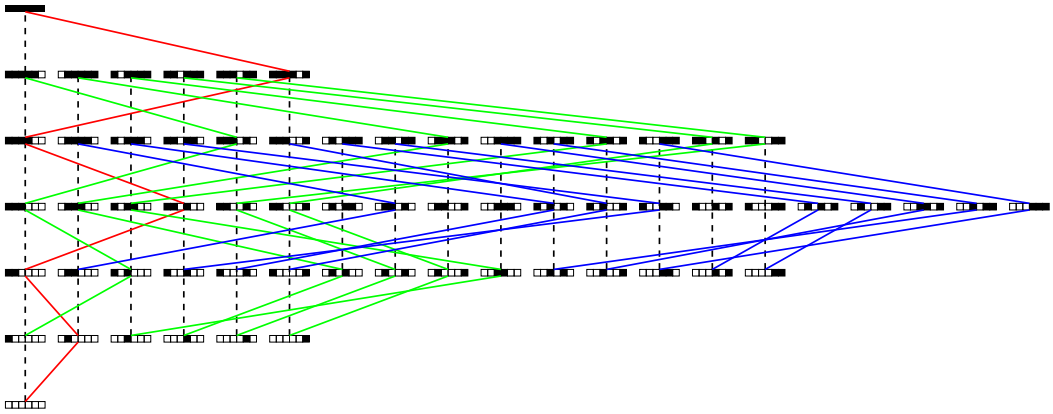


Figure 3 The edge-disjoint SCDs $\mathcal{D}_0$ (dashed vertical paths) and $\mathcal{D}_1$ (solid paths; chains of the same length are drawn with the same color) in the 6-cube. The bitstrings are drawn with white squares representing 0s and black squares representing 1s.

Table 1 Known pairwise edge-disjoint SCDs in the n-cube for $n = 1, 2, \ldots, 11$. The definitions of $\mathcal{X}_5, \mathcal{Y}_5, \mathcal{Z}_5$ and $\mathcal{X}_7, \mathcal{Y}_7$ are given in Section 4.3.

n	1	2	3	4	5	6	7	8	9	10	11
$\lfloor n/2 \rfloor + 1$	1	2	2	3	3	4	4	5	5	6	6
SCDs	$\mathcal{D}_0$	$\mathcal{D}_0, \overline{\mathcal{D}_0}$	$\mathcal{D}_0, \overline{\mathcal{D}_0}$	$\mathcal{D}_0, \overline{\mathcal{D}_0}$, $\mathcal{D}_1$	$\mathcal{X}_5, \mathcal{Y}_5$, $\mathcal{Z}_5$	$\mathcal{D}_0, \overline{\mathcal{D}_0}$, $\mathcal{D}_1, \overline{\mathcal{D}_1}$	$\mathcal{X}_7, \overline{\mathcal{X}_7}$ $\mathcal{Y}_7, \overline{\mathcal{Y}_7}$	$\mathcal{D}_0, \overline{\mathcal{D}_0}$, $\mathcal{D}_1, \overline{\mathcal{D}_1}$	$\mathcal{D}_0, \overline{\mathcal{D}_0}$	$\mathcal{D}_0, \overline{\mathcal{D}_0}$, $\mathcal{D}_1, \overline{\mathcal{D}_1}$	$\mathcal{D}_0, \overline{\mathcal{D}_0}$

Figure 3 shows the SCDs $\mathcal{D}_0$ and $\mathcal{D}_1$ in Q_6. Their complements $\overline{\mathcal{D}_0}$ and $\overline{\mathcal{D}_1}$ are not shown for clarity. Note that four edge-disjoint SCDs are best possible for Q_6, as they use up all edges incident with the middle level.

For odd values of n, we can still construct four edge-disjoint SCDs in the n-cube (except in a few small cases). However, the construction is not as direct and explicit as for even n.

▶ **Theorem 4.** *For $n = 7$ and any odd $n \geq 13$, the n-cube contains four pairwise edge-disjoint SCDs.*

For odd n, we can combine any two of the four edge-disjoint SCDs in the n-cube guaranteed by Theorem 4 to a cycle factor in the middle 2ℓ levels, as explained before, yielding in total $\binom{4}{2} = 6$ distinct cycle factors, four of which are non-isomorphic. To prove Theorem 4, we construct four edge-disjoint SCDs in the 7-cube in an ad hoc fashion and then apply the following product construction.

▶ **Theorem 5.** *If Q_a and Q_b each contain k pairwise edge-disjoint SCDs, then Q_{a+b} contains k pairwise edge-disjoint SCDs.*

Theorem 5 shows in particular that from k edge-disjoint SCDs in a hypercube of fixed dimension n, we obtain k edge-disjoint SCDs for infinitely many larger dimensions $2n, 3n, 4n, \ldots$.

We conjecture that the n-cube has $\lfloor n/2 \rfloor + 1$ pairwise edge-disjoint SCDs, but so far we only know that this holds for $n \leq 7$. Clearly, finding this many edge-disjoint SCDs would be best possible, as they use up all middle edges of the cube. Maximum sets of pairwise edge-disjoint SCDs in the n-cube we found for $n = 1, 2, \ldots, 11$ are shown in Table 1, together with the aforementioned upper bound.

1.2 Related work

Apart from building Gray codes, symmetric chain decompositions have many other interesting applications, e.g., to construct rotation-symmetric Venn diagrams for n sets when n is a prime number [14, 26], and to solve the Littlewood-Offord problem on sums of vectors [2].

A notion that is closely related to edge-disjoint SCDs is that of *orthogonal* chain decompositions, which were first considered by Shearer and Kleitman [31]. Two chain decompositions are called *orthogonal* if every pair of chains has at most one vertex in common, where one also allows chains that are not symmetric around the middle or chains that skip some levels. Shearer and Kleitman showed in their paper that $\mathcal{D}_0$ and $\overline{\mathcal{D}_0}$ are almost orthogonal (only the longest chains have two elements in common), and they conjectured that the n-cube has $\lfloor n/2 \rfloor + 1$ pairwise orthogonal chain decompositions where each decomposition consists of $\binom{n}{\lfloor n/2 \rfloor}$ many chains. Spink recently made some progress towards this conjecture, by showing that the n-cube has three orthogonal chain decompositions [32].

Pikhurko showed via a parenthesis matching argument that all edges of the n-cube can be decomposed into symmetric chains [25]. However, it is not clear whether these chains contain a subset that forms an SCD. Another interesting construction relating Hamilton cycles and SCDs in the n-cube was presented by Streib and Trotter [33]. They inductively construct a Hamilton cycle in the n-cube for any $n \geq 2$ that can be partitioned into symmetric chains forming an SCD. This Hamilton cycle has the minimal number of 'peaks' where the differences in the Hamming weight change sign.

1.3 Outline of this paper

In Section 2 we introduce several definitions that will be used throughout this paper. In Section 3 we sketch the main ideas for proving Theorem 1. The full proof is omitted due to space constraints and can be found in the preprint [10]. In Section 4 we present the proofs of Theorems 3–5, and we describe the construction of the SCD $\mathcal{D}_1$ and of the SCDs in Q_5 and Q_7 referred to in Table 1. We conclude in Section 5 with some open problems.

2 Preliminaries

We begin by introducing some terminology that is used throughout the following sections.

2.1 Bitstrings and lattice paths

We use $L_{n,k}$ to denote the set of all bitstrings of length n with Hamming weight k, so this is exactly the kth level of Q_n. For any bitstring x, we write $\overline{x}$ for its complement and $\mathrm{rev}(x)$ for the reversed bitstring. We often interpret a bitstring x as a path in the integer lattice $\mathbb{Z}^2$ starting at the origin $(0, 0)$, where every 1-bit is interpreted as an $\nearrow$-step that changes the current coordinate by $(+1, +1)$ and every 0-bit is interpreted as a $\searrow$-step that changes the current coordinate by $(+1, -1)$; see Figure 4. Note that for any bitstring x, the inverted bitstring $\overline{x}$ corresponds to mirroring the lattice path horizontally, and the inverted and reversed bitstring $\overline{\mathrm{rev}}(x)$ corresponds to mirroring the lattice path vertically.

2.2 Lexical matchings

We now introduce certain matchings between two consecutive levels of the hypercube, which were first described by Kierstead and Trotter [19]. Originally, these matchings were defined and analyzed for the graph between the middle two levels of the $(2n + 1)$-cube in an attempt

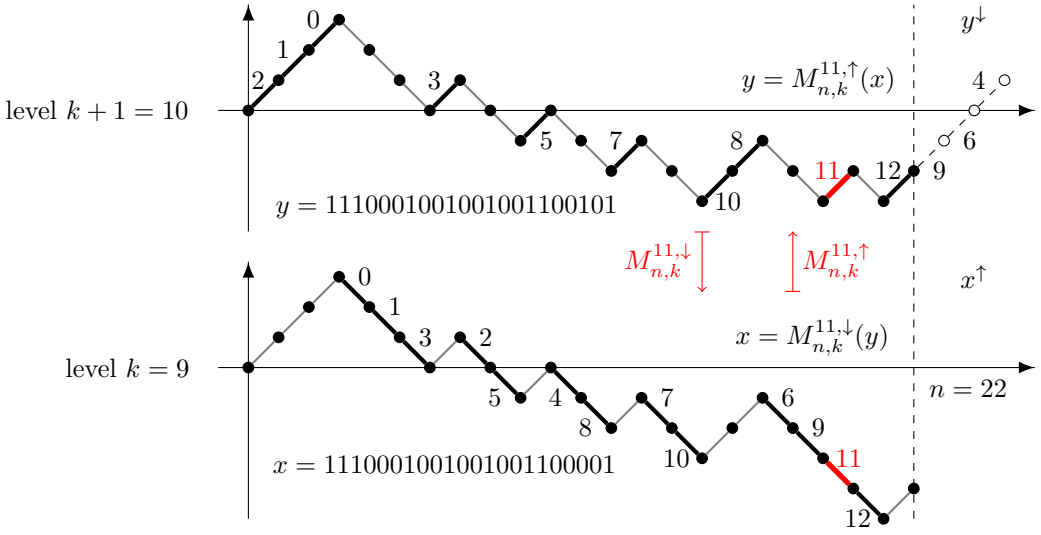


Figure 4 Definition of i-lexical matchings between levels 9 and 10 of Q_{22}, where steps flipped along the i-lexical matching edge are marked with i. Between those two levels, the vertex x is incident with i-lexical matching edges for each $i \in \{0, 1, \ldots, 12\}$, and the vertex y is incident with i-lexical matching edges for each $i \in \{0, 1, \ldots, 12\} \setminus \{4, 6, 9\}$.

to tackle the middle two levels problem. We first generalize them to the n-cube for arbitrary n and an arbitrary pair of consecutive levels k and $k+1$. For $i \in \{0, 1, \ldots, n-1\}$ the *i-lexical matching* is defined as follows; see Figure 4. We interpret a bitstring x as a lattice path, and we let $x^{\uparrow}$ denote the lattice path that is obtained by appending $\searrow$-steps to x until the resulting path ends at height -1. If x ends at a height less than -1, then $x^{\uparrow} := x$. Similarly, we let $x^{\downarrow}$ denote the lattice path obtained by appending $\nearrow$-steps to x until the resulting path ends at height $+1$. If x ends at a height more than $+1$, then $x^{\downarrow} := x$. We define the matching by two partial mappings $M_{n,k}^{i,\uparrow} \colon L_{n,k} \to L_{n,k+1}$ and $M_{n,k}^{i,\downarrow} \colon L_{n,k+1} \to L_{n,k}$ defined as follows: For any $x \in L_{n,k}$ we consider the lattice path $x^{\uparrow}$ and scan it row-wise from top to bottom, and from right to left in each row. The partial mapping $M_{n,k}^{i,\uparrow}(x)$ is obtained by flipping the ith $\searrow$-step encountered in this fashion, where counting starts with $0, 1, \ldots$, if this $\searrow$-step is part of x; otherwise x is left unmatched. Similarly, for any $x \in L_{n,k+1}$ we consider the lattice path $x^{\downarrow}$ and scan it row-wise from top to bottom, and from left to right in each row. The partial mapping $M_{n,k}^{i,\downarrow}(x)$ is obtained by flipping the ith $\nearrow$-step encountered in this fashion if this $\nearrow$-step is part of x; otherwise x is left unmatched. It is straightforward to verify that these two partial mappings are inverse to each other, so they indeed define a matching between levels k and $k+1$ of Q_n, which we denote by $M_{n,k}^{i}$.

The following properties of lexical matchings are direct consequences of these definitions.

▶ **Lemma 6.** *Let $0 \le k \le n-1$ and $l := \max\{k, n-k-1\}$. The lexical matchings defined before have the following properties.*

- **(i)** *For every $0 \le i \le l$, the matching $M_{n,k}^{i}$ saturates all vertices in the smaller of the two levels k and $k+1$.*
- **(ii)** *The matchings $M_{n,k}^{i}$, $i = 0, 1, \ldots, l$, form a partition of all edges of the subgraph of Q_n between levels k and $k+1$.*
- **(iii)** *For every $0 \le i \le l$ we have $\overline{M_{n,k}^{i}} = M_{n,n-k-1}^{l-i}$ and $\mathrm{rev}(M_{n,k}^{i}) = M_{n,k}^{l-i}$. Consequently, we have $\overline{\mathrm{rev}}(M_{n,k}^{i}) = M_{n,n-k-1}^{i}$.*

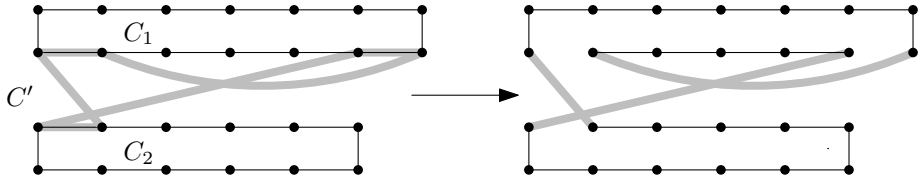


Figure 5 Joining two cycles C_1 and C_2 (black) from our cycle factor by taking the symmetric difference with a 6-cycle C' (gray).

Property (i) holds as in the smaller of the two levels k and $k+1$, no steps are appended to the lattice paths when computing the i-lexical matching between those levels. Property (ii) holds as the vertices in the smaller of the two levels k and $k+1$ have degree $l+1$ and the matchings $M^i_{n,k}$, $i = 0, 1, \ldots, l$, are all disjoint. Property (iii) follows from the observation that complementing a bitstring corresponds to mirroring the lattice path horizontally, and reverting a bitstring corresponds to mirroring the lattice path horizontally and vertically.

3 The middle four levels problem

In this section we outline the main steps for proving Theorem 1. The proof proceeds similarly as the proof of the middle two levels problem [12, 22]. In a first step, we construct a cycle factor in the middle four levels of the $(2n+1)$-cube, and in a second step we modify the cycles in the factor locally to join them to a Hamilton cycle. The cycle factor is constructed by taking the union of the following edge sets: all n-lexical and $(n+1)$-lexical matching edges between the upper two levels $n+1$ and $n+2$ and between the lower two levels $n-1$ and n, as well as certain carefully chosen edges from the $(n-2)$-lexical, the $(n-1)$-lexical, and the n-lexical matching between the middle two levels n and $n+1$. The most technical step here is to choose an appropriate set of edges between the middle two levels, so that the resulting subgraph has degree two at every vertex. When this is accomplished, we define a set of 6-cycles between levels $n+1$ and $n+2$ such that any two of these 6-cycles are edge-disjoint and every such 6-cycle C' intersects with two cycles C_1 and C_2 from our cycle factor as shown in Figure 5. Consequently, taking the symmetric difference of the edge sets of C_1, C_2, and C' results in a single cycle on the same vertex set as C_1 and C_2. We repeat this joining process until we end up with a single Hamilton cycle. In this process, we exploit that all of the 6-cycles used for the joining are edge-disjoint, and that on any cycle of the factor, no pairs of edges that two 6-cycles have in common with this cycle are interleaved, so there are never any conflicts between them. The main advantage of this two-step approach to proving Hamiltonicity is that it effectively reduces the problem of proving that a graph has a Hamilton cycle to the problem of proving that a suitably defined auxiliary graph is connected, which is much easier. All details of this proof can be found in [10].

4 Pairwise edge-disjoint SCDs

We proceed to prove Theorems 3–5.

4.1 Proof of Theorem 3

To prove Theorem 3, we first give an equivalent definition of the SCD $\mathcal{D}_0$ defined in the introduction via the parenthesis matching approach, which is valid only for even values of $n \geq 2$; recall Figure 1 (a) and Figure 2.

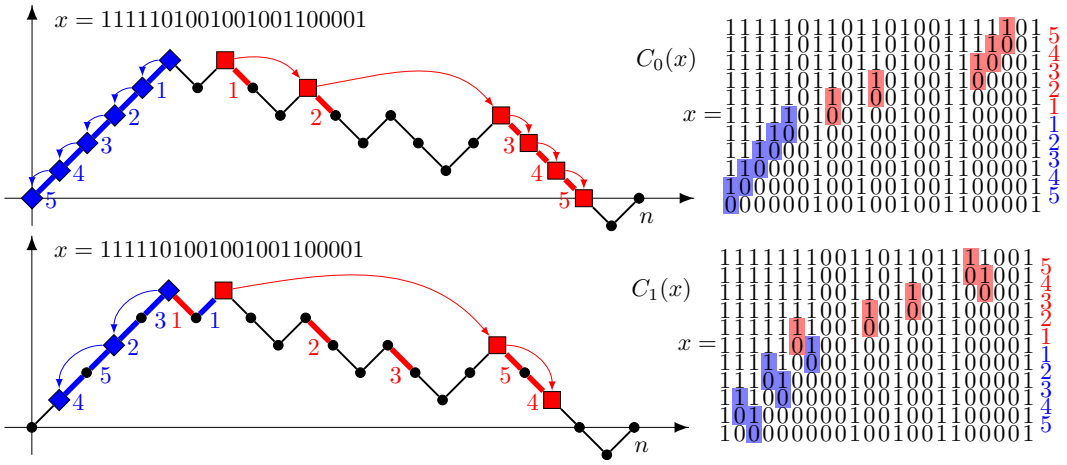


Figure 6 The labeling procedures that define the symmetric chains $C_0(x)$ (top) and $C_1(x)$ (bottom). The markers that define the upward and downward steps of the chains are drawn as a square and a diamond, respectively. The chain $C_0(x)$ is the same as the one shown in Figure 2.

For even $n \geq 2$, we consider a vertex $x \in L_{n,n/2}$ in the middle level $n/2$ of Q_n, and we define the sequence of vertices reached from x when moving up the corresponding chain, and the sequence of vertices reached when moving down the chain. For this we consider the lattice path corresponding to the bitstring x. This lattice path ends at the coordinate $(n, 0)$ as the number of 0s equals the number of 1s. We now label a subsequence of $\searrow$-steps of this lattice path with integers $j = 1, 2, \ldots$ according to the following procedure; see the top part of Figure 6 for an illustration:

(a0) We place a marker at the rightmost highest point of x and set $j := 1$.

(b0) If the marker is at height $h \geq 1$, we label the $\searrow$-step starting at the marker with j, and we move the marker to the starting point of the rightmost $\searrow$-step starting at height $h - 1$. We set $j := j + 1$ and repeat.

(c0) If the marker is at height $h = 0$, we stop.

Flipping the $\searrow$-steps of x marked with $1, 2, \ldots$ in this order yields the sequence of vertices reached from x when moving up the chain containing x. An analogous labeling procedure obtained by interchanging left and right, $\searrow$-steps and $\nearrow$-steps, and starting with ending points yields the sequence of vertices reached from x when moving down this chain. We denote this chain by $C_0(x)$. Observe that $C_0(x)$ is a symmetric chain, as the height of the marker decreases by 1 in each step, so the number of edges we move up from x equals the number of edges we move down from x. It is easy to verify that the SCD $\mathcal{D}_0$ defined before via the parenthesis matching approach satisfies $\mathcal{D}_0 = \bigcup_{x \in L_{n,n/2}} C_0(x)$.

Proof of Theorem 3. We first define a set $\mathcal{D}_1$ of chains in Q_n for even values of $n \geq 2$ via a labeling rule similar to the rule for $\mathcal{D}_0$ described before. From this definition it follows immediately that all chains in $\mathcal{D}_1$ are symmetric. We then use an equivalent characterization of $\mathcal{D}_0$ and $\mathcal{D}_1$ as the unions of certain lexical matchings to show that the chains in $\mathcal{D}_1$ form a partition of all vertices of Q_n, proving that $\mathcal{D}_1$ is an SCD, and that $\mathcal{D}_0$, $\overline{\mathcal{D}_0}$, $\mathcal{D}_1$, and $\overline{\mathcal{D}_1}$ are pairwise edge-disjoint.

For even $n \geq 2$, we consider a vertex $x \in L_{n,n/2}$ in the middle level of Q_n. We interpret it as a lattice path, and label some of its $\searrow$-steps as follows; see the bottom part of Figure 6:

(a1) We place a marker at the rightmost highest point of x and set $j := 1$. If there is a $\searrow$-step to the left of the marker starting at the same height, we label the nearest such step with 1 and set $j := 2$.

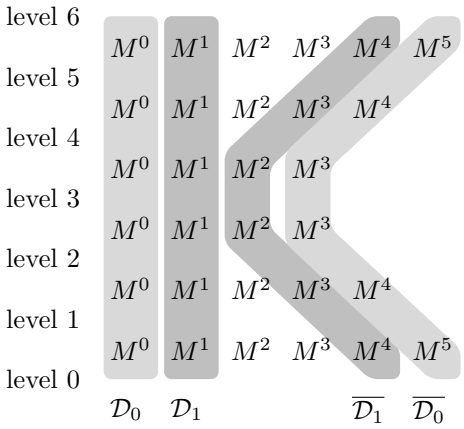


Figure 7 Unions of lexical matchings $M^i = M^i_{n,k}$ yielding edge-disjoint chain decompositions in Q_n for $n = 6$. The resulting chains in $\mathcal{D}_0$ and $\mathcal{D}_1$ in Q_6 are shown in Figure 3.

(b1) If the marker is at height $h \geq 2$, we label the rightmost $\searrow$-step starting at height $h-1$ with j. We consider all $\searrow$-steps starting at height $h-2$ to the right of the labeled step and the $\searrow$-step starting at the marker, we label the second step from the right from this set with $j+1$, and we move the marker to the starting point of the rightmost $\searrow$-step starting at height $h-2$. We set $j := j+2$ and repeat.

(c1) If the marker is at height $h = 1$ or $h = 0$, we stop.

We let $C_1(x)$ denote the chain obtained by flipping bits according to this labeling rule and the corresponding symmetric rule obtained by interchanging left and right, $\searrow$-steps and $\nearrow$-steps, and starting with ending points. Observe that $C_1(x)$ is a symmetric chain, as the height of the marker decreases by 2 in each iteration (and we label two steps in each iteration) and the conditional marking in step (a1) occurs if and only if the highest point of x is unique, so the number of edges we move up from x equals the number of edges we move down from x. At this point it is not clear yet that the chains $C_1(x)$, $x \in L_{n,n/2}$, are disjoint, nor that they cover all vertices of Q_n. This is what we will argue about next, which will prove that

$$\mathcal{D}_1 := \bigcup\nolimits_{x \in L_{n,n/2}} C_1(x) \tag{1}$$

is actually an SCD of Q_n.

By property (i) from Lemma 6, for any sequence $\mathbf{i} := (i_0, i_1, \ldots, i_{n-1})$ of indices $i_k \in \{0, 1, \ldots, \max\{k, n-k-1\}\}$ the union

$$\mathcal{D}_\mathbf{i} := \bigcup\nolimits_{k=0}^{n-1} M^{i_k}_{n,k} \tag{2}$$

is a chain decomposition of Q_n. The resulting chains are not necessarily symmetric, though. From the definitions in Section 2.2 it also follows that $\mathcal{D}_0$ equals the union of the 0-lexical matchings, and that for even $n \geq 2$, $\mathcal{D}_1$ as defined in (1) equals the union of the 1-lexical matchings; formally we have

$$\mathcal{D}_0 = \mathcal{D}_{(0,0,\ldots,0)} = \bigcup\nolimits_{k=0}^{n-1} M^0_{n,k} \quad , \quad \mathcal{D}_1 = \mathcal{D}_{(1,1,\ldots,1)} = \bigcup\nolimits_{k=0}^{n-1} M^1_{n,k} \quad .$$

Consequently, $\mathcal{D}_1$ is indeed a chain decomposition, and by the definition of $\mathcal{D}_1$ via the labeling procedure, all chains in this decomposition are symmetric, so $\mathcal{D}_1$ is indeed an SCD. The fact that $\mathcal{D}_0$, $\overline{\mathcal{D}_0}$, $\mathcal{D}_1$, and $\overline{\mathcal{D}_1}$ are pairwise edge-disjoint can be seen by applying property (iii) from Lemma 6 and by observing that by property (ii), $\mathcal{D}_\mathbf{i}$ and $\mathcal{D}_\mathbf{j}$ as defined in (2) are edge-disjoint if and only if the sequences $\mathbf{i}$ and $\mathbf{j}$ differ in every position; see Figure 7. ◀

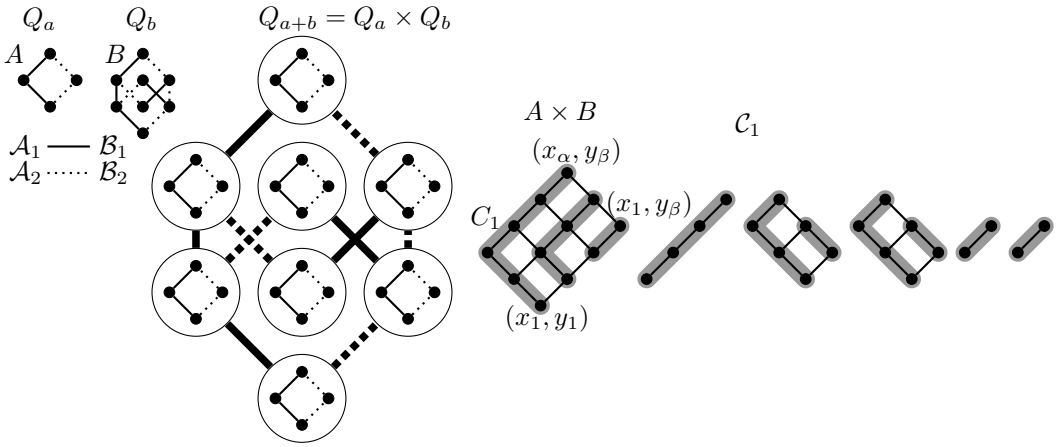


Figure 8 Illustration of the proof of Theorem 5. Construction of two edge-disjoint SCDs in Q_5 from two edge-disjoint SCDs in Q_2 and two edge-disjoint SCDs in Q_3. The chains of the SCD $\mathcal{C}_1$ of Q_5 as constructed in the proof are highlighted in gray.

Clearly, $\mathcal{D}_{(0,0,\ldots,0)}$ as defined in (2) equals $\mathcal{D}_0$ for *every* $n \geq 1$, so the union of all 0-lexical matchings forms an SCD in any dimension. In contrast to that, the union of all 1-lexical matchings $\mathcal{D}_{(1,1,\ldots,1)}$ only forms an SCD for *even* $n \geq 2$.

4.2 Proof of Theorem 5

Proof of Theorem 5. For the reader's convenience, this proof is illustrated in Figure 8. Let $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ and $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$ denote k pairwise edge-disjoint SCDs of Q_a and Q_b, respectively. We will think of Q_{a+b} as the Cartesian product $Q_a \times Q_b$ of Q_a and Q_b. We show how to construct for every $i \in [k]$ an SCD $\mathcal{C}_i$ of $Q_{a+b} = Q_a \times Q_b$ which uses only edges of the form $((u, v), (u', v'))$ where (u, u') is an edge from $\mathcal{A}_i$ or (v, v') is an edge from $\mathcal{B}_i$. From this it follows that the SCDs $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ are pairwise edge-disjoint.

The SCD $\mathcal{C}_i$ of Q_{a+b} is defined as follows: The Cartesian products $A \times B$ of chains $A \in \mathcal{A}_i$ and $B \in \mathcal{B}_i$ partition the vertices of Q_{a+b} into two-dimensional grids. $\mathcal{C}_i$ is obtained by partitioning each of those grids into symmetric chains in the natural way; see Figure 8 (cf. [4]): Specifically, let $A =: (x_1, \ldots, x_\alpha)$ and $B =: (y_1, \ldots, y_\beta)$ be the vertices in the chains A and B from bottom to top. As A and B are symmetric, we know that $|x_1| + |x_\alpha| = a$ and $|y_1| + |y_\beta| = b$, where $|x|$ denotes the Hamming weight of the bitstring x. This implies that $|(x_1, y_1)| + |(x_\alpha, y_\beta)| = |x_1| + |y_1| + |x_\alpha| + |y_\beta| = a + b$, i.e., the bottom and top vertex of the grid $A \times B$ are on symmetric levels in Q_{a+b}. We may therefore decompose $A \times B$ into disjoint symmetric chains C_j, $j = 1, 2, \ldots, \min\{\alpha, \beta\}$, by setting

$$C_j := \big((x_1, y_j), (x_2, y_j), \ldots, (x_{\alpha-j+1}, y_j), (x_{\alpha-j+1}, y_{j+1}), \ldots, (x_{\alpha-j+1}, y_\beta)\big) \ . \qquad \blacktriangleleft$$

4.3 Proof of Theorem 4

We begin by constructing the SCDs in Q_5 and Q_7 mentioned in Table 1.

▶ **Lemma 7.** *Q_5 contains three pairwise edge-disjoint SCDs, Q_7 contains four pairwise edge-disjoint SCDs, and this is best possible.*

Proof. We consider the graph Q_n with the two vertices in the outermost levels 0 and n removed, and we identify all bitstrings that differ only by rotation into so-called *necklaces*.

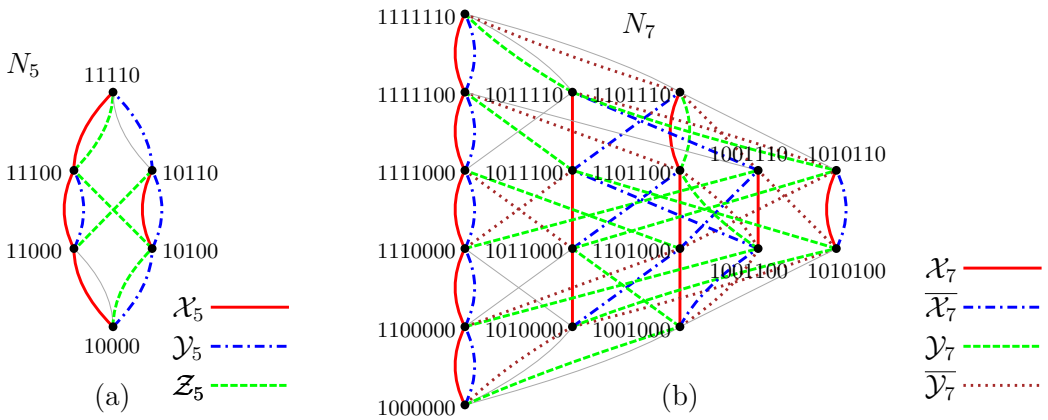


Figure 9 Illustration of the three edge-disjoint SCDs in N_5 (a) and four edge-disjoint SCDs in N_7 (b). The names of the SCDs correspond to the ones used in Table 1.

The resulting graph N_n is a multigraph version of the cover graph of the *necklace poset*. Specifically, the multiplicity of the edges in N_n corresponds to the number of ways a bit from a necklace can be flipped to reach the corresponding adjacent necklace. E.g., in N_5 the necklace $x := 10000$ has two edges leading to $y := 11000$, as we can flip the second or the fifth bit in x to reach y. This way, a necklace on level k has $n-k$ edges going up, and k edges going down, like the vertices in Q_n. The multigraphs N_5 and N_7 are shown in Figure 9. If n is prime, then every SCD in N_n corresponds to an SCD in Q_n, by turning each chain from N_n into n chains in Q_n obtained by rotating a representative of each necklace in all possible ways. Moreover, one of the chains of length $n-2$ needs to be extended by the all-zero and all-one bitstring to a chain of length n in Q_n. Observe that in this way, k edge-disjoint SCDs in N_n give rise to k edge-disjoint SCDs in Q_n.

As $n = 5$ and $n = 7$ are prime, we thus obtain three edge-disjoint SCDs in Q_5 from the SCDs in N_5 shown in Figure 9 (a), and four edge-disjoint SCDs in Q_7 from the SCDs in N_7 shown in Figure 9 (b). These SCDs use up all middle edges, so this is best possible. ◀

Proof of Theorem 4. For $n = 7$ the statement follows from Lemma 7. For odd $n \geq 13$ we apply Theorem 5 to Q_{n-7} and Q_7, using the four edge-disjoint SCDs in Q_{n-7} given by Theorem 3 (note that $n - 7 \geq 6$), and the four edge-disjoint SCDs in Q_7 given by Lemma 7. ◀

5 Open problems

- Understanding the structure of the cycle factors constructed as in the proof of Theorem 2 is an important step towards a general solution of Problem M. We performed some computer experiments in this direction; see [10]. What is the number and length of cycles in these factors? Is there a combinatorial interpretation of those numbers?
- Are there other explicit constructions of SCDs in the n-cube, different from $\mathcal{D}_0$, $\mathcal{D}_1$, and their complements?
- We conjecture that the n-cube has $\lfloor n/2 \rfloor + 1$ pairwise edge-disjoint SCDs. The main difficulty here is that we are missing a simple criterion like Hall's matching condition guaranteeing the existence of an SCD. Even finding five edge-disjoint SCDs in the n-cube for some small fixed n would be interesting, as this solution would extend to infinitely many larger values of n by Theorem 5. Beyond that, it would be very nice to construct more than constantly many edge-disjoint SCDs in the n-cube as n grows.

References

1 M. Aigner. Lexicographic matching in Boolean algebras. *J. Combin. Theory Ser. B*, 14:187–194, 1973.

2 B. Bollobás. *Combinatorics: set systems, hypergraphs, families of vectors and combinatorial probability*. Cambridge University Press, Cambridge, 1986.

3 B. Bultena and F. Ruskey. Transition restricted Gray codes. *Electron. J. Combin.*, 3(1):Paper 11, 11 pp., 1996. URL: `http://www.combinatorics.org/Volume_3/Abstracts/v3i1r11.html`.

4 N. de Bruijn, C. van Ebbenhorst Tengbergen, and D. Kruyswijk. On the set of divisors of a number. *Nieuw Arch. Wiskunde (2)*, 23:191–193, 1951.

5 D. Dimitrov, T. Dvořák, P. Gregor, and R. Škrekovski. Linear time construction of a compressed Gray code. *European J. Combin.*, 34(1):69–81, 2013. `doi:10.1016/j.ejc.2012.07.015`.

6 M. El-Hashash and A. Hassan. On the Hamiltonicity of two subgraphs of the hypercube. In *Proceedings of the Thirty-second Southeastern International Conference on Combinatorics, Graph Theory and Computing (Baton Rouge, LA, 2001)*, volume 148, pages 7–32, 2001.

7 L. Goddyn and P. Gvozdjak. Binary Gray codes with long bit runs. *Electron. J. Combin.*, 10:Paper 27, 10 pp., 2003. URL: `http://www.combinatorics.org/Volume_10/Abstracts/v10i1r27.html`.

8 F. Gray. Pulse code communication, 1953. March 17, 1953 (filed Nov. 1947). U.S. Patent 2,632,058.

9 C. Greene and D. J. Kleitman. Strong versions of Sperner's theorem. *J. Combin. Theory Ser. A*, 20(1):80–88, 1976.

10 P. Gregor, S. Jäger, T. Mütze, J. Sawada, and K. Wille. Gray codes and symmetric chains. *arXiv:1802.06021*. Preprint version of the present article, 2018.

11 P. Gregor and T. Mütze. Trimming and gluing Gray codes. *Theoret. Comput. Sci.*, 714:74–95, 2018. `doi:10.1016/j.tcs.2017.12.003`.

12 P. Gregor, T. Mütze, and J. Nummenpalo. A short proof of the middle levels theorem. To appear in *Discrete Analysis*. *arXiv:1710.08249*, 2018.

13 P. Gregor and R. Škrekovski. On generalized middle-level problem. *Inform. Sci.*, 180(12):2448–2457, 2010. `doi:10.1016/j.ins.2010.02.009`.

14 J. Griggs, C. E. Killian, and C. D. Savage. Venn diagrams and symmetric chain decompositions in the Boolean lattice. *Electron. J. Combin.*, 11(1):Paper 2, 30 pp., 2004. URL: `http://www.combinatorics.org/Volume_11/Abstracts/v11i1r2.html`.

15 A. E. Holroyd. Perfect snake-in-the-box codes for rank modulation. *IEEE Trans. Inform. Theory*, 63(1):104–110, 2017. `doi:10.1109/TIT.2016.2620160`.

16 A. E. Holroyd, F. Ruskey, and A. Williams. Shorthand universal cycles for permutations. *Algorithmica*, 64(2):215–245, 2012. `doi:10.1007/s00453-011-9544-z`.

17 J. R. Johnson. Universal cycles for permutations. *Discrete Math.*, 309(17):5264–5270, 2009. `doi:10.1016/j.disc.2007.11.004`.

18 J. R. Johnson. An inductive construction for Hamilton cycles in Kneser graphs. *Electron. J. Combin.*, 18(1):Paper 189, 12 pp., 2011.

19 H. A. Kierstead and W. T. Trotter. Explicit matchings in the middle levels of the Boolean lattice. *Order*, 5(2):163–171, 1988. `doi:10.1007/BF00337621`.

20 D. E. Knuth. *The Art of Computer Programming. Vol. 4A. Combinatorial Algorithms. Part 1*. Addison-Wesley, Upper Saddle River, NJ, 2011.

21 S. Locke and R. Stong. Problem 10892: Spanning cycles in hypercubes. *Amer. Math. Monthly*, 110:440–441, 2003.

22 T. Mütze. Proof of the middle levels conjecture. *Proc. Lond. Math. Soc.*, 112(4):677–713, 2016. `doi:10.1112/plms/pdw004`.

23 T. Mütze and J. Nummenpalo. A constant-time algorithm for middle levels Gray codes. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2238–2253. SIAM, Philadelphia, PA, 2017. doi:10.1137/1.9781611974782.147.

24 T. Mütze, J. Nummenpalo, and B. Walczak. Sparse Kneser graphs are Hamiltonian. To appear in *Proceedings of the 50th Annual ACM Symposium on the Theory of Computing (STOC 2018). arXiv:1711.01636*, 2018.

25 O. Pikhurko. On edge decompositions of posets. *Order*, 16(3):231–244 (2000), 1999. doi:10.1023/A:1006419611661.

26 F. Ruskey, C. D. Savage, and S. Wagon. The search for simple symmetric Venn diagrams. *Notices Amer. Math. Soc.*, 53(11):1304–1312, 2006.

27 C. D. Savage. Long cycles in the middle two levels of the Boolean lattice. *Ars Combin.*, 35(A):97–108, 1993.

28 C. D. Savage. A survey of combinatorial Gray codes. *SIAM Rev.*, 39(4):605–629, 1997. doi:10.1137/S0036144595295272.

29 C. D. Savage and P. Winkler. Monotone Gray codes and the middle levels problem. *J. Combin. Theory Ser. A*, 70(2):230–248, 1995. doi:10.1016/0097-3165(95)90091-8.

30 J. Sawada and A. Williams. A Hamilton path for the sigma-tau problem. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 568–575, 2018. doi:10.1137/1.9781611975031.37.

31 J. Shearer and D. J. Kleitman. Probabilities of independent choices being ordered. *Stud. Appl. Math.*, 60(3):271–276, 1979. doi:10.1002/sapm1979603271.

32 H. Spink. Orthogonal symmetric chain decompositions of hypercubes. *arXiv:1706.08545*, June 2017.

33 N. Streib and W. T. Trotter. Hamiltonian cycles and symmetric chains in Boolean lattices. *Graphs Combin.*, 30(6):1565–1586, 2014. doi:10.1007/s00373-013-1350-8.

34 I. N. Suparta and A. J. van Zanten. A construction of Gray codes inducing complete graphs. *Discrete Math.*, 308(18):4124–4132, 2008. doi:10.1016/j.disc.2007.07.116.

35 G. C. Tootill. The use of cyclic permuted codes in relay counting circuits. *Proceedings IEE, Part B Supplement*, 103, 1956.

36 D. E. White and S. G. Williamson. Recursive matching algorithms and linear orders on the subset lattice. *J. Combin. Theory Ser. A*, 23(2):117–127, 1977.

An Improved Isomorphism Test for Bounded-Tree-Width Graphs

Martin Grohe
RWTH Aachen University, Aachen, Germany
grohe@informatik.rwth-aachen.de

Daniel Neuen
RWTH Aachen University, Aachen, Germany
neuen@informatik.rwth-aachen.de

Pascal Schweitzer
Technische Universität Kaiserslautern, Kaiserslautern, Germany
schweitzer@cs.uni-kl.de

Daniel Wiebking
RWTH Aachen University, Aachen, Germany
wiebking@informatik.rwth-aachen.de

Abstract

We give a new fpt algorithm testing isomorphism of n-vertex graphs of tree width k in time $2^{k \operatorname{polylog}(k)} \operatorname{poly}(n)$, improving the fpt algorithm due to Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh (FOCS 2014), which runs in time $2^{\mathcal{O}(k^5 \log k)} \operatorname{poly}(n)$. Based on an improved version of the isomorphism-invariant graph decomposition technique introduced by Lokshtanov et al., we prove restrictions on the structure of the automorphism groups of graphs of tree width k. Our algorithm then makes heavy use of the group theoretic techniques introduced by Luks (JCSS 1982) in his isomorphism test for bounded degree graphs and Babai (STOC 2016) in his quasipolynomial isomorphism test. In fact, we even use Babai's algorithm as a black box in one place. We give a second algorithm which, at the price of a slightly worse run time $2^{\mathcal{O}(k^2 \log k)} \operatorname{poly}(n)$, avoids the use of Babai's algorithm and, more importantly, has the additional benefit that it can also be used as a canonization algorithm.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability, Mathematics of computing → Graph algorithms

Keywords and phrases graph isomorphism, graph canonization, tree width, decompositions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.67

Related Version A full version of the paper is available at `https://arxiv.org/abs/1803.06858`.

Funding Supported by the German Research Foundation DFG Koselleck Grant GR 1492/14-1

1 Introduction

Already early on in the beginning of research on the graph isomorphism problem (which asks for structural equivalence of two given input graphs) a close connection to the structure and study of the automorphism group of a graph was observed. For example, Mathon [11] argued that the isomorphism problem is polynomially equivalent to the task of computing a generating set for the automorphism group and also to computing the size of the automorphism group.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 67; pp. 67:1–67:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

With Luks's polynomial time isomorphism test for graphs of bounded degree [10], the striking usefulness of group theoretic techniques for isomorphism problems became apparent and they have been exploited ever since (e.g. [2, 12, 15, 13]). In his algorithm, Luks shows and uses that the automorphism group of a connected graph of bounded degree, after a vertex has been fixed, has a very restricted structure. More precisely, the group is in the class Γ_k of all groups whose composition factors are isomorphic to a subgroup of the symmetric group $\mathrm{Sym}(k)$.

Most recently, Babai's quasipolynomial time algorithm for general graph isomorphism [1] adds several novel techniques to tame and manage the groups that may appear as the automorphism group of the input graphs.

A second approach towards isomorphism testing is via decomposition techniques (e.g. [3, 5, 7]). These decompose the graph into smaller pieces while maintaining control of the complexity of the interplay between the pieces. When taking this route it is imperative to decompose the graph in an isomorphism-invariant fashion so as not to compare two graphs that have been decomposed in structurally different ways.

A prime example of this strategy is Bodlaender's isomorphism test [3] for graphs of bounded treewidth. Bodlaender's algorithm is a dynamic programming algorithm that takes into account all k-tuples of vertices that separate the graph, leading to a running time of $\mathcal{O}(n^{k+c})$ to test isomorphism of graphs of tree width at most k.

Only recently, Lokshtanov, Pilipczuk, Pilipczuk, and Saurabh [9] designed a fixed-parameter tractable isomorphism test for graphs of bounded tree width which has a running time of $2^{\mathcal{O}(k^5 \log k)}\,\mathrm{poly(n)}$. This algorithm first "improves" a given input graph G to a graph G^k by adding an edge between every pair of vertices between which more than k pairwise internally vertex disjoint paths exist. The improved graph G^k isomorphism-invariantly decomposes along clique separators into clique-separator free parts, which we will call *basic* throughout the paper. The decomposition can in fact be extended to an isomorphism-invariant tree decomposition into basic parts, as was shown in [4] to design a logspace isomorphism test for graphs of bounded tree width. For the basic parts, Lokshtanov et al. [9] show that, after fixing a vertex of sufficiently low degree, is it possible to compute an isomorphism-invariant tree decomposition whose bags have a size at most exponential in k and whose adhesion is at most $\mathcal{O}(k^3)$. They use this invariant decomposition to compute a canonical form essentially by a brute-force dynamic programming algorithm.

The problem of computing a canonical form is the task to compute, to a given input graph G, a graph G' isomorphic to G such that the output G' depends only on the isomorphism class of G and not on G itself.

The isomorphism problem reduces to the task of computing a canonical form: for two given input graphs we compute their canonical forms and check whether the canonical forms are equal (rather than isomorphic).

As far as we know, computing a canonical form could be algorithmically more difficult than testing isomorphism. It is usually not very difficult to turn combinatorial isomorphism tests into canonization algorithms, sometimes the algorithms are canonization algorithms in the first place. However, canonization based on group theoretic isomorphism tests is more challenging. For example, it is still open whether there is a graph canonization algorithm running in quasipolynomial time.

Our Results

Our main result is a new fpt algorithm testing isomorphism of graphs of bounded tree width.

▶ **Theorem 1.** *There is a graph isomorphism test running in time* $2^{k\,\mathrm{polylog}(k)}\,\mathrm{poly}(n)$*, where* n *is the size and* k *the minimum tree width of the input graphs.*

In the first part of the paper, we analyze the structure of the automorphism group of a graph G of tree width k. Following [9] and [4], we pursue a two-stage decomposition strategy for graphs of bounded tree width, where in the first step we decompose the improved graph along clique separators into basic parts. We observe that these basic parts are essential for understanding the automorphism groups. We show (Theorem 8) that with respect to a fixed vertex v of degree at most k, we can construct for each basic graph H an isomorphism-invariant tree decomposition of width at most $2^{\mathcal{O}(k \log k)}$ and adhesion at most $\mathcal{O}(k^2)$ where, in addition, each bag is equipped with a graph of small degree which is defined in an isomorphism-invariant way and gives us insight about the structure of the bag. In particular, using Luks's results [10], this also restricts the structure of the automorphism group.

Our construction is based on a similar construction of an isomorphism-invariant tree decomposition in [9]. Compared to that construction, we improve the adhesion (that is, the maximum size of intersections between adjacent bags of the decomposition) from $\mathcal{O}(k^3)$ to $\mathcal{O}(k^2)$. More importantly, we expand the decomposition by assigning a group and a graph to each bag.

Using these groups, we can prove that $\mathrm{Aut}(H)_v$ (the group of all automorphisms of H that keep the vertex v fixed) is a Γ_{k+1} group. This significantly restricts possible automorphism groups. Moreover, using the graph structure assigned to each bag, we can also compute the automorphism group of a graph of tree width k within the desired time bounds. The first, already nontrivial step towards computing the automorphism group, is a reduction from arbitrary graphs of tree width k to basic graphs. The second step reduces the problem of computing the automorphism group of a basic graph to the problem of computing the automorphism group of a structure that we call an *expanded d-ary tree*. In the reduction, the parameter d will be polynomially bounded in k. Then as the third step, we can apply a recent result [6] due to the first three authors that allows us to compute the automorphism groups of such expanded d-ary trees. This result is heavily based on techniques introduced by Babai [1] in his quasipolynomial isomorphism test. In fact, it even uses Babai's algorithm as a black box in one place.

We prove a second result that avoids the results of [6, 1] and even allows us to compute canonical forms, albeit at the price of an increased running time.

▶ **Theorem 2.** *There is a graph canonization algorithm running in time $2^{\mathcal{O}(k^2 \log k)} \mathrm{poly}(n)$, where n is the size and k the tree width of the input graph.*

Even though it does not employ Babai's new techniques, this algorithm still heavily depends on the group theoretic machinery. As argued above, the design of group theoretic canonization algorithms often requires extra work, and can be slightly technical, compared to the design of an isomorphism algorithm. Here, we need to combine the group theoretic canonization techniques going back to Babai and Luks [2] with graph decomposition techniques, which poses additional technical challenges and requires new canonization subroutines.

2 Preliminaries

Graphs. We use standard graph notation. All graphs $G = (V, E)$ considered are undirected finite simple graphs. We denote an edge $\{u, v\} \in E$ by uv. Let $U, W \subseteq V$ be subsets of vertices. We write $E(U, W)$ for the edges with one vertex in U and the other vertex from W, whereas $E(U)$ are the edges with both vertices in U. By $N(U)$, we denote the neighborhood of U, i.e., all vertices outside U that are adjacent to U. For the induced subgraph on U, we write $G[U]$, whereas $G - U$ is the induced subgraph on $V \setminus U$. A *rooted graph* is a

triple $G = (V, E, r)$ where $r \in V$ is the *root* of the graph. For two vertices $v, w \in V$ we denote by $\mathrm{dist}_G(v, w)$ the distance between v and w, i.e. the length of the shortest path from v to w. The *depth* of a rooted graph is the maximum distance from a vertex to the root, that is, $\mathrm{depth}(G) = \max_{v \in V} \mathrm{dist}_G(r, v)$. The *forward-degree* of a vertex $v \in V$ is $\mathrm{fdeg}(v) = |\{w \in N(v) \mid \mathrm{dist}(w, r) = \mathrm{dist}(v, r) + 1\}|$. Note that $|V| \leq (d+1)^{\mathrm{depth}(G)}$ where $d = \max_{v \in V} \mathrm{fdeg}(v)$ is the maximal forward-degree.

Separators. A pair (A, B) where $A \cup B = V(G)$ is called a *separation* if $E(A \setminus B, B \setminus A) = \emptyset$. In this case we call $A \cap B$ a *separator*. A separation (A, B) is an (L, R)-*separation* if $L \subseteq A$ and $R \subseteq B$ and in this case $A \cap B$ is called an (L, R)-*separator*. A separation (A, B) is a called a *clique separation* if $A \cap B$ is a clique and $A \setminus B \neq \emptyset$ and $B \setminus A \neq \emptyset$. In this case we call $A \cap B$ a *clique separator*.

Tree Decompositions. A *tree decomposition* of a graph G is a pair (T, β), where T is a rooted tree and $\beta : V(T) \to \mathrm{Pow}(V(G))$ is a mapping into the power set of $V(G)$ such that:

1. for each vertex $v \in V(G)$, the set $\{t \in V(G) \mid v \in \beta(t)\}$ induces a nonempty and connected subtree of T, and
2. for each edge $e \in E(G)$, there exists $t \in V(T)$ such that $e \subseteq \beta(t)$.

Sets $\beta(t)$ for $t \in V(T)$ are called the *bags* of the decomposition, while sets $\beta(s) \cap \beta(t)$ for $st \in E(T)$ are called the *adhesions sets*. The *width* of a tree decomposition T is equal to its maximum bag size decremented by one, i.e. $\max_{t \in V(T)} |\beta(t)| - 1$. The *adhesion width* of T is equal to its maximum adhesion size, i.e. $\max_{st \in E(T)} |\beta(s) \cap \beta(t)|$. The *tree width* of a graph, denoted by $\mathrm{tw}(G)$, is equal to the minimum width of its tree decompositions.

A graph G is k-*degenerate* if every subgraph of G has a vertex with degree at most k. It is well known that every graph of tree width k is k-degenerate.

Groups. For a function $\phi \colon V \to V'$ and $v \in V$ we write v^ϕ for the image of v under ϕ, that is, $v^\phi = \phi(v)$. We write composition of functions from left to right, e.g, $v^{(\sigma\rho)} = (v^\sigma)^\rho = \rho(\sigma(v))$. By $[t]$ we denote the set of natural numbers from 1 to t. By $\mathrm{Sym}(V)$ we denote the symmetric group on a set V and we also write $\mathrm{Sym}(t)$ for $\mathrm{Sym}([t])$. We use upper case Greek letters $\Delta, \Phi, \Gamma, \Theta$ and Ψ for permutation groups.

Labeling cosets. A *labeling coset* of a set V is a set of bijective mappings $\tau\Delta$ where τ is a bijection from V to $[|V|]$ and Δ is a subgroup of $\mathrm{Sym}(|V|)$. By $\mathrm{Label}(V)$, we denote the labeling coset $\tau\,\mathrm{Sym}(|V|)$. We say that $\tau\Delta$ is a *labeling subcoset* of a labeling coset $\rho\Theta$, written $\tau\Delta \leq \rho\Theta$, if $\tau\Delta$ is a subset of $\rho\Theta$ and $\tau\Delta$ forms a labeling coset again. Sometimes we will choose a single symbol to denote a labeling coset $\tau\Delta$. For this will usually use the Greek letter Λ. Recall that Γ_k denotes the class of all finite groups whose composition factors are isomorphic to subgroups of $\mathrm{Sym}(k)$. Let $\widetilde{\Gamma}_k$ be the class of all labeling cosets $\Lambda = \tau\Delta$ such that $\Delta \in \Gamma_k$.

Orderings on sets of natural numbers. We extend the natural ordering of the natural numbers to finite sets of natural numbers. For two such sets M_1, M_2 we define $M_1 \prec M_2$ if $|M_1| < |M_2|$ or if $|M_1| = |M_2|$ and the smallest element of $M_1 \setminus M_2$ is smaller than the smallest element of $M_2 \setminus M_1$.

Isomorphisms. In this paper we will always define what the isomorphisms between our considered objects are. But this can also be done in a more general context. Let $\phi : V \to V'$. For a vector $(v_1, \dots, v_k)$ we define $(v_1, \dots, v_k)^\phi$ as $(v_1^\phi, \dots, v_k^\phi)$ inductively. Analogously, for a set we define $\{v_1, \dots, v_n\}^\phi$ as $\{v_1^\phi, \dots, v_n^\phi\}$. For a labeling coset $\Lambda \leq \mathrm{Label}(V)$ we write Λ^ϕ for $\phi^{-1}\Lambda$. In the paper we will introduce isomorphisms $\mathrm{Iso}(X, X')$ for various objects X and X'. Unless otherwise stated these are all $\phi : V \to V'$ such that $X^\phi = X'$ where we apply ϕ as previously defined. For example, the isomorphism between two graphs G and G' are all $\phi : V \to V'$ such that $G^\phi = G'$ which means that G has an edge uv, if and only if G' has the edge $u^\phi v^\phi$.

3 Clique separators and improved graphs

To perform isomorphism tests of graphs of bounded tree width, a crucial step in [9] is to deal with clique separators. For this step the concept of a k-improved graph is the key.

▶ **Definition 3** ([9]). The *k-improvement* of a graph G is the graph G^k obtained from G by connecting every pair of non-adjacent vertices v, w for which there are more than k pairwise internally vertex disjoint paths connecting v and w. We say that a graph G *is k-improved* when $G^k = G$.

A graph is *k-basic* if it is k-improved and does not have any separating cliques. In particular, a k-basic graph is connected.

We summarize several structural properties of G^k.

▶ **Lemma 4** ([9]). *Let G be a graph and $k \in \mathbb{N}$.*

1. *The k-improvement G^k is k-improved, i.e., $(G^k)^k = G^k$.*
2. *Every tree decomposition (T, β) of G of width at most k is also a tree decomposition of G^k.*
3. *There exists an algorithm that, given G and k, runs in $\mathcal{O}(k^2 n^3)$ time and either correctly concludes that $\mathrm{tw}(G) > k$, or computes G^k.*

Since the construction of G^k from G is isomorphism-invariant, the concept of the improved graph can be exploited for isomorphism testing and canonization. A k-basic graph has severe limitations concerning its structure as we explore in the following sections. In the canonization algorithm from [9] a result of Leimer [8] is exploited that says that every graph has a tree decomposition into clique-separator free parts, and the family of bags is isomorphism-invariant. While it is usually sufficient to work with an isomorphism-invariant set of bags (see [14]) we actually require an isomorphism invariant decomposition, which can indeed be obtained.

▶ **Theorem 5** ([8],[4]). *There is an algorithm that, given a connected graph G, computes a tree decomposition (T, β) of G, called clique separator decomposition, with the following properties.*

1. *For every $t \in V(T)$ the graph $G[\beta(t)]$ is clique-separator free (and in particular connected).*
2. *Each adhesion set of (T, β) is a clique.*
3. *$|V(T)| \in \mathcal{O}(|V(G)|)$.*
4. *For each bag $\beta(t)$ the adhesion sets of the children are all equal to $\beta(t)$ or the adhesion sets of the children are all distinct.*

The algorithm runs in polynomial time and the output of the algorithm is isomorphism-invariant (w.r.t. G).

4 Decomposing basic graphs

In this section, we shall construct bounded-width tree decompositions of k-basic graphs of tree width at most k. Crucially, these decompositions will be isomorphism-invariant after fixing one vertex of the graph. Our construction refines a similar construction of [9].

Let us define three parameters c_{S}, c_{M}, and c_{L} (small, medium and large) that depend on k: $c_{\mathrm{S}} \coloneqq 6(k+1) \in \mathcal{O}(k)$, $c_{\mathrm{M}} \coloneqq c_{\mathrm{S}}+c_{\mathrm{S}}(k+1) \in \mathcal{O}(k^2)$ and $c_{\mathrm{L}} \coloneqq c_{\mathrm{M}}+2(k+1)\binom{c_{\mathrm{M}}}{k+2}^2 \in 2^{\mathcal{O}(k \log k)}$.

The interpretation of these parameters is that c_{M} will bound the size of the adhesion sets and c_{L} will bound the bag size. The parameter c_{S} is used by the algorithm which in certain situations behaves differently depending on sets being larger than c_{S} or not.

The bound $c_{\mathrm{M}} \in \mathcal{O}(k^2)$ improves the corresponding bound $c_{\mathrm{M}} \in \mathcal{O}(k^3)$ in [9]. However, the more significant extension of the construction in [9] is that in addition to the tree decomposition we also construct both an isomorphism-invariant graph of bounded forward-degree and depth and an isomorphism-invariant Γ_{k+1}-group associated with each bag.

The *weight* of a set $S \subseteq V(G)$ with respect to a (weight) function $w : V(G) \to \mathbb{N}$ is $\sum_{v \in S} w(v)$. The *weight* of a separation (A, B) is the weight of its separator $A \cap B$. For sets $L, R \subseteq V(G)$, among all (L, R)-separations (A, B) of minimal weight there exists a unique separation with an inclusion minimal A. For this separation we call $A \cap B$ the *leftmost minimal separator* and denote it by $S_{L,R}(w)$. Moreover, we define $S_{L,R} = S_{L,R}(\mathbf{1})$ where $\mathbf{1}$ denotes the function that maps every vertex to 1.

For $U \subseteq V(G)$ we define a weight function $w_{U,k}$ such that $w_{U,k}(u) = k$ for all $u \in U$ and $w_{U,k}(v) = 1$ for all $v \in V \setminus U$. Given a weight function w, using Menger's theorem and the Ford-Fulkerson algorithm it is possible to compute $S_{L,R}(w)$. The following lemma generalizes Lemma 3.2 of [9]. Through this generalization we obtain the adhesion bound $\mathcal{O}(k^2)$ for our decomposition.

▶ **Lemma 6.** *Let G be a graph, let $S \subseteq V(G)$ be a subset of vertices, and let $\{T_i \subseteq V(G)\}_{i \in [t]}$ and $\{w_i : V(G) \to \mathbb{N}\}_{i \in [t]}$ be families where each T_i is a minimum weight (L_i, R_i)-separator with respect to w_i for some $L_i, R_i \subseteq S$. Let $w : V(G) \to \mathbb{N}$ be another weight function such that for all $i \in [t]$:*

1. *$w(v) = w_i(v)$ for all $v \in V(G) \setminus S$, and*
2. *$w(v) \geq w_i(v)$ for all $v \in V(G)$.*

Let $D \coloneqq S \cup \bigcup_{i \in [t]} T_i$. Suppose that Z is the vertex set of any connected component of $G - D$. Then $w(N(Z)) \leq w(S)$.

The lemma can be used to extend a set of vertices S that is not a clique separator to a set D in an isomorphism-invariant fashion while controlling the size of the adhesion sets of the components of $G - D$. It will be important for us that we can also extend a labeling coset of S to a labeling coset of D and furthermore construct a graph of bounded forward-degree and depth associated with D and S.

▶ **Lemma 7.** *Let $k \in \mathbb{N}$ and let G be a graph that is k-improved. Let $S \subseteq V(G)$ and let $\Lambda \leq \mathrm{Label}(S)$ be a labeling coset such that*

1. *$\emptyset \subsetneq S \subsetneq V(G)$,*
2. *$|S| \leq c_M$,*
3. *S is not a clique,*
4. *$G - S$ is connected,*
5. *$S = N_G(V(G) \setminus S)$, and*
6. *$\Lambda \in \widetilde{\Gamma}_{k+1}$.*

There is an algorithm that either correctly concludes that $tw(G) > k$, or finds a proper superset D of S and a labeling coset $\widehat{\Lambda} \leq \mathrm{Label}(D)$ and a connected rooted graph H with the following properties:

(A) $D \supsetneq S$,
(B) $|D| \leq c_L$,
(C) $\widehat{\Lambda} \in \widetilde{\Gamma}_{k+1}$,
(D) *if Z is the vertex set of any connected component of $G - D$, then $|N(Z)| \leq c_M$,*
(E) $D \subseteq V(H)$, $\operatorname{depth}(H) \leq k+3$ *and* $\operatorname{fdeg}(v) \in k^{\mathcal{O}(1)}$ *for all* $v \in V(H)$.

The algorithm runs in time $2^{\mathcal{O}(k \log k)}|V(G)|^{\mathcal{O}(1)}$ and the output $(D, \widehat{\Lambda}, H)$ is isomorphism-invariant (w.r.t. the input data G, S, Λ and k).

Here, the output of an algorithm $\mathcal{A}$ is isomorphism-invariant if all isomorphisms between two input data (G, S, Λ, k) and (G', S', Λ', k') extend to an isomorphism between the output $(D, \widehat{\Lambda}, H)$ and $(D', \widehat{\Lambda}', H')$ (an isomorphism between (G, S, Λ, k) and (G', S', Λ', k') is a mapping $\phi : V(G) \to V(G')$ such that $(G, S, \Lambda, k)^\phi = (G', S', \Lambda', k')$ where we apply ϕ as defined in the preliminaries).

Proof. We consider two cases depending on the size of S.

Case $|S| \leq c_S$: Let $\mathcal{I} := \{(\{x\}, \{y\}) \mid x, y \in S, x \neq y, xy \notin E(G)\}$ and let $D = S \cup \bigcup_{(L,R)\in\mathcal{I}} S_{L,R}(w_{L\cup R, k+1})$. We set $w := w_{S,k+1}$ and then we have the following for every vertex set Z of a connected component of $G - D$ by Lemma 6.

$$|N(Z)| \leq w(N(Z)) \overset{6}{\leq} w(S) \leq c_S(k+1) \leq c_M.$$

For every $xy \notin E(G)$ there is a $(\{x\}, \{y\})$-separator of size at most k disjoint from $\{x, y\}$, because G is k-improved. Thus $|D| \leq |S| + k|S|^2 \leq c_S + kc_S^2 \leq c_L$. Moreover, since $G - S$ is connected and $S = N_G(V(G) \setminus S)$, for all distinct $x, y \in S$ every minimum weight $(\{x\}, \{y\})$-separator contains a vertex that is not in S. It follows that $D \neq S$.

Case $c_S < |S| \leq c_M$: Let $\mathcal{I} := \{(L, R) \mid |L| = |R| \leq k+2, |S_{L,R}| \leq k+1\}$ and $D = S \cup \bigcup_{(L,R)\in\mathcal{I}} S_{L,R}$.

The properties of D follow from similar arguments as in the first case. The fact that $\mathcal{I}$ is nonempty follows from the existence of a *balanced* separation (for details see [9]). Next, we show how to find $\widehat{\Lambda}$ in both cases. To each $x \in D \setminus S$ we associate the set $A_x := \{(L, R) \in \mathcal{I} \mid x \in S_{L,R}\}$. Two vertices x and y occur in exactly the same separators if $A_x = A_y$. In this case we call them equivalent and write $x \equiv y$. Let $A_1, \ldots, A_t \subseteq D \setminus S$ be the equivalence classes of "$\equiv$". Since each x is contained in some separator of size at most $k+1$ we conclude that the size of each A_i is at most $k+1$.

For each labeling $\lambda \in \operatorname{Label}(S)$ we choose an extension $\widehat{\lambda} : D \to \{1, \ldots, |D|\}$ such that $\widehat{\lambda}|_S = \lambda$ and for $x, y \in D \setminus S$ we have $x^{\widehat{\lambda}} < y^{\widehat{\lambda}}$ if $A_x^\lambda \prec A_y^\lambda$. (Recall that $\prec$ is the linear order of subsets of $\mathbb{N}$ as defined in the preliminaries). Inside each equivalence class A_i, the ordering is chosen arbitrarily. Define $\widehat{\Lambda} := (\{\mathrm{id}_S\} \times \operatorname{Sym}(A_1) \times \ldots \times \operatorname{Sym}(A_t)) \cdot \{\widehat{\lambda} \mid \lambda \in \Lambda\} \leq \operatorname{Label}(D)$. By construction the coset $\widehat{\Lambda}$ does not depend on the choices of the extensions $\widehat{\lambda}$. Since $|A_i| \leq k+1$ for all $1 \leq i \leq t$ we conclude that $\widehat{\Lambda} \in \widetilde{\Gamma}_{k+1}$, as desired.

It remains to explain how to efficiently compute $\widehat{\Lambda}$. For this we simply remark that it suffices to use a set of extensions $M \subseteq \Lambda$ such that Λ is the smallest coset containing all elements of M (i.e., we can use a coset analogue of a generating set). We conclude that $\widehat{\Lambda}$ can be computed in polynomial time in the size of $\mathcal{I}$.

Last but not least, we show how to construct the graph H. The Case $|S| \leq c_S$ is easy to handle. In this case we define H as the complete graph on the set $D \cup \{r\}$ where r is some new vertex, which becomes the root of H. The forward-degree of r is bounded by $|D|$ which in turn is bounded by $k^{\mathcal{O}(1)}$. We consider the Case $c_S < |S| \leq c_M$. We define $V(H) := \{(L, R) \mid L, R \subseteq S, |L| = |R| \leq k+2\} \cup D$. Clearly, we have $\mathcal{I} \subseteq V(H)$. For the root we choose $(\emptyset, \emptyset) \in V(H)$. We define the edges $E(H) := \{(L, R)(L', R') \mid L \subseteq L', R \subseteq R', |L| +$

$1 = |R|+1 = |L'| = |R'|\} \cup \{x(L,R) \mid x \in D, x \in S_{L,R}, (L,R) \in \mathcal{I}\}$. Since for each pair (L,R) there are at most $|S|^2$ different extensions (L',R') with $|L|+1 = |R|+1 = |L'| = |R'|$ and since each separator $S_{L,R}$ contains at most $k+1$ vertices, we conclude that the forward-degree of each vertex in H is bounded by $|S|^2+k+1 \in k^{\mathcal{O}(1)}$. The depth of H is at most $k+3$. ◀

A *labeled tree decomposition* (T,β,α,η) is a 4-tuple where (T,β) is a tree decomposition and α is a function that maps each $t \in V(T)$ to a labeling coset $\alpha(t) \leq \text{Label}(\beta(t))$ and η is a function that maps each $t \in V(T)$ to a graph $\eta(t)$.

The previous lemma can be used as a recursive tool to compute our desired isomorphism-invariant labeled tree decomposition.

▶ **Theorem 8.** *Let $k \in \mathbb{N}$ and let G be a k-basic graph and let v be a vertex of degree at most k. There is an algorithm that either correctly concludes that $\text{tw}(G) > k$, or computes a labeled tree decomposition (T,β,α,η) with the following properties.*

1. *the width of (T,β) is bounded by c_L,*
2. *the adhesion width of (T,β) is bounded by c_M,*
3. *the degree of T is bounded by kc_L^2 and the number of children of t with common adhesion set is bounded by k for each $t \in V(T)$,*
4. *$|V(T)|$ is bounded by $\mathcal{O}(|V(G)|)$,*
5. *for each bag $\beta(t)$ the adhesion sets of the children are all equal to $\beta(t)$ or the adhesion sets of the children are all distinct. In the former case the bag size is bounded by c_M,*
6. *for each $t \in V(T)$ the graph $\eta(t) = H_t$ is a connected rooted graph such that $\beta(t) \cup \beta(t)^2 \subseteq V(H_t)$ and for each adhesion set S there is a corresponding vertex $S \in V(H_t)$, $\text{depth}(H_t) \in \mathcal{O}(k)$ and $\text{fdeg}(v) \in k^{\mathcal{O}(1)}$ for all $v \in V(H_t)$, and*
7. *$\alpha(t) \in \widetilde{\Gamma}_{k+1}$.*

The algorithm runs in time $2^{\mathcal{O}(k^2 \log k)}|V(G)|^{\mathcal{O}(1)}$ and the output (T,β,α,η) of the algorithm is isomorphism invariant (w.r.t. G, v and k). Furthermore, if we drop Property 7 as a requirement, the triple (T,β,η) can be computed in time $2^{\mathcal{O}(k \log k)}|V(G)|^{\mathcal{O}(1)}$.

Here, the output of an algorithm $\mathcal{A}$ is isomorphism-invariant, if all isomorphisms between two input data extend to an isomorphism between the output. More precisely, an isomorphism $\phi \in \text{Iso}((G,v,k),(G',v',k'))$ extends to an isomorphism between (T,β,α,η) and $(T',\beta',\alpha',\eta')$ if there is a bijection between the tree decompositions $\phi_T : V(T) \to V(T')$ and for each node $t \in V(T)$ a bijection between the vertices of graphs $\phi_t : V(\eta(t)) \to V(\eta'(\phi_T(t)))$ which extends ϕ, i.e. $\phi_t(x) = x^\phi$ for all $x \in \beta(t) \cup \beta(t)^2 \cup 2^{\beta(t)}$ where we naturally apply ϕ as defined in the preliminaries. Furthermore, these extensions define an isomorphism between the output data, i.e. for all nodes $t \in V(T)$ we have that $\beta(t)^\phi = \beta'(\phi_T(t))$, $\alpha(t)^\phi = \alpha'(\phi_T(t))$ and $\eta(t)^{\phi_t} = \eta'(\phi_T(t))$.

▶ Remark. We later use the isomorphism-invariance of the labeled tree decomposition (T,β,α,η) from the previous theorem in more detail. Let $t \in V(T)$ be a non-root node and let $S \subseteq V(G)$ be the adhesion set to the parent node of t and let $I_t = (T_t,\beta_t,\alpha_t,\eta_t)$ be the decomposition of the subtree rooted at t and G_t the graph corresponding to I_t. Then η_t is isomorphism-invariant w.r.t. T_t, β_t, G_t and S.

5 Coset-Hypergraph-Isomorphism

After having computed isomorphism-invariant tree decompositions in the previous sections we now want to compute the set of isomorphisms from one graph to another in a bottom up fashion. Let G_1, G_2 be the two input graphs and suppose we are given isomorphism-invariant

tree decompositions (T_1, β_1) and (T_2, β_2). For a node $t \in V(T_i)$ we let $(G_i)_t$ be the graph induced by the union of all bags contained in the subtree rooted at t. The basic idea is to compute for all pairs $t \in V(T_1), t' \in V(T_2)$ the set of isomorphisms from $(G_1)_t$ to $(G_2)_{t'}$ (in addition the isomorphisms shall also respect the underlying tree decomposition) in a bottom up fashion.

The purpose of this section is to give an algorithm that solves this problem at a given bag (assuming we have already solved the problem for all pairs of children of t and t'). Let us first give some intuition for this task. Suppose we are looking for all bijections from $\beta_1(t)$ to $\beta_2(t')$ that can be extended to an isomorphism from $(G_1)_t$ to $(G_2)_{t'}$. Let $t_1, \dots, t_\ell$ be the children of t and $t'_1, \dots, t'_\ell$ the children of t'. Then we essentially have to solve the following two problems. First, we have to respect the edges appearing in the bags $\beta_1(t)$ and $\beta_2(t')$. But also, every adhesion set $\beta(t) \cap \beta(t_i)$ has to be mapped to another adhesion set $\beta(t') \cap \beta(t'_j)$ in such a way that the corresponding bijection (between the adhesion sets) extends to an isomorphism from $(G_1)_{t_i}$ to $(G_2)_{t'_j}$. In order to solve this problem we first consider the case in which the adhesion sets are all distinct and define the following abstraction.

An instance of *coset-hypergraph-isomorphism* is an 8-tuple $\mathcal{I} = (V_1, V_2, \mathcal{S}_1, \mathcal{S}_2, \chi_1, \chi_2, \mathcal{F}, \mathfrak{f})$ such that

1. $\mathcal{S}_i \subseteq \mathrm{Pow}(V_i)$,
2. $\chi_i \colon \mathcal{S}_i \to \mathbb{N}$ is a coloring,
3. $\mathcal{F} = \{\Theta_S \leq \mathrm{Sym}(S) \mid S \in \mathcal{S}_1\}$, and
4. $\mathfrak{f} = \{\tau_{S_1,S_2} \colon S_1 \to S_2 \mid S_1 \in \mathcal{S}_1, S_2 \in \mathcal{S}_2 \text{ such that } \chi_1(S_1) = \chi_2(S_2)\}$ such that
   a. every $\tau_{S_1,S_2} \in \mathfrak{f}$ is bijective, and
   b. for every color $i \in \mathbb{N}$ and every $S_1, S'_1 \in \chi_1^{-1}(i)$ and $S_2, S'_2 \in \chi_2^{-1}(i)$ and $\theta \in \Theta_{S_1}, \theta' \in \Theta_{S'_1}$ it holds that

$$\theta \tau_{S_1,S_2} (\tau_{S'_1,S_2})^{-1} \theta' \tau_{S'_1,S'_2} \in \Theta_{S_1} \tau_{S_1,S'_2}. \tag{N}$$

The instance is *solvable* if there is a bijective mapping $\phi \colon V_1 \to V_2$ such that

1. $S \in \mathcal{S}_1$ if and only if $S^\phi \in \mathcal{S}_2$ for all $S \in \mathrm{Pow}(V_1)$,
2. $\chi_1(S) = \chi_2(S^\phi)$ for all $S \in \mathcal{S}_1$, and
3. for every $S \in \mathcal{S}_1$ it holds that $\phi|_S \in \Theta_S \tau_{S,S^\phi}$.

In this case we call ϕ an *isomorphism* of the instance $\mathcal{I}$. Moreover, let $\mathrm{Iso}(\mathcal{I})$ be the set of all isomorphisms of $\mathcal{I}$. Observe that property (N) describes a consistency condition: if we can use σ_1 to map S_1 to S_2, σ_2 to map S'_1 to S_2, and σ_3 to map S'_1 to S'_2, then the mapping $\sigma_1 \sigma_2^{-1} \sigma_3$ can be used to map S_1 to S'_2. As a result the set of all isomorphisms of the instance $\mathcal{I}$ forms a coset, that is $\mathrm{Iso}(\mathcal{I}) = \Theta\phi$ for some $\Theta \leq \mathrm{Sym}(V_1)$ and $\phi \in \mathrm{Iso}(\mathcal{I})$.

In the application in the main recursive algorithm, the sets $V_i = \beta(t_i)$, the hyperedges $\mathcal{S}_i$ are the adhesion sets of t_i (and we will also encode the edges appearing in the bag in this way), and the cosets $\Theta_{S_1} \tau_{S_1,S_2}$ tell us which mappings between the adhesion sets S_1 and S_2 extend to an isomorphism between the corresponding subgraphs. The colorings χ_1 and χ_2 are used to indicate which subgraphs can not be mapped to each other (and also to distinguish between the adhesion sets and edges of the bags which will both appear in the set of hyperedges).

The next Lemma gives us one of the central subroutines for our recursive algorithm.

▶ **Lemma 9.** *Let $\mathcal{I} = (V_1, V_2, \mathcal{S}_1, \mathcal{S}_2, \chi_1, \chi_2, \mathcal{F}, \mathfrak{f})$ be an instance of coset-hypergraph-isomorphism. Moreover, suppose there are isomorphism-invariant rooted graphs $H_1 = (W_1, E_1, r_1)$ and $H_2 = (W_2, E_2, r_2)$ such that*

1. *$V_i \cup \mathcal{S}_i \subseteq W_i$,*
2. *$\mathrm{fdeg}(w) \leq d$ for all $w \in W_i$,*

3. $\operatorname{depth}(H_i) \leq h$, *and*
4. $|S| \leq d$ *for all* $S \in \mathcal{S}_i$

for all $i \in \{1, 2\}$. *Then a representation for the set* $\operatorname{Iso}(\mathcal{I})$ *can be computed in time* $2^{\mathcal{O}(h \cdot (\log d)^c)}$ *for some constant* c.

Here, isomorphism-invariant means that for every isomorphism $\phi \in \operatorname{Iso}(\mathcal{I})$ there is an isomorphism ϕ_H from H_1 to H_2 such that $v^\phi = v^{\phi_H}$ for all $v \in V_1$ and $\{v^\phi \mid v \in S\} = S^{\phi_H}$ for all $S \in \mathcal{S}_1$.

The proof of this lemma is based on the following theorem.

▶ **Theorem 10** ([6]). *Let* $\mathfrak{x}, \mathfrak{y} \colon V \to \Sigma$ *be two strings, let* $\Gamma \leq \operatorname{Sym}(V)$ *be a* Γ_d*-group and* $\gamma \in \operatorname{Sym}(V)$. *Then one can compute a representation of the set of all permutations* $\gamma' \in \Gamma\gamma$ *mapping* $\mathfrak{x}$ *to* $\mathfrak{y}$ *in time* $n^{\mathcal{O}((\log d)^c)}$ *for some constant* c *where* $n = |V|$.

Actually, we shall only need the following corollary. A rooted tree $T = (V, E, r)$ is *d-ary* if every node has at most d children. An expanded rooted tree is a tuple (T, C) where $T = (V, E, r)$ is a rooted tree and $C \colon L(T)^2 \to \operatorname{rg}(C)$ is a coloring of pairs of leaves of T ($L(T)$ denotes the set of leaves of T). Isomorphisms between expanded trees (T, C) and (T, C') are required to respect the colorings C and C'.

▶ **Corollary 11.** *Let* (T, C) *and* (T', C') *be two expanded* d*-ary trees and let* $\Gamma \leq \operatorname{Aut}(T)$ *and* $\gamma \in \operatorname{Iso}(T, T')$. *Then one can compute a representation of the set* $\{\phi \in \Gamma\gamma \mid (T, C)^\phi = (T', C')\}$ *in time* $n^{\mathcal{O}((\log d)^c)}$ *for some constant* c.

Proof sketch of Lemma 9. The proof essentially proceeds in three steps. First, the two graphs H_i ($i = 1, 2$) are extended by adding a vertex (S_i, v) for every $S_i \in \mathcal{S}_i$, $v \in S_i$ which is connected to the vertex S_i. Moreover, we compute the tree unfoldings T_i of both extended graphs.

Then, in the second step, we compute the set of isomorphisms γ from T_1 to T_2 such that for every $S \in \mathcal{S}_1$ the set $\{S\} \times S$ is mapped to $(\{S\} \times S)^\gamma$ according to the restrictions given by the cosets from the instance $\mathcal{I}$. This is possible since all the sets may be mapped independently of each other. In particular, at this point, identical elements $v \in V_1$ appearing in different sets $S, S' \in \mathcal{S}_1$ may be mapped to different elements $v', v'' \in V_2$.

To resolve this problem we use the coloring on the pairs of leaves of T_1 and T_2 to encode which elements in the tree unfolding correspond to identical elements in the original graph. The set of isomorphisms respecting these additional constraints can be computed using Corollary 11. ◀

Looking at the properties of the tree decompositions computed in Theorem 5 and 8 we have for every node t that either the adhesion sets to the children are all equal or they are all distinct. Up to this point we have only considered the problem that all adhesion sets are distinct (i.e. the coset-hypergraph-isomorphism problem). Next we consider the case that all adhesion sets are equal. Towards this end we define the following variant.

An instance of *multiple-colored-coset-isomorphism* is a 6-tuple $\mathcal{I} = (V_1, V_2, \chi_1, \chi_2, \mathcal{F}, \mathfrak{f})$ such that

1. $\chi_i \colon [t] \to \mathbb{N}$ is a coloring,
2. $\mathcal{F} = \{\Theta_i \leq \operatorname{Sym}(V) \mid i \in [t]\}$, and
3. $\mathfrak{f} = \{\tau_{i,j} \colon V_1 \to V_2 \mid i, j \in [t] \text{ such that } \chi_1(i) = \chi_2(j)\}$ such that
   a. every $\tau_{i,j} \in \mathfrak{f}$ is bijective, and
   b. for every color $i \in \mathbb{N}$ and every $j_1, j_1' \in \chi_1^{-1}(i)$ and $j_2, j_2' \in \chi_2^{-1}(i)$ and $\theta \in \Theta_{j_1}, \theta' \in \Theta_{j_1'}$ it holds that

$$\theta \tau_{j_1, j_2} \tau_{j_1', j_2}^{-1} \theta' \tau_{j_1', j_2'} \in \Theta_{j_1} \tau_{j_1, j_2'}. \tag{N2}$$

The instance is *solvable* if there is a bijective mapping $\phi\colon V_1 \to V_2$ and a $\pi \in \mathrm{Sym}(t)$ such that

1. $\chi_1(i) = \chi_2(\pi(i))$ for all $i \in [t]$, and
2. for every $i \in [t]$ it holds that $\phi \in \Theta_i \tau_{i,\pi(i)}$.

The set $\mathrm{Iso}(\mathcal{I})$ is defined analogously.

▶ **Lemma 12.** *Let $\mathcal{I} = (V_1, V_2, \chi_1, \chi_2, \mathcal{F}, \mathfrak{f})$ be an instance of multiple-colored-coset-isomorphism. Then a representation for the set $\mathrm{Iso}(\mathcal{I})$ can be computed in time*

$$\min((|V_1|!|\mathcal{F}|)^{\mathcal{O}(1)}, |\mathcal{F}|!^{\mathcal{O}(1)} 2^{\mathcal{O}((\log |V_1|)^c)}).$$

6 The isomorphism and the canonization algorithm

6.1 The isomorphism test

Before describing the main algorithm we need to state the following auxiliary lemma.

▶ **Lemma 13.** *Let $G = (D, E)$ be a graph of tree width at most k and let H be a rooted graph such that $D \subseteq V(H)$. Then, one can compute an isomorphism-invariant rooted graph H' such that*

1. *H is an induced subgraph of H',*
2. *$\mathrm{fdeg}_{H'}(w) \leq \max\{d, k+1\} + 1$ for all $w \in V(H')$ where $d = \max_{w \in V(H)} \mathrm{fdeg}_H(w)$,*
3. *$\mathrm{depth}(H') \leq \mathrm{depth}(H) + k + 2$, and*
4. *for every clique $C \subseteq D$ there is a corresponding vertex $C \in V(H')$*

in time $2^{\mathcal{O}(k)} \cdot |V(H)|^{\mathcal{O}(1)}$.

Here, isomorphism invariant means that every isomorphism $\phi \in \mathrm{Iso}(H_1, H_2)$, which naturally restricts to an isomorphism from G_1 to G_2, can be extended to an isomorphism from H_1' to H_2'.

▶ **Theorem 14.** *Let $k \in \mathbb{N}$ and let G_1, G_2 be connected graphs. There is an algorithm that either correctly concludes that $\mathrm{tw}(G_1) > k$, or computes the set of isomorphisms $\mathrm{Iso}(G_1, G_2)$ in time $2^{\mathcal{O}(k(\log k)^c)} |V(G)|^{\mathcal{O}(1)}$ for some constant c.*

Proof sketch. The first step is to decompose the given graphs. Notice that the decomposition techniques in Theorem 8 can only be applied to k-basic graphs, i.e. k-improved and clique-separator free. Therefore, we proceed as follows. First, we consider the k-improvement G_1^k and G_2^k of G_1 and G_2, respectively. We build the clique-separator decompositions of the k-improvements using Theorem 5. Secondly, we refine each (k-basic) bag of the decomposition by constructing a labeled decomposition for each bag using Theorem 8. (In fact, we need to fix a vertex in order to apply Theorem 8. For this reason we are just able to construct a family of decompositions in each k-basic bag. But it turns out, that we are also able to handle this. Also note that we use the version of Theorem 8 that runs in time $2^{\mathcal{O}(k \log k)} |V(G)|^{\mathcal{O}(1)}$ and omits the labeling cosets $\alpha(t)$.) The crucial point is that in our final decomposition the size of each bag is bounded by $c_{\mathrm{L}} \in 2^{\mathcal{O}(k \log k)}$ and more importantly each bag is assigned to a graph that restricts possible automorphisms of the corresponding graph structure. More precisely each bag $\beta(t)$ is assigned to a graph $\eta(t)$ of bounded degree such that $V(\eta(t))$ contains the vertices of $\beta(t)$, the edges of $\beta(t)$ and also the adhesion sets of t. In fact, we also need to embed the clique separators of the outer decomposition into the graph $\eta(t)$. However, this can be achieved using Lemma 13.

From now on, we start to compute isomorphisms preserving the decompositions T_1 and T_2 in a bottom up fashion. (In order to compute edge-preserving bijections, we need to consider

the edges of G_1 and G_2 rather than the "improved" edges in G_1^k and G_2^k). The decomposition of both graphs have a root bags denoted t_1 and t_2, respectively. Both roots have children $t_{11}, \dots, t_{1\ell}$ and $t_{21}, \dots, t_{2\ell}$. By T_{ji} we denote the decomposition rooted at t_{ji} and consisting of all bags that are descendants of t_{ji}. By using a dynamic programming approach we assume that the isomorphisms from T_{1i} to $T_{2i'}$ are already computed. To compute the isomorphisms from T_1 to T_2 we use our subroutine, namely coset-hypergraph-isomorphism, which is defined and algorithmically solved in Section 5. For a moment, assume that all adhesion sets of t_1 (and t_2, respectively) are distinct. We restrict the computed isomorphisms from T_{1i} to $T_{2i'}$ to their corresponding adhesions sets of t_1 and t_2, respectively. These restrictions are used to define $\mathcal{F}$ and $\mathfrak{f}$. The edge relation can also be encoded in $\mathcal{F}$ and $\mathfrak{f}$. More precisely, we add each edge $uv \in E(\beta_1(t_1))$ and $u'v' \in E(\beta_2(t_2))$ to $\mathcal{S}_1$ and $\mathcal{S}_2$, and define $\Theta_{uv}\tau_{uv,u'v'}$ as the set of bijections mapping uv to $u'v'$. The colorings χ_1 and χ_2 help to define the right instance, e.g. they distinguish the edges from the adhesion sets in $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Finally, we solve the instance $(\beta_1(t_1), \beta_2(t_2), \mathcal{S}_1, \mathcal{S}_2, \chi_1, \chi_2, \mathcal{F}, \mathfrak{f})$ with the algorithm from Lemma 9. In case all adhesion sets are equal we simply use Lemma 12 instead. ◀

▶ Remark. The degree of the polylogarithmic term (i.e. the constant c) in the running time of the previous theorem is related to the corresponding constant in Babai's quasipolynomial time isomorphism test [1]. Since the constant that Babai's algorithm achieves is not specified in [1], we also do not specify the constant.

6.2 Canonization

We briefly describe how to adapt our techniques to obtain an algorithm which computes a canonization for a given graph of tree width of at most k. Since Babai's quasipolynomial time algorithm [1] only tests isomorphism of two given input graphs and can not be used for canonization purposes, we need to replace the methods introduced in Section 5. To achieve this we still use group theoretic techniques, but compared to isomorphism the machinery is quite lengthy and technical.

▶ **Theorem 15.** *There is a graph canonization algorithm running in time* $2^{\mathcal{O}(k^2 \log k)} \operatorname{poly}(n)$*, where* n *is the size and* k *the tree width of the input graph.*

Proof sketch. The basic approach for the canonization algorithm is very similar to the approach presented in Theorem 14. One of the main differences is how the algorithm gets its insight into the structure of the bags of the decomposition of the k-basic graphs. In the isomorphism algorithm the graph $\eta(t)$ serves as a tool to exploit the structure of each bag t. For the canonization algorithm we instead use the fact that each bag can be guarded with a labeling coset $\alpha(t)$ of bounded composition-width.

We then define the corresponding variant of the coset-hypergraph-isomorphism problem suited for canonization. For the algorithm we assume that, instead of graphs of small degree and depth, we get a labeling coset of composition width at most $k+1$. This problem can then be solved in time $n^{\mathcal{O}(k)}$ using the group theoretic techniques from [2, 12, 15] where n denotes the number of vertices (i.e. the number of vertices in a single bag in the application in the main canonization algorithm). Since the bag size is bounded by $2^{\mathcal{O}(k \log k)}$ this gives the overall running time of $2^{\mathcal{O}(k^2 \log k)} \operatorname{poly}(n)$. ◀

References

1. László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.
2. László Babai and Eugene M. Luks. Canonical labeling of graphs. In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 171–183. ACM, 1983. doi:10.1145/800061.808746.
3. Hans L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *J. Algorithms*, 11(4):631–643, 1990. doi:10.1016/0196-6774(90)90013-5.
4. Michael Elberfeld and Pascal Schweitzer. Canonizing graphs of bounded tree width in logspace. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 32:1–32:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. URL: http://www.dagstuhl.de/dagpub/978-3-95977-001-9, doi:10.4230/LIPIcs.STACS.2016.32.
5. Martin Grohe and Dániel Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. *SIAM J. Comput.*, 44(1):114–159, 2015. doi:10.1137/120892234.
6. Martin Grohe, Daniel Neuen, and Pascal Schweitzer. A faster isomorphism test for graphs of small degree. *CoRR*, abs/1802.04659, 2018. arXiv:1802.04659.
7. Martin Grohe and Pascal Schweitzer. Isomorphism testing for graphs of bounded rank width. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1010–1029. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.66.
8. Hanns-Georg Leimer. Optimal decomposition by clique separators. *Discrete Mathematics*, 113(1-3):99–123, 1993. doi:10.1016/0012-365X(93)90510-Z.
9. Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. *SIAM J. Comput.*, 46(1):161–189, 2017. doi:10.1137/140999980.
10. Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982. doi:10.1016/0022-0000(82)90009-5.
11. Rudolf Mathon. A note on the graph isomorphism counting problem. *Inf. Process. Lett.*, 8(3):131–132, 1979. doi:10.1016/0020-0190(79)90004-8.
12. Gary L. Miller. Isomorphism testing and canonical forms for k-contractable graphs (A generalization of bounded valence and bounded genus). In Marek Karpinski, editor, *Fundamentals of Computation Theory, Proceedings of the 1983 International FCT-Conference, Borgholm, Sweden, August 21-27, 1983*, volume 158 of *Lecture Notes in Computer Science*, pages 310–327. Springer, 1983. doi:10.1007/3-540-12689-9_114.
13. Daniel Neuen. Graph isomorphism for unit square graphs. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 70:1–70:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. URL: http://drops.dagstuhl.de/opus/portals/lipics/index.php?semnr=16013, doi:10.4230/LIPIcs.ESA.2016.70.
14. Yota Otachi and Pascal Schweitzer. Reduction techniques for graph isomorphism in the context of width parameters. In R. Ravi and Inge Li Gørtz, editors, *Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July*

2-4, 2014. Proceedings, volume 8503 of *Lecture Notes in Computer Science*, pages 368–379. Springer, 2014. doi:10.1007/978-3-319-08404-6_32.

15 Ilia N. Ponomarenko. The isomorphism problem for classes of graphs closed under contraction. *Journal of Mathematical Sciences*, 55(2):1621–1643, 1991.

A Polynomial-Time Approximation Algorithm for All-Terminal Network Reliability

Heng Guo
School of Informatics, University of Edinburgh, Informatics Forum, Edinburgh, EH8 9AB, United Kingdom.
hguo@inf.ed.ac.uk
https://orcid.org/0000-0001-8199-5596

Mark Jerrum[1]
School of Mathematical Sciences, Queen Mary, University of London, Mile End Road, London, E1 4NS, United Kingdom.
m.jerrum@qmul.ac.uk
https://orcid.org/0000-0003-0863-7279

Abstract

We give a fully polynomial-time randomized approximation scheme (FPRAS) for the all-terminal network reliability problem, which is to determine the probability that, in a undirected graph, assuming each edge fails independently, the remaining graph is still connected. Our main contribution is to confirm a conjecture by Gorodezky and Pak (*Random Struct. Algorithms*, 2014), that the expected running time of the "cluster-popping" algorithm in bi-directed graphs is bounded by a polynomial in the size of the input.

2012 ACM Subject Classification Theory of computation → Generating random combinatorial structures

Keywords and phrases Approximate counting, Network Reliability, Sampling, Markov chains

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.68

Related Version Also available at `https://arxiv.org/abs/1709.08561`.

Acknowledgements We thank Mark Huber for bringing reference [8] to our attention, Mark Walters for the coupling idea leading to Lemma 12, and Igor Pak for comments on an earlier version. We also thank the organizers of the "LMS – EPSRC Durham Symposium on Markov Processes, Mixing Times and Cutoff", where part of the work is carried out.

1 Introduction

Network reliability problems are extensively studied #**P**-hard problems [5] (see also [3, 22, 18, 2]). In fact, these problems are amongst the first of those shown to be #**P**-hard, and the two-terminal version is listed in Valiant's original thirteen [24]. The general setup is that in a given (undirected or directed) graph, every edge (or arc) e has an independent probability p_e to fail, and we are interested in various kinds of connectivity notions of the remaining graph. For example, the two-terminal connectedness [24] asks for the probability that for two vertices s and t, s is connected to t in the remaining graph, and the (undirected) all-terminal network reliability asks for the probability of all vertices being connected after

[1] The work described here was supported by the EPSRC research grant EP/N004221/1 "Algorithms that Count".


edges fail. The latter can also be viewed as a specialization of the Tutte polynomial $T_G(x, y)$ with $x = 1$ and $y > 1$, yet another classic topic whose computational complexity is extensively studied [12, 25, 6, 7].

Prior to our work, the approximation complexity of network reliability problems remained elusive despite their importance. There is no known efficient approximation algorithm (for any variant), but nor is there any evidence that such an algorithm does not exist. A notable exception is Karger's fully polynomial-time randomized approximation scheme (FPRAS) for (undirected) all-terminal network *unreliability* [15] (see also [11, 16, 17] for more recent developments). Although approximating unreliability is potentially more useful in practice, it does not entail an approximation of its complement.

In this paper, we give an FPRAS for the all-terminal network reliability problem, defined below and denoted Reliability.

Name Reliability
Instance A (undirected) graph $G = (V, E)$, and failure probabilities $\mathbf{p} = (p_e)_{e \in E}$.
Output $Z_{rel}(G; \mathbf{p})$, which is the probability that if each edge e fails with probability p_e, the remaining graph is connected.

When p_e is independent of e, Reliability is an evaluation of the Tutte polynomial. The *Tutte polynomial* is a two-variable polynomial $T_G(x, y)$ associated with a graph G, which encodes much interesting information about G. As (x, y) ranges over $\mathbb{R}^2$ or $\mathbb{C}^2$ we obtain a family of graph parameters, the so-called *Tutte plane*. As already noted, the study of the computational complexity of these parameters has a long history. Reliability with a uniform failure probability $0 < p < 1$ is equivalent to evaluating the Tutte polynomial $T_G(x, y)$ on the line $x = 1$ and $y = \frac{1}{p} > 1$. Our algorithm is the first positive result on the complexity of the Tutte plane since Jerrum and Sinclair presented an FPRAS for the partition function of the ferromagnetic Ising model, which is equivalent to the Tutte polynomial on the positive branch of the hyperbola $(x - 1)(y - 1) = 2$ [14]. It also answers a well-known open problem from 1980s, when the #**P**-hardness of Reliability was established [13, 22] and the study of approximate counting initiated. This problem is explicitly proposed in, for example, [26, Conjecture 8.7.11] and [15]. We note that many conjectures by Welsh ([26, Chapter 8.7] and [27]) remain open, and we hope that our work is only a beginning to answering these questions.

Another related and important reliability measure is *reachability*, introduced and studied by Ball and Provan [3]. A directed graph $G = (V, A)$ with a distinguished root r is said to be *root-connected* if all vertices can reach r. Reachability, denoted $Z_{reach}(G, r; \mathbf{p})$ for failure probabilities $\mathbf{p} = (p_e)_{e \in A}$, is the probability that, if each arc e fails with probability p_e independently, the remaining graph is still root-connected.

We define the computational problem formally.

Name Reachability
Instance A directed graph $G = (V, A)$ with root r, and failure probabilities $\mathbf{p} = (p_e)_{e \in A}$.
Output $Z_{reach}(G, r; \mathbf{p})$.

Exact polynomial-time algorithms are known when the graph is acyclic [3] or has a small number of cycles [10]. However, in general the problem is #**P**-hard [22].

Ball [1] showed that Reliability is equivalent to Reachability in bi-directed graphs. A bi-directed[2] graph is one where every arc has an anti-parallel twin with the same failure probability. It is shown [1] that $Z_{rel}(G;\mathbf{p}) = Z_{reach}(\overrightarrow{G}, r; \mathbf{p}')$, where $\overrightarrow{G}$ and $\mathbf{p}'$ are obtained by replacing every undirected edge in G with a pair of anti-parallel arcs having the same failure probability in either direction, and r is chosen arbitrarily. See Lemma 12.

Our FPRAS for Reliability utilizes this equivalence via approximating Reachability in bi-directed graphs. The core ingredient is the "cluster-popping" algorithm introduced by Gorodezky and Pak [8]. The goal is to sample root-connected subgraphs with probability proportional to their weights, and then the reduction from counting to sampling is via a sequence of contractions. A cluster is a subset of vertices not including the root and without any out-going arc. The sampling algorithm randomizes all arcs independently, and then repeatedly resamples arcs going out from minimal clusters until no cluster is left, at which point the remaining subgraph is guaranteed to be root-connected. This approach is similar to Wilson's "cycle-popping" algorithm [28] for rooted spanning trees, and to the "sink-popping" algorithm [4] for sink-free orientations. Gorodezky and Pak [8] have noted that cluster-popping can take exponential time in general, but they conjectured that in bi-directed graphs, the algorithm runs within polynomial-time.

We confirm this conjecture. Let $p_{\max}$ be the maximum failure probability of edges (or arcs). Let m be the number of edges (or arcs) and n the number of vertices.

▶ **Theorem 1.** *There is an FPRAS for* Reliability *(or equivalently,* Reachability *in bi-directed graphs). The expected running time is* $O\left(\varepsilon^{-2} p_{\max}(1-p_{\max})^{-3} m^2 n^3\right)$ *for an* $(1 \pm \varepsilon)$*-approximation. There is also an exact sampler to draw (edge-weighted) connected subgraphs with expected running time at most* $p_{\max}(1-p_{\max})^{-1} m^2 n$.

We analyze the "cluster-popping" algorithm [8] under the *partial rejection sampling* framework [9], which is a general approach to sampling from a product distribution conditioned on avoiding a number of "bad" events. Partial rejection sampling is inspired by the Moser-Tardos algorithm for the Lovász Local Lemma [20]. It starts with randomizing all variables independently, and then gradually eliminating "bad" events. At every step, we need to find an appropriate set of variables to resample. We call an instance *extremal* [19, 23], if any two bad events are either disjoint or independent. For extremal instances, the resampling set can be simply chosen to be the set of all variables involved in occurring bad events [9], and the algorithm becomes exactly the same as the Moser-Tardos resampling algorithm [20]. In particular, all three "popping" algorithms [28, 4, 8] are special cases of partial rejection sampling for extremal instances. In case of "cluster-popping", the bad events are exactly minimal clusters.

The advantage of the partial rejection sampling treatment is that we have an explicit formula for the expected number of resampling events for any extremal instance [19, 9], which equals to the ratio between the probability of having exactly one bad event and the probability of avoiding all bad events. In order to bound this ratio, we use a combinatorial encoding idea and design a mapping from subgraphs with a unique minimal cluster to root-connected subgraphs. To make this mapping injective, we record an extra vertex and an arc so that we can recover the pre-image. This extra cost is upper-bounded by a polynomial in the size of the graph.

[2] There are other definitions of "bi-directed graphs" in the literature. Our definition is sometimes also called a symmetric directed graph.

Cluster-popping only draws root-connected subgraphs in the bi-directed setting. In order to sample connected subgraphs in the undirected setting, we provide an alternative proof of the equivalence between Reliability and Reachability in bi-directed graphs, which essentially is a coupling argument. This coupling has a new consequence that, once we have a sample of a root-connected subgraph, it is easy to generate a connected subgraph according to the correct distribution.

In Section 2 we introduce the "cluster-popping" algorithm and the partial rejection framework. In Section 3 we analyze its running time in bi-directed graphs. For completeness, in Section 4 we include the approximate counting algorithm due to Gorodezky and Pak [8]. In Section 5 we give a coupling proof of the equivalence between Reliability and Reachability in bi-directed graphs. In Section 6 we conclude by mentioning a few open problems.

2 Cluster-popping

Let $G = (V, A)$ be a directed[3] graph with root r. The graph G is called *root-connected* if there is a directed path in G from every non-root vertex to r. Let $0 < p_e < 1$ be the failure probability of arc e, and define the weight of a subgraph S to be $\mathrm{wt}(S) := \prod_{e\in S}(1-p_e)\prod_{e\notin S} p_e$. Then reachability, $Z_{reach}(G, r; \mathbf{p})$, is defined as follows,

$$Z_{reach}(G, r; \mathbf{p}) := \sum_{\substack{S\subseteq A \\ (V,S) \text{ is root-connected}}} \mathrm{wt}(S).$$

Here, $\mathbf{p} = (p_e : e \in A)$ denotes the vector of failure probabilities.

Let $\pi_G(\cdot)$ (or $\pi(\cdot)$ for short) be the distribution resulting from choosing each arc e independently with probability $1 - p_e$, and conditioning on the resulting graph being root-connected. In other words, the support of $\pi(\cdot)$ is the collection of all root-connected subgraphs, and the probability of each subgraph S is proportional to its weight $\mathrm{wt}(S)$. Then $Z_{reach}(G, r; \mathbf{p})$ is the normalizing factor of the distribution $\pi(\cdot)$. Gorodezky and Pak [8] have shown that approximating $Z_{reach}(G, r; \mathbf{p})$ can be reduced to sampling from $\pi(\cdot)$ when the graph is bi-directed.

The "cluster-popping" algorithm of Gorodezky and Pak [8], to sample root-connected subgraphs from $\pi(\cdot)$, can be viewed as a special case of partial rejection sampling [9] for extremal instances. With every arc e of G we associate a random variable that records whether that arc has failed. Bad events are characterized by the following notion of clusters.

▶ **Definition 2.** In a directed graph (V, A) with root r, a subset $C \subseteq V$ of vertices is called a *cluster* if $r \notin C$ and there is no arc $u \to v \in A$ such that $u \in C$ and $v \notin C$.

We say C is a *minimal cluster* if C is a cluster and for any proper subset $C' \subset C$, C' is not a cluster.

If (V, A) contains no cluster, then it is root-connected. For each vertex v, let $A_{\mathrm{out}}(v)$ be the set of outgoing arcs from v. We also abuse the notation to write $A_{\mathrm{out}}(S) = \bigcup_{v\in S} A_{\mathrm{out}}(v)$ for a subset $S \subset V$ of vertices. Notice that $A_{\mathrm{out}}(S)$ contains edges between vertices inside S. To "pop" a cluster C, we re-randomize all arcs in $A_{\mathrm{out}}(C)$. However, re-randomizing clusters does not yield the desired distribution. We will instead re-randomize minimal clusters.

[3] It is easy to see that in a undirected graph, reachability is the same as all-terminal reliability.

Algorithm 1 Cluster Popping.

Let S be a subset of arcs by choosing each arc e with probability $1 - p_e$ independently.
while There is a cluster in (V, S). **do**
  Let $C_1, \ldots, C_k$ be all minimal clusters in (V, S), and $C = \bigcup_{i=1}^{k} C_i$.
  Re-randomize all arcs in $A_{\text{out}}(C)$ to get a new S.
end while
return S

▶ **Claim 3.** *Any minimal cluster is strongly connected.*

Proof. Let C be a minimal cluster, and $v \in C$ be an arbitrary vertex in C. We claim that v can reach all vertices of C. If not, let C' be the set of reachable vertices of v and $C' \subsetneq C$. Since C' does not have any outgoing arcs, C' is a cluster. This contradicts to the minimality of C. ◀

▶ **Claim 4.** *If C_1 and C_2 are two distinct minimal clusters, then $C_1 \cap C_2 = \emptyset$.*

Proof. By Claim 3, C_1 and C_2 are both strongly connected components. If $C_1 \cap C_2 \neq \emptyset$, then they must be identical. ◀

For every subset $C \subseteq V$ of vertices, we define a bad event B_C, which occurs if C is a minimal cluster. Observe that B_C relies only on the status of arcs in $A_{\text{out}}(C)$. Thus, if $C_1 \cap C_2 = \emptyset$, then B_{C_1} and B_{C_2} are independent, even if some of their vertices are adjacent. By Claim 4, we know that two bad events B_{C_1} and B_{C_2} are either independent or disjoint. Thus the aforementioned extremal condition is met. Moreover, it was shown [9, Theorem 8] that if the instance is extremal, then at every step, we only need to resample variables involved in occurring bad events. This leads to the cluster-popping algorithm of Gorodezky and Pak [8], which is formally described in Algorithm 1.

The correctness of Algorithm 1 is first shown by Gorodezky and Pak [8]. It can also be easily verified using [9, Theorem 8].

▶ **Theorem 5** ([8, Theorem 2.2])**.** *The output of Algorithm 1 is drawn from π_G.*

An advantage of thinking in the partial rejection sampling framework is that we have a closed form formula for the expected running time of these algorithms on extremal instances. Let Ω_k be the collection of subgraphs with k minimal clusters, and $Z_k := \sum_{S \in \Omega_k} \text{wt}(S)$. Then $Z_0 = Z_{reach}(G, r; \mathbf{p})$, since any subgraph in Ω_0 has no cluster and is thus root-connected.

▶ **Theorem 6** ([9])**.** *Let T be the number of resampled events of the partial rejection sampling algorithm for extremal instances. Then $\mathbb{E}\, T = \frac{Z_1}{Z_0}$. In particular, for Algorithm 1, T is the number of popped clusters.*

The less-than-or-equal-to direction of Theorem 6 was shown by Kolipaka and Szegedy [19], which is the direction we will need later. The other direction is useful to show running-time lower bounds, but that is not our focus in this paper.

3 Running time of Algorithm 1 in bi-directed graphs

Gorodezky and Pak [8] have given examples of directed graphs in which Algorithm 1 requires exponential time. In the following we focus on bi-directed graphs. A graph G is called *bi-directed* if $u \to v$ is present in G, then $v \to u$ is present in G as well, and the failure

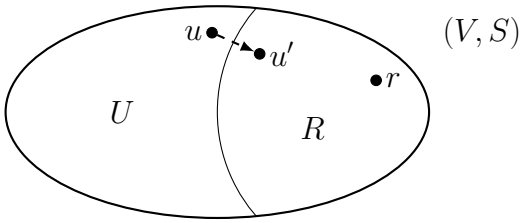


Figure 1 An illustration of R, U, and $u \to u'$.

probabilities are the same for these two arcs. We use Bi-directed Reachability to denote Reachability in bi-directed graphs. For an arc $e = u \to v$, let $\overline{e} := v \to u$ denote its reverse arc. Then in a bi-directed graph, $p_e = p_{\overline{e}}$.

▶ **Lemma 7.** *Let $G = (V, A)$ be a root-connected bi-directed graph with root r. We have that $Z_1 \le \max_{e \in A} \left\{ \frac{p_e}{1-p_e} \right\} mnZ_0$, where $n = |V|$, and $m = |A|$.*

Proof. We construct an injective mapping $\varphi : \Omega_1 \to \Omega_0 \times V \times A$. For each subgraph $S \in \Omega_1$, $\varphi(S)$ is defined by "repairing" S so that no minimal cluster is present. We choose in advance an arbitrary ordering of vertices and arcs. Let C be the unique minimal cluster in S and v be the first vertex in C. Let R denote the set of all vertices which can reach the root r in the subgraph S. Since $S \in \Omega_1$, $R \neq V$. Let $U = V \setminus R$. Since G is root-connected, there is an arc in A from U to R. Let $u \to u'$ be the first such arc, where $u \in U$ and $u' \in R$. We let

$$\varphi(S) := (S_{\text{fix}}, v, u \to u'),$$

where $S_{\text{fix}} \in \Omega_0$ is defined next. Figure 1 is an illustration of these objects.

Consider the subgraph $H = (U, S[U])$, where

$$S[U] := \{x \to y \mid x \in U,\ y \in U,\ x \to y \in S\}.$$

We consider the directed acyclic graph (DAG) of strongly connected components of H, and call it $\widehat{H}$. (We use the decoration $\widehat{\ }$ to denote arcs, vertices, etc. in $\widehat{H}$.) To be more precise, we replace each strongly connected component by a single vertex. For a vertex $w \in U$, let $[w]$ denote the strongly connected component containing w. For example, $[v]$ is the same as the minimal cluster C by Claim 3. We may also view $[w]$ as a vertex in $\widehat{H}$ and we do not distinguish the two views. The arcs in $\widehat{H}$ are naturally induced by $S[U]$. Namely, for $[x] \neq [y]$, an arc $[x] \to [y]$ is present in $\widehat{H}$ if there exists $x' \in [x]$, $y' \in [y]$ such that $x' \to y' \in S$.

We claim that $\widehat{H}$ is root-connected with root $[v]$. This is because $[v]$ must be the unique sink in $\widehat{H}$ and $\widehat{H}$ is acyclic. If there is another sink $[w]$ where $v \notin [w]$, then $[w]$ is a minimal cluster in H. This contradicts $S \in \Omega_1$.

Since $\widehat{H}$ is root-connected, there is at least one path from $[u]$ to $[v]$. Let $\widehat{W}$ denote the set of vertices of $\widehat{H}$ that can be reached from $[u]$ in $\widehat{H}$ (including $[u]$), and $W := \{x \mid [x] \in \widehat{W}\}$. Then W is a cluster and $[u]$ is the unique source in $\widehat{H}[\widehat{W}]$. As $\widehat{H}$ is root-connected, $[v] \in \widehat{W}$. Define

$$S_{\text{flip}} := \left\{x \to y \;\middle|\; [x] \neq [y],\ x, y \in W, \text{ and } x \to y \in S\right\},$$

which is the set of edges to be flipped. Notice that $S[W]$ is different from S_{flip}, namely all arcs that are inside strongly connected components are ignored in S_{flip}. Now we are ready to define S_{fix}. We reverse all arcs in S_{flip} and add the arc $u \to u'$ to fix the minimal cluster.

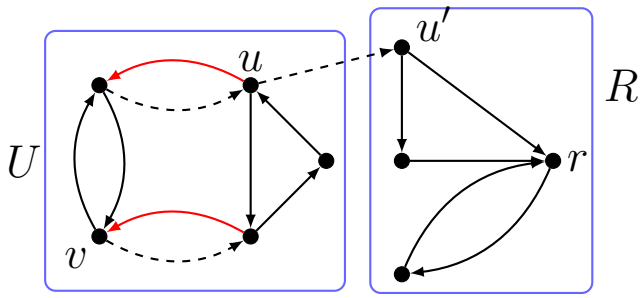


Figure 2 An example of S_{flip} (red arcs) in the subgraph (V, S). Dashed arcs are to be added to S_{fix}. The underlying graph has more arcs than are drawn here.

Formally, let

$$S_{\text{fix}} := S \cup \{u \to u'\} \cup \{y \to x \mid x \to y \in S_{\text{flip}}\} \setminus S_{\text{flip}}.$$

Figure 2 is an example of these objects we defined.

Let $\widehat{H}_{\text{fix}}$ be the graph obtained from $\widehat{H}$ by reversing all arcs induced by S_{flip}. Observe that $[u]$ becomes the unique sink in $\widehat{H}_{\text{fix}}[\widehat{W}]$ (and $[v]$ becomes the unique source).

We verify that $S_{\text{fix}} \in \Omega_0$. For any $x \in R$, x can still reach r in (V, S_{fix}) since the path from x to r in (V, S) is not changed. Since $u \to u' \in S_{\text{fix}}$, u can reach $u' \in R$ and hence r. For any $y \in W$, y can reach u as $[u]$ is the unique sink in $\widehat{H}_{\text{fix}}[\widehat{W}]$. For any $z \in U \setminus W$, z can reach $v \in W$ since the path from z to v in (V, S) is not changed.

Next we verify that φ is injective. To do so, we show that we can recover S given S_{fix}, $u \to u'$, and v. First remove $u \to u'$ from S_{fix}. The set of vertices which can reach r in $(V, S_{\text{fix}} \setminus \{u \to u'\})$ is exactly R in (V, S). Namely we can recover U and R. As a consequence, we can recover all arcs in S that are incident with R, as these arcs are not changed.

What is left to do is to recover arcs in $S[U]$. To do so, we need to find out which arcs have been flipped. We claim that $\widehat{H}_{\text{fix}}$ is acyclic. Suppose there is a cycle in $\widehat{H}_{\text{fix}}$. Since $\widehat{H}$ is acyclic, the cycle must involve flipped arcs and thus vertices in $\widehat{W}$. Let $[x] \in \widehat{W}$ be the lowest one under the topological ordering of $\widehat{H}[\widehat{W}]$. Since $\widehat{W}$ is a cluster, the outgoing arc $[x] \to [y]$ along the cycle in $\widehat{H}_{\text{fix}}$ must have been flipped, implying that $[y] \in \widehat{W}$ and $[y] \to [x]$ is in $\widehat{H}[\widehat{W}]$. This contradicts to the minimality of $[x]$.

Since $\widehat{H}_{\text{fix}}$ is acyclic, the strongly connected components of $H_{\text{fix}} := (U, S_{\text{fix}}[U])$ are identical to those of $H = (U, S[U])$. Hence contracting all strongly connected components of H_{fix} results in exactly $\widehat{H}_{\text{fix}}$. All we need to recover now is the set $\widehat{W}$. Let $\widehat{W}'$ be the set of vertices reachable from $[v]$ in $\widehat{H}_{\text{fix}}$. It is easy to see that $\widehat{W} \subseteq \widehat{W}'$. We claim that actually $\widehat{W} = \widehat{W}'$. For any $[x] \in \widehat{W}'$, there is a path from $[v]$ to $[x]$ in $\widehat{H}_{\text{fix}}$. Suppose $[x] \notin \widehat{W}$. Since $[v] \in \widehat{W}$, we may assume that $[y]$ is the first vertex along the path such that $[y] \to [z]$ where $[z] \notin \widehat{W}$. Thus $[y] \to [z]$ has not been flipped and is present in $\widehat{H}$. However, this contradicts the fact that $\widehat{W}$ is a cluster in $\widehat{H}$.

To summarize, given S_{fix}, $u \to u'$, and v, we may uniquely recover S. Hence the mapping φ is injective. Moreover, flipping arcs does not change the weight as $p_e = p_{\overline{e}}$, and only adding the arc $u \to u'$ would. We have that $\text{wt}(S_{\text{fix}}) = \frac{1-p_{u\to u'}}{p_{u\to u'}}\text{wt}(S)$. The lemma follows. ◀

We remark that an alternative way of repairing S in the proof above is to reverse all arcs in $S[W]$ without defining S_{flip}. The key point is that doing so leaves the strongly connected components intact. However this makes the argument less intuitive.

Let $p_{\max} = \max_{e \in A} p_e$. Combining Theorem 6 and Lemma 7, we have the following theorem. Notice that for each popping, we resample only a subset of arcs.

▶ **Theorem 8.** *Let T be the expected number of popped clusters in Algorithm 1. For a root-connected bi-directed graph $G = (V, A)$, $\mathbb{E}\, T \le \frac{p_{\max}}{1-p_{\max}} mn$, where $n = |V|$, and $m = |A|$. The expected running time is at most $\frac{p_{\max}}{1-p_{\max}} m^2 n$.*

4 Approximate counting

We include the approximate counting algorithm of Gorodezky and Pak [8] for completeness. Let $G = (V, A)$ be an instance of Bi-directed Reachability with root r and parameters $\mathbf{p}$. We construct a sequence of graphs $G_0, .., G_{n-1}$ where $n = |V|$ and $G_0 = G$. Given G_{i-1}, choose two arbitrary adjacent vertices u_i and v_i, remove all arcs between u_i and v_i (in either direction), and identify u_i and v_i to get $G_i = (V_i, A_i)$. Namely we contract all arcs between u_i and v_i, but parallel arcs in the resulting graph are preserved. If one of u_i and v_i is r, the new vertex is labelled r. Thus $G_{n-1} = (\{r\}, \emptyset)$. Since A_i is always a subset of A, we denote by $\mathbf{p}_i$ the parameters $\mathbf{p}$ restricted to A_i.

For $i = 1, \ldots, n-1$, define a random variable R_i as follows:

$$R_i := \begin{cases} 1 & (V_{i-1}, S_{i-1}) \text{ is root-connected in } G_{i-1}; \\ 0 & \text{otherwise,} \end{cases}$$

where $S_{i-1} \subset A_{i-1}$ is a random root-connected subgraph drawn from the distribution $\pi_{G_i}(\cdot)$, together with all arcs e between u_i and v_i added independently with probability $1 - p_e$. It is easy to see that

$$\mathbb{E}\, R_i = \frac{Z_{reach}(G_{i-1}, r; \mathbf{p}_{i-1})}{Z_{reach}(G_i, r; \mathbf{p}_i)}, \quad \text{and} \quad Z_{reach}(G, r; \mathbf{p}) = \prod_{i=1}^{n-1} \mathbb{E}\, R_i.$$

Let $p_{\max} = \max_{e \in A} p_e$ and $s = \lceil 5(1-p_{\max})^{-2}(n-1)\varepsilon^{-2} \rceil$ where s is the desired precision. We estimate $\mathbb{E}\, R_i$ by the empirical mean of s independent samples of Z_i, denoted by $\widetilde{R}_i$, and let $\widetilde{Z} = \prod_{i=1}^{n-1} \widetilde{R}_i$ and $Z = Z_{reach}(G, r; \mathbf{p})$. Gorodezky and Pak [8] showed the following.

▶ **Proposition 9** ([8, Section 9])**.** $\Pr\left(\left|Z - \widetilde{Z}\right| > \varepsilon Z\right) \le 1/4$.

In order to sample Z_i, we use Algorithm 1 to draw independent samples of root-connected subgraphs. Theorem 8 implies that each sample takes at most $\frac{p_{\max}}{1-p_{\max}} m^2 n$ time in expectation. We need $O\left(\frac{n}{\varepsilon^2 (1-p_{\max})^2}\right)$ samples for each Z_i. Putting everything together, we obtain the following theorem.

▶ **Theorem 10.** *There is an FPRAS for* Bi-directed Reachability*. The expected running time is $O\left(\varepsilon^{-2} p_{\max} (1-p_{\max})^{-3} m^2 n^3\right)$ for an $(1 \pm \varepsilon)$-approximation.*

A natural question is what if $1 - p_{\max}$ is close to 0. Intuitively, this means that some arc is very likely to fail. We note that, if $1 - p_e = O(n^{-3})$ for every arc e, then with high probability, sampling from the distribution $\pi(\cdot)$ yields a rooted spanning tree (with probability proportional to its weight). Thus, in this case, we can approximate $\pi(\cdot)$ by an efficient rooted spanning tree sampler, for example, the cycle-popping algorithm [28] (which runs in time $O(mn)$ in expectation).

5 Coupling between reliability and bi-directed reachability

In this section, we give an alternative proof of Ball's equivalence between Reliability and Bi-directed Reachability [1, Corollary 1]. Our proof constructs a coupling, between the (edge-weighted) distribution of connected subgraphs in the undirected setting, and the (edge-weighted) distribution of root-connected subgraphs in the bi-directed setting. This coupling, together with Algorithm 1, yields an efficient exact sampler for connected subgraphs.

We use $\{u, v\}$ to denote an undirected edge, and (u, v) or (v, u) to denote a directed one (namely an arc). Let $G = (V, E)$ be an undirected graph, and $\mathbf{p} = (p_e)_{e\in E}$ be a vector of failure probabilities. Let $\overrightarrow{G} = (V, A)$ be the bi-directed graph obtained by replacing every edge in G with a pair of anti-parallel arcs. Namely, $A = \{(u, v), (v, u) \mid \{u, v\} \in E\}$. Moreover, let $p_{(u,v)} = p_{(v,u)} = p_{\{u,v\}}$ and denote these failure probabilities by $\mathbf{p}'$. For $S \subseteq E$ or $S \subseteq A$, let $\mathrm{wt}(S) = \prod_{e\in S}(1 - p_e) \prod_{e\notin S} p_e$.

Consider the following coupling between the product distribution over edges of G and the one over arcs of $\overrightarrow{G}$. We reveal edges in a breadth-first search (BFS) fashion in both graphs, from the same "root" vertex r. If an edge $\{u, v\}$ is present in the subgraph of G, we couple it with the arc (u, v) or (v, u), whose direction is pointing towards r in the subgraph of $\overrightarrow{G}$. The arc in the other direction is drawn independently from everything else. The key observation is that to decide the set of vertices that can reach r, at any point, only one direction of a bi-directed edge is useful and the other is irrelevant. One can verify that in the end, the subgraph of G is connected if and only if the subgraph of $\overrightarrow{G}$ is root-connected. We will formalize this intuition next.

Fix an arbitrary ordering of V, which will be used for the exploration, and let the first vertex be a distinguished root r. Let $\mathcal{P}(S)$ denote the power set of S for a set S. Define a mapping $\Phi : \mathcal{P}(E) \to \mathcal{P}(A)$ as follows. For $S \subseteq E$, we explore all vertices that can reach r in (V, S) in a deterministic order, and add arcs to $\Phi(S)$ in the direction towards r. To be more specific, we maintain the set of explored and the set of active vertices, denoted by V_e and V_a, respectively. At the beginning, $V_e = \emptyset$ and $V_a = \{r\}$. Given V_e and V_a, let v be the first vertex (according to the predetermined ordering) in V_a. For all $u \in V \setminus V_e$, if $\{u, v\} \in S$, add (u, v) to $\Phi(S)$ and add u to V_a (u may be in V_a already). Then move v from V_a to V_e. This process ends when all vertices that can reach r in (V, S) are explored. Let σ_S be the arriving order of V_e. We will call σ_S the traversal order. We remark that if $\{u, v\} \in S$ then exactly one of the arcs (u, v) and (v, u) is in $\Phi(S)$, and otherwise neither arc is in $\Phi(S)$.

Strictly speaking, the exploration above is not a BFS (V_a may contain a newly added vertex that is lower in the predetermined ordering than all other older vertices). To perform a BFS we need to in addition maintain a layer ordering, which seems unnecessary. The key properties of the exploration are: 1) all edges incident to the current vertex are processed; 2) V_e is always connected (or root-connected for Ψ below).

Similarly, define $\Psi : \mathcal{P}(A) \to \mathcal{P}(E)$ as follows. For $S' \subseteq A$, we again maintain V_e and V_a, and initialize $V_e = \emptyset$ and $V_a = \{r\}$. Given V_e and V_a, let v be the first vertex in V_a. For all $u \in V \setminus V_e$, if $(u, v) \in S$, add $\{u, v\}$ to $\Psi(S')$ and add u to V_a. Then move v from V_a to V_e. This process ends when all vertices that can reach r in (V, S') are explored. Analogously, let $\sigma_{S'}$ be the arriving order of V_e. We remark that if $(u, v) \notin S'$, and v is visited before u, then $\{u, v\} \notin \Psi(S')$, even in case of $(v, u) \in S'$.

Let $\Omega := \{S \subseteq E \mid (V, S) \text{ is connected}\}$, and $\overrightarrow{\Omega} := \{S \subseteq A \mid (V, S) \text{ is root-connected}\}$. We have the following lemma.

▶ **Lemma 11.** *Let Φ, Ψ, Ω, and $\overrightarrow{\Omega}$ be defined as above. Then the following hold:*

1. *if $S \in \Omega$, then $\Phi(S) \in \overrightarrow{\Omega}$;*
2. *if $S' \in \overrightarrow{\Omega}$, then $\Psi(S') \in \Omega$;*
3. *if $S \in \Omega$, then $\Psi(\Phi(S)) = S$;*
4. *$\Psi(\overrightarrow{\Omega}) = \Omega$;*
5. *for any $S \in \Omega$, $\mathrm{wt}(S) = \sum_{S' \in \Psi^{-1}(S)} \mathrm{wt}(S')$.*

Proof. 1. It is easy to verify that, at any point of the construction of Φ, all vertices in V_e can reach r, in both (V, S) and $(V, \Phi(S))$. If $S \in \Omega$, then $V_e = V$ at the end of Φ. Hence $(V, \Phi(S))$ is root-connected, and $\Phi(S) \in \overrightarrow{\Omega}$.
2. This item is completely analogous to item (1).
3. If $\{u, v\} \in S$ and u is processed first during the exploration, then $(v, u) \in \Phi(S)$. The traversal orderings σ_S and $\sigma_{\Phi(S)}$ are the same. Hence, during the construction of $\Psi(\Phi(S))$, u is still processed first, and $\{v, u\} \in \Psi(\Phi(S))$. On the other hand, if $\{u, v\} \notin S$, then neither (u, v) nor (v, u) is in $\Phi(S)$ and thus $\{u, v\} \notin \Psi(\Phi(S))$.
4. This item is a straightforward consequence of items (1), (2), and (3).
5. By item (3), we have that $\Phi(S) \in \Psi^{-1}(S)$. Let

$$\Phi_c(S) := \left\{(u, v) \mid (u, v) \notin \Phi(S) \text{ and } v < u \text{ in the traversal order } \sigma_{\Phi(S)}\right\}.$$

Note that $\Phi(S) \cup \Phi_c(S)$ covers all unordered pairs of vertices as $S \in \Omega$. Moreover,

$$\prod_{e \in \Phi(S)} (1 - p_e) \prod_{e \in \Phi_c(S)} p_e = \mathrm{wt}(S). \tag{$*$}$$

Call S' consistent with $\Phi(S)$ if $\Phi(S) \subseteq S'$ and $S' \cap \Phi_c(S) = \emptyset$.
We claim that $S' \in \Psi^{-1}(S)$ if and only if S' is consistent with $\Phi(S)$. Suppose S' is not consistent with $\Phi(S)$. Consider the exploration of $\Phi(S)$ and S' in the construction of Ψ simultaneously. Since S' is not consistent with $\Phi(S)$, either $\Phi(S) \setminus S' \neq \emptyset$ or $S' \cap \Phi_c(S) \neq \emptyset$. Let v be the first vertex during the exploration so that there is an arc $(u, v) \in \Phi(S) \setminus S'$, or $(u, v) \in S' \cap \Phi_c(S)$ for some $u \notin V_e$. Since $S \in \Omega$, all vertices will be processed, and such a v must exist. (In the latter case, since $(u, v) \in \Phi_c(S)$, v is active first.) If $(u, v) \in \Phi(S) \setminus S'$, then $\{u, v\} \notin \Psi(S')$ but $\{u, v\} \in \Psi(\Phi(S))$. If $(u, v) \in S' \cap \Phi_c(S)$, $\{u, v\} \notin \Psi(\Phi(S))$ but $\{u, v\} \in \Psi(S')$. In either case, $\Psi(S') \neq \Psi(\Phi(S)) = S$ (by item (3)). On the other hand, if $\Phi(S) \subseteq S'$ and $S' \cap \Phi_c(S) = \emptyset$, then we can trace through the construction of $\Psi(\Phi(S))$ and $\Psi(S')$ to verify that $\Psi(S') = \Psi(\Phi(S)) = S$.
The claim together with $(*)$ implies that

$$\sum_{S' \in \Psi^{-1}(S)} \mathrm{wt}(S') = \sum_{S' \text{ is consistent with } \Phi(S)} \mathrm{wt}(S') = \prod_{e \in \Phi(S)} (1 - p_e) \prod_{e \in \Phi_c(S)} p_e = \mathrm{wt}(S). \qquad \blacktriangleleft$$

▶ **Lemma 12.** $Z_{rel}(G; \mathbf{p}) = Z_{reach}(\overrightarrow{G}, r; \mathbf{p}')$.

Proof. First notice that $Z_{rel}(G; \mathbf{p}) = \sum_{S \in \Omega} \mathrm{wt}(S)$ and $Z_{reach}(\overrightarrow{G}, r; \mathbf{p}') = \sum_{S \in \overrightarrow{\Omega}} \mathrm{wt}(S)$. By item (4) of Lemma 11, $\Psi(\overrightarrow{\Omega}) = \Omega$, implying that $\left(\Psi^{-1}(S)\right)_{S \in \Omega}$ is a partition of $\overrightarrow{\Omega}$. Combining this with item (5) of Lemma 11,

$$\sum_{S \in \Omega} \mathrm{wt}(S) = \sum_{S \in \Omega} \sum_{S' \in \Psi^{-1}(S)} \mathrm{wt}(S') = \sum_{S' \in \overrightarrow{\Omega}} \mathrm{wt}(S').$$

The lemma follows. $\blacktriangleleft$

Lemma 12 is first shown by Ball [1, Corollary 2] via modifying edges one by one. Instead, our proof is essentially a coupling argument and has a new consequence that Algorithm 1 can be used to sample edge-weighted connected subgraphs. Recall our notation $\pi_G(\cdot)$, and generalise it to undirected graphs. Thus, for an undirected (or directed) graph G, $\pi_G(\cdot)$ is the distribution resulting from drawing each edge (or arc) e independently with probability $1 - p_e$, and conditioning on the graph drawn being connected (or root-connected).

▶ **Lemma 13.** *If a random root-connected subgraph S' is drawn from $\pi_{\overrightarrow{G}}(\cdot)$, then $\Psi(S')$ has distribution $\pi_G(\cdot)$.*

Proof. Since $S' \in \overrightarrow{G'}$, by item (2) of Lemma 11, $\Psi(S') \in \Omega$. Moreover, for any $s \in \Omega$,

$$\Pr[\Psi(S') = s] = \sum_{s' \in \Psi^{-1}(s)} \Pr[S' = s'] = \sum_{s' \in \Psi^{-1}(s)} \frac{\mathrm{wt}(s')}{Z_{reach}(\overrightarrow{G}, r; \mathbf{p}')} = \frac{\mathrm{wt}(s)}{Z_{rel}(G; \mathbf{p})} = \pi_G(s),$$

where we used item (5) of Lemma 11 and Lemma 12 in the last line. ◀

There is also a coupling going the reversed direction of Lemma 13, by drawing a random connected subgraph S from $\pi_G(\cdot)$, mapping it to $\Phi(S)$, and excluding all arcs in $\Phi_c(S)$. All other arcs are drawn independently. The resulting S' has distribution $\pi_{\overrightarrow{G}}(\cdot)$. Its correctness is not hard to prove, given Lemma 11, but it is not the direction of use to us and we omit its proof.

Theorem 10 and Lemma 12 imply the counting part of Theorem 1. Theorem 8 and Lemma 13 imply the sampling part of Theorem 1.

6 Concluding remarks

In this paper we give an FPRAS for Reliability (or, equivalently, Bi-directed Reachability), by confirming a conjecture of Gorodezky and Pak [8]. We also give an exact sampler for edge-weighted connected subgraphs with polynomial running time in expectation. The core ingredient of our algorithms is the cluster-popping algorithm to sample root-connected subgraphs, namely Algorithm 1. We manage to analyze it using the partial rejection sampling framework.

Reliability is equivalent to counting weighted connected subgraphs, which is the evaluation of the Tutte polynomial $T_G(x, y)$ for points $x = 1$ and $y > 1$. An interesting question is about the dual of this half-line, namely for points $x > 1$ and $y = 1$, whose evaluation is to count weighted acyclic subgraphs. It is well known that for a planar graph G, $T_G(x, 1) = T_{G^*}(1, x)$ where G^* is the planar dual of G [21]. Hence, Theorem 1 implies that in planar graphs, $T_G(x, 1)$ can be efficiently approximated for $x > 1$. Can we remove the restriction of planar graphs?

Another interesting direction is to generalize Algorithm 1 beyond bi-directed graphs. What about Eulerian graphs? Is approximating Reachability **NP**-hard in general?

References

1 Michael O. Ball. Complexity of network reliability computations. *Networks*, 10(2):153–165, 1980.

2 Michael O. Ball. Computational complexity of network reliability analysis: An overview. *IEEE Trans. Rel.*, 35(3):230–239, 1986.

3 Michael O. Ball and J. Scott Provan. Calculating bounds on reachability and connectedness in stochastic networks. *Networks*, 13(2):253–278, 1983.

4 Henry Cohn, Robin Pemantle, and James G. Propp. Generating a random sink-free orientation in quadratic time. *Electr. J. Comb.*, 9(1), 2002.
5 Charles J. Colbourn. *The Combinatorics of Network Reliability*. Oxford University Press, 1987.
6 Leslie Ann Goldberg and Mark Jerrum. Inapproximability of the Tutte polynomial. *Inf. Comput.*, 206(7):908–929, 2008.
7 Leslie Ann Goldberg and Mark Jerrum. The complexity of computing the sign of the Tutte polynomial. *SIAM J. Comput.*, 43(6):1921–1952, 2014.
8 Igor Gorodezky and Igor Pak. Generalized loop-erased random walks and approximate reachability. *Random Struct. Algorithms*, 44(2):201–223, 2014.
9 Heng Guo, Mark Jerrum, and Jingcheng Liu. Uniform sampling through the Lovasz local lemma. In *STOC*, pages 342–355, 2017.
10 Jane N. Hagstrom. Computing rooted communication reliability in an almost acyclic digraph. *Networks*, 21(5):581–593, 1991.
11 David G. Harris and Aravind Srinivasan. Improved bounds and algorithms for graph cuts and network reliability. In *SODA*, pages 259–278. SIAM, 2014.
12 François Jaeger, Dirk L. Vertigan, and Dominic J. A. Welsh. On the computational complexity of the Jones and Tutte polynomials. *Math. Proc. Cambridge Philos. Soc.*, 108(1):35–53, 1990.
13 Mark Jerrum. On the complexity of evaluating multivariate polynomials. *Ph.D. dissertation*. Technical Report CST-11-81, Dept. Comput. Sci., Univ. Edinburgh, 1981.
14 Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM J. Comput.*, 22(5):1087–1116, 1993.
15 David R. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM J. Comput.*, 29(2):492–514, 1999.
16 David R. Karger. A fast and simple unbiased estimator for network (un)reliability. In *FOCS*, pages 635–644, 2016.
17 David R. Karger. Faster (and still pretty simple) unbiased estimators for network (un)reliability. In *FOCS*, pages 755–766, 2017.
18 Richard M. Karp and Michael Luby. Monte-Carlo algorithms for the planar multiterminal network reliability problem. *J. Complexity*, 1(1):45–64, 1985.
19 Kashyap Babu Rao Kolipaka and Mario Szegedy. Moser and Tardos meet Lovász. In *STOC*, pages 235–244, 2011.
20 Robin A. Moser and Gábor Tardos. A constructive proof of the general Lovász Local Lemma. *J. ACM*, 57(2), 2010.
21 James G. Oxley. *Matroid theory*. Oxford University Press, 1992.
22 J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.
23 James B. Shearer. On a problem of Spencer. *Combinatorica*, 5(3):241–245, 1985.
24 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.
25 Dirk Vertigan and Dominic J. A. Welsh. The compunational complexity of the Tutte plane: the bipartite case. *Comb. Probab. Comput.*, 1:181–187, 1992.
26 Dominic J. A. Welsh. *Complexity: knots, colourings and counting*, volume 186 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1993.
27 Dominic J. A. Welsh. The Tutte polynomial. *Random Struct. Algorithms*, 15(3-4):210–228, 1999.
28 David B. Wilson. Generating random spanning trees more quickly than the cover time. In *STOC*, pages 296–303, 1996.

Perfect Simulation of the Hard Disks Model by Partial Rejection Sampling

Heng Guo
School of Informatics, University of Edinburgh, Informatics Forum, Edinburgh, EH8 9AB, United Kingdom.
hguo@inf.ed.ac.uk
https://orcid.org/0000-0001-8199-5596

Mark Jerrum[1]
School of Mathematical Sciences, Queen Mary, University of London, Mile End Road, London, E1 4NS, United Kingdom.
m.jerrum@qmul.ac.uk
https://orcid.org/0000-0003-0863-7279

Abstract

We present a perfect simulation of the hard disks model via the partial rejection sampling method. Provided the density of disks is not too high, the method produces exact samples in $O(\log n)$ rounds, where n is the expected number of disks. The method extends easily to the hard spheres model in $d > 2$ dimensions. In order to apply the partial rejection method to this continuous setting, we provide an alternative perspective of its correctness and run-time analysis that is valid for general state spaces.

2012 ACM Subject Classification Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases Hard disks model, Sampling, Markov chains

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.69

Related Version Also available at `https://arxiv.org/abs/1801.07342`.

Acknowledgements We thank Mark Huber and Will Perkins for inspiring conversations and bringing the hard disks model to our attention.

1 Introduction

The *hard disks model* is one of the simplest gas models in statistical physics. Its configurations are non-overlapping disks of uniform radius r in a bounded region of $\mathbb{R}^2$. For convenience, in this paper, we take this region to be the unit square $[0,1]^2$. This model was precisely the one studied by Metropolis et al. [12], in their pioneering work on the Markov chain Monte Carlo (MCMC) method. They used Los Alamos' MANIAC computer to simulate a system with 224 disks.

There are two variants of this model. To obtain the *canonical ensemble*, we fix the number (or equivalently, density) of disks and decree that all configurations are "equally likely", subject only to the disks not overlapping. In the *grand canonical ensemble*, we fix the "average" number of disks. To be more specific, centers of the disks are distributed

[1] The work described here was supported by the EPSRC research grant EP/N004221/1 "Algorithms that Count".

according to a Poisson point process of intensity $\lambda_r = \lambda/(\pi r^2)$, conditioned on the disks being non-overlapping. The hard disks model, and its higher dimensional generalization (called the *hard spheres model*) are also related to the optimal sphere packing density [4, 18, 2]. See [6, 1] and references therein for more details. See also [11] for the physics perspective.

Our main aim in this work is to describe and analyse a very simple algorithm for exactly sampling from the grand canonical ensemble, based on the partial rejection sampling paradigm introduced by Guo, Jerrum and Liu [3].

More precisely, the challenge is the following: produce a realisation $P \subset [0,1]^2$ of a Poisson point process of intensity λ_r in the unit square, conditioned on the event that no pair of points in P are closer than $2r$ in Euclidean distance. We refer to this target measure as the *hard disks distribution*. It describes an arrangement of open disks of radius r with centres in $[0,1]^2$ that are not allowed to overlap, but which otherwise do not interact. It is a special case of the Strauss process [17]. Note that, although the disks do not overlap each other, they may extend beyond the boundary of the unit square. Also, the intensity of the underlying Poisson process is normalised so that the expected number of points of P lying in a disk of radius r is λ. This normalisation gives us sensible asymptotics as the radius of the disks tends to zero (equivalently, the number of disks tends to infinity).

Classical rejection sampling applied to this problem yields the following algorithm: repeatedly sample a realisation P of the Poisson process of intensity λ in the unit square until P satisfies the condition that no two points are closer than $2r$, and return P. Unfortunately, for every $\lambda > 0$, however small, the expected number of unsuccessful trials using this approach increases exponentially in r^{-1}, as $r \to 0$. Partial rejection sampling [3] requires only a subset of P to be resampled at each iteration. Algorithm 1 below arises from a routine application of the paradigm to the problem at hand.

The original partial rejection method [3] and its analysis are tailored for the discrete case. In this paper we provide an alternative view on the correctness of the method, which is also valid in the continuous setting. In other words, as with classical rejection sampling, Algorithm 1 terminates with probability 1, producing a realisation of the exact hard disks distribution. In contrast to classical rejection sampling, the expected number of iterations (resampling steps) is now asymptotically $O(\log(r^{-1}))$ as $r \to 0$, provided λ is not too large. We prove that rapid termination occurs when $\lambda < 0.21027$. This analysis is not tight, and experiments suggests that the actual threshold for rapid termination is around $\lambda \approx 0.51$.[2] The experimental advantage of partial rejections is its simple termination rule, unlike MCMC, where it is difficult to determine how long the algorithm should run.

The method extends naturally to the *hard spheres model* in $d > 2$ dimensions. Here, the desired distribution is a Poisson point process in $[0,1]^d$ conditioned on no pair of points being closer than $2r$. The natural normalisation for the intensity of the Poisson process in d dimensions is $\lambda_{r,d} = \lambda/(v_d r^d)$, where v_d is the volume of a ball of radius 1 in $\mathbb{R}^d$. With this convention, we prove that rapid termination occurs in d dimensions provided $\lambda < 2^{-(d+\frac{1}{2})}$.

The *expected packing density* $\alpha(\lambda)$ or simply α for this model is the expected total volume of balls. (Note that, neglecting boundary effects, α is the proportion of the unit cube occupied by balls.) The quantity $\alpha(\lambda)$ grows monotonically with λ, but intuitively we expect its rate of growth to slow down dramatically as the balls pack more tightly. The connection between expected packing density α and intensity λ has recently been thoroughly explored by Jenssen, Joos and Perkins [6]. Using their results, we show that partial rejection sampling

[2] The physics prediction of phase transitions, for the *canonical ensemble* in two dimensions, is ≈ 0.7. This threshold is more related to the expected packing density $\alpha(\lambda)$ discussed below.

Algorithm 1 Partial Rejection Sampling for the hard disks model.

PRS for Hard Disks(λ, r) // r is the disk radius and $\lambda_r = \lambda/(\pi r^2)$ the intensity
Let P be a sample from the Poisson point process of intensity λ_r on the unit square
while $B \leftarrow \mathsf{BadPoints}(P) \neq \emptyset$ **do**
  $S \leftarrow B + D_{2r}(\mathbf{0})$ // Resampling set is the Minkowski sum of B with a disk of radius $2r$
  Let P_S be a sample from the Poisson point process of intensity λ_r on S
  $P \leftarrow (P \setminus B) \cup P_S$
end while
return P

can achieve expected packing density $\Omega(2^{-d})$ while retaining runtime $O(\log(r^{-1}))$. Although sphere packings of density $\Omega(d2^{-d})$ have been proved to exist, there is no polynomial-time sampler that provably achieves packing density beyond $O(2^{-d})$, as far as we are aware.

Other approaches to exact sampling include Coupling From The Past (CFTP), which was adapted to point processes by Kendall [8] and Kendall and Møller [9]. Recently, Moka, Juneja and Mandjes [13] proposed an algorithm based on rejection and importance sampling. Although this algorithm, like ours, is based on rejection sampling, it does not share our asymptotic performance guarantee. Indeed, its running time appears to grow exponentially as the number of disks goes to infinity, with the density of disks held constant. See also [6, 14] for rigorous bounds on packing densities of grand canonical ensembles.

Approximate sampling via Markov chain simulation has been studied by Kannan, Mahoney and Montenegro [7] and Hayes and Moore [5] in the context of the canonical ensemble, where the number of disks in a configuration is fixed. The best rigorous density bound to guarantee rapid mixing is below ≈ 0.154 [5]. It is should be noted that this is not directly comparable with our $\lambda < 0.21027$ due to the difference in models. To obtain canonical ensembles, we could use Algorithm 1 and further condition on the number of desired disks. However, the only rigorous guarantee of this approach, via [6], is $\alpha(0.21027) > 0.0887$.

2 The sampling algorithm

The following notation will be used throughout. If P is a finite subset of $[0,1]^2$ then

$$\mathsf{BadPairs}(P) = \big\{\{x,y\} : x,y \in P \wedge x \neq y \wedge \|x-y\| < 2r\big\},$$

where $\|\cdot\|$ denotes Euclidean norm, and

$$\mathsf{BadPoints}(P) = \bigcup \mathsf{BadPairs}(P).$$

The open disk of radius r with centre $x \in [0,1]^2$ is denoted by $D_r(x)$. The finite set $\Pi \subset [0,1]^2$ always denotes a realisation of the Poisson point process of intensity λ_r on $[0,1]^2$. For a random variable X and event $\mathcal{E}$ we use $\mathcal{D}(X)$ to denote the distribution (law) of X, and $\mathcal{D}(X \mid \mathcal{E})$ the distribution of X conditioned on $\mathcal{E}$ occurring. Thus, $\mathcal{D}(\Pi \mid \mathsf{BadPoints}(\Pi) = \emptyset)$ is the hard disks distribution that we are interested in.

Our goal is to analyse the correctness and running time of a sampling algorithm for the hard disks model (see Algorithm 1 below), specifically to determine the largest value of λ for which it terminates quickly, i.e., in $O(\log r^{-1})$ iterations. The algorithm is an example application of "Partial Rejection Sampling" [3], adapted to the continuous state space setting.

3 Correctness

Let B be any finite subset of $[0,1]^2$. We say that B is a *feasible set* of bad points if $\mathsf{BadPoints}(B) = B$; this is equivalent to saying that there is a finite subset $R \subset [0,1]^2$ with $B = \mathsf{BadPoints}(R)$. The key to establishing correctness of Algorithm 1 is the following loop invariant:

$$\mathcal{D}(P \mid \mathsf{BadPoints}(P) = B) = \mathcal{D}(\Pi \mid \mathsf{BadPoints}(\Pi) = B),$$

for every feasible set B, where P is any intermediate set of points during the execution of the algorithm, and Π is a realisation of the Poisson point process. Let us consider what the right hand side means operationally. Let $S = B + D_{2r}(\mathbf{0})$. (This is the resampling set used by the algorithm.) Let Q be a sample from the distribution $\mathcal{D}(\Pi \mid \mathsf{BadPoints}(\Pi) = B)$. The only points in Q that lie inside S are the points in B. (Any extra points would create more bad pairs than there actually are.) Thus $Q \cap S = B$. Outside of S there are no bad pairs; thus $Q \cap \overline{S}$ is a sample from the hard disks distribution on $\overline{S} = [0,1]^2 \setminus S$. Note that, setting $B = \emptyset$, we see that $\mathcal{D}(\Pi \mid \mathsf{BadPoints}(\Pi) = \emptyset)$ is just the hard disks distribution on $[0,1]^2$.

▶ **Theorem 1.** *Algorithm 1 is correct: conditional on halting, Algorithm 1 produces a sample from the hard disks distribution with intensity $\lambda_r = \lambda/(\pi r^2)$.*

Theorem 1 follows from Lemma 2 below.

Let T (a random variable) be the number of iterations of the while-loop. On each iteration, the while loop terminates with probability bounded away from 0; thus T is finite with probability 1. (Indeed, T has finite expectation.) Let P_t, for $1 \le t \le T$, be the point set P after t iterations of the loop, and P_0 be the initial value of P (which is just a realisation of the Poisson point process on $[0,1]^2$). We say that $B_0, B_1, \ldots, B_t \subset [0,1]^2$ is a *feasible sequence* of (finite) point sets if there exists a run of Algorithm 1 with $\mathsf{BadPoints}(P_0) = B_0, \ldots, \mathsf{BadPoints}(P_t) = B_t$.

▶ **Lemma 2.** *Let $B_0, B_1, \ldots, B_t \subset [0,1]^2$ be a feasible sequence. Then*

$$\begin{aligned}\mathcal{D}\big(P_t \,\big|\, \mathsf{BadPoints}(P_0) = B_0 \wedge \cdots \wedge \mathsf{BadPoints}(P_t) = B_t\big) &= \mathcal{D}(P_t \,\big|\, \mathsf{BadPoints}(P_t) = B_t) \\ &= \mathcal{D}(\Pi \,\big|\, \mathsf{BadPoints}(\Pi) = B_t).\end{aligned}$$

Proof. We prove the result by induction on t. The base case, $t = 0$, holds by construction: P_0 is just a realisation of the Poisson point process on $[0,1]^2$. Our induction hypothesis is

$$\mathcal{D}\big(P_t \,\big|\, \mathsf{BadPoints}(P_0) = B_0 \wedge \cdots \wedge \mathsf{BadPoints}(P_t) = B_t\big) = \mathcal{D}(\Pi \,\big|\, \mathsf{BadPoints}(\Pi) = B_t), \quad (1)$$

for every feasible sequence $B_0, \ldots, B_t$. Extend the feasible sequence to B_{t+1}. For the inductive step, we assume (1) and aim to derive

$$\begin{aligned}\mathcal{D}\big(P_{t+1} \,\big|\, \mathsf{BadPoints}(P_0) = B_0 \wedge \cdots \wedge \mathsf{BadPoints}(P_{t+1}) = B_{t+1}\big) \\ = \mathcal{D}(\Pi \mid \mathsf{BadPoints}(\Pi) = B_{t+1}). \quad (2)\end{aligned}$$

The resampling set on iteration $t+1$ is $S = B_t + D_{2r}(\mathbf{0})$. As a first step we argue below that

$$\mathcal{D}\big(P_{t+1} \,\big|\, \mathsf{BadPoints}(P_0) = B_0 \wedge \cdots \wedge \mathsf{BadPoints}(P_t) = B_t\big) = \mathcal{D}(\Pi \mid \mathsf{BadPairs}(\Pi) \cap \overline{S}^{(2)} = \emptyset), \quad (3)$$

where $\overline{S}^{(2)}$ denotes the set of unordered pairs of elements from $\overline{S}$. We have noted that (1) implies that, outside of the resampling set S, the point set P_t is a realisation of the

hard disks distribution. Also, the algorithm does not resample points outside of S. Thus $P_{t+1} \cap \overline{S} = P_t \cap \overline{S}$ is Poisson distributed, conditioned on there being no bad pairs. Inside S, resampling has left behind a fresh Poisson point process $P_{t+1} \cap S$. These considerations give (3).

Next, we condition on B_{t+1}:

$$\mathcal{D}\big(P_{t+1} \,\big|\, \mathsf{BadPoints}(P_0) = B_0 \wedge \cdots \wedge \mathsf{BadPoints}(P_t) = B_t \wedge \mathsf{BadPoints}(P_{t+1}) = B_{t+1}\big) \\ = \mathcal{D}\big(\Pi \,\big|\, \mathsf{BadPairs}(\Pi) \cap \overline{S}^{(2)} = \emptyset \wedge \mathsf{BadPoints}(\Pi) = B_{t+1}\big).$$

Since B_{t+1} contains no bad pairs with both endpoints in $\overline{S}$, the event $\mathsf{BadPoints}(\Pi) = B_{t+1}$ entails the event $\mathsf{BadPairs}(\Pi) \cap \overline{S}^{(2)} = \emptyset$. Thus, we have

$$\mathcal{D}\big(P_{t+1} \,\big|\, \mathsf{BadPoints}(P_0) = B_0 \wedge \cdots \wedge \mathsf{BadPoints}(P_t) = B_t \wedge \mathsf{BadPoints}(P_{t+1}) = B_{t+1}\big) \\ = \mathcal{D}(\Pi \mid \mathsf{BadPoints}(P_{t+1}) = B_{t+1}).$$

The right hand side of this equation does not involve $B_0, \ldots, B_t$, and so

$$\mathcal{D}(P_{t+1} \mid \mathsf{BadPoints}(P_{t+1}) = B_{t+1}) = \mathcal{D}(\Pi \mid \mathsf{BadPoints}(\Pi) = B_{t+1}).$$

This completes the induction step (2) and the proof. ◀

Proof of Theorem 1. As we observed earlier, T, the number of iterations of the while-loop, is finite with probability 1. By Lemma 2, noting that $B_T = \emptyset$,

$$\mathcal{D}(P_T) = \mathcal{D}(\Pi \mid \mathsf{BadPoints}(\Pi) = \emptyset).$$

In other words, at termination, Algorithm 1 produces a realisation of the hard disks distribution on $[0,1]^2$. ◀

4 Run-time analysis

We consider how the number of "bad events", i.e., the cardinality of the set $\mathsf{BadPairs}(P_t)$, evolves with time. As usual Π denotes a realisation of the Poisson point process of intensity λ_r. Also denote by Δ a realisation of the hard disks process of the same intensity. We need the following stochastic domination result.

▶ **Lemma 3.** *The hard disks distribution is stochastically dominated by the Poisson point process with the same intensity. That is, we can construct a joint sample space for* Π *and* Δ *such that* $\Delta \subseteq \Pi$.

Holley's criterion is a useful test for stochastic domination, but it is not of direct use to us in the proof of Lemma 3, because it applies only to finite state spaces. Fortunately, Preston [15, Theorem 9.1], has derived a version of Holley's criterion that fits our situation. We will mostly follow Preston's notation, except that, to save confusion, we will use P and Q, rather than x and y, to denote finite sets of points. In order to state his result, we need some notation. In our application, $\widetilde{\omega}_n$ is the distribution on $([0,1]^2)^{(n)}$ obtained by sampling n points independently and uniformly at random from $[0,1]^2$, and regarding the points as indistinguishable; furthermore, $\widetilde{\omega} = \sum_{n=0}^{\infty} \widetilde{\omega}_n / n!$. (For consistency with Preston, we have left $\widetilde{\omega}$ unnormalised. If we had made it into a probability distribution by division by e, then $\widetilde{\omega}$ could be thought of as follows: sample an integer k from the Poisson distribution with mean 1, and then pick k (unlabelled) points uniformly and independently.) Denote by Ω

the set of all finite subsets of $[0,1]^2$, and by $\mathcal{F}$ the set of non-negative measurable functions $\Omega \to \mathbb{R}$ satisfying

$$\int f \, d\widetilde{\omega} = 1, \tag{4}$$

and

$$f(P) = 0 \text{ and } P \subseteq Q \text{ implies } f(Q) = 0. \tag{5}$$

(See Preston [15, Section 9] for detailed formal definitions of the concepts here.)

▶ **Lemma 4** (Theorem 9.1 of [15]). *Let $f_1, f_2 \in \mathcal{F}$ and suppose that*

$$\frac{f_1(P+\xi)}{f_1(P)} \geq \frac{f_2(Q+\xi)}{f_2(Q)}, \quad \text{for all } Q \subseteq P \in \Omega \text{ and } \xi \in [0,1]^2 \setminus P \tag{6}$$

(where, by convention, $0/0 = 0$). Then, for all bounded, measurable, non-decreasing functions $g : \Omega \to \mathbb{R}$,

$$\int g f_1 \, d\widetilde{\omega} \geq \int g f_2 \, d\widetilde{\omega},$$

i.e., if μ_i is the probability measure having density f_i with respect to $\widetilde{\omega}$, then μ_1 stochastically dominates μ_2.

Proof of Lemma 3. We set

$$f_1(P) = C_1 \lambda_r^{|P|}$$

and

$$f_2(P) = \begin{cases} C_2 \lambda_r^{|P|}, & \text{if } \mathsf{BadPairs}(P) = \emptyset; \\ 0, & \text{otherwise.} \end{cases}$$

The normalising constants C_1 and C_2 are chosen so that both $f = f_1$ and $f = f_2$ satisfy (4). (There is an explicit expression for C_1, namely $C_1 = \exp(-\lambda_r)$, but not for C_2.) Notice that both f_1 and f_2 also satisfy (5). Notice also that the probability measures μ_1 and μ_2 of the Poisson point process and the hard disks process have densities f_1 and f_2 with respect to $\widetilde{\omega}$. The premise (6) of Lemma 4 holds, since the left-hand side is always λ_r and the right-hand side is either λ_r or 0. The conclusion is that μ_1 dominates μ_2. Strassen's Theorem [10, 16], allows us to conclude the existence of a coupling of Π and Δ as advertised in the statement of the lemma (except, possibly, on a set of measure zero). ◀

▶ **Lemma 5.** *There is a bound $\bar{\lambda} > 0$ such that the expected number of iterations of the while-loop in Algorithm 1 is $O(\log r^{-1})$ when $\lambda < \bar{\lambda}$.*

Proof. First observe that $\mathsf{BadPairs}(P)$ determines $\mathsf{BadPoints}(P)$ and vice versa. So conditioning on the set $\mathsf{BadPairs}(P)$ is equivalent to conditioning on $\mathsf{BadPoints}(P)$.

Introduce random variables $Z_t = |\mathsf{BadPairs}(P_t)|$, for $1 \leq t \leq T$. Our strategy is to show that

$$\mathbb{E}(Z_{t+1} \mid Z_0, \ldots, Z_t) \leq \alpha^{-1} Z_t, \tag{7}$$

for some $\alpha > 1$. Then $Z_0, \alpha Z_1, \alpha^2 Z_2, \alpha^3 Z_3, \ldots$ is a supermartingale (with the convention that $Z_t = 0$ for all $t > T$). Therefore, $\mathbb{E}\, Z_t \leq \alpha^{-t}\, \mathbb{E}\, Z_0 \leq \frac{1}{2}\lambda_r^2 \alpha^{-t}$. Here, we have used the fact that $|P_0|$ is a Poisson random variable with expectation λ_r, and

$$Z_0 = |\mathsf{BadPairs}(P_0)| \leq \tfrac{1}{2}|P_0|\,(|P_0| - 1),$$

and hence

$$\mathbb{E}\, Z_0 \leq \mathbb{E}\left[|P_0|\,(|P_0|-1)\right] = \tfrac{1}{2}\lambda_r^2.$$

Setting $t = O(\log r^{-1} + \log \varepsilon^{-1})$, we obtain $\mathbb{E}\, Z_t < 1/\varepsilon$, and hence $\Pr(T > t) \leq \varepsilon$. It follows that the expected number of iterations of the while-loop of Algorithm 1 is $O(\log r^{-1})$. Note that the probability of non-termination decreases exponentially with t, so the probability of large deviations above the expected value of T is low.

Crude estimates give $\bar{\lambda} = 1/(4\sqrt{2})$. The calculation goes as follows. Suppose, in (7), we condition on the random variables $\mathsf{BadPoints}(P_0), \ldots, \mathsf{BadPoints}(P_t)$, rather than simply on $Z_0, \ldots, Z_t$. This is more stringent conditioning, since the former random variables determine the latter. It is enough to establish (7) under the more stringent conditioning. So fix $\mathsf{BadPoints}(P_0) = B_0, \ldots, \mathsf{BadPoints}(P_t) = B_t$, and note that this choice also fixes the resampling sets $S_0, \ldots, S_t$. Suppose $Z_t = |\mathsf{BadPairs}(P_t)| = k$. Inside the resampling set S_t we have a Poisson point process $P_{t+1} \cap S_t$ of intensity λ_r. Outside, by Lemma 2, there is a realisation of the hard disks process. Since we are seeking an upper bound on Z_{t+1} we may, by Lemma 3, replace $P_{t+1} \cap \overline{S}_t$ by a Poisson point process of intensity λ_r.

Let $k' = \mathbb{E}(Z_{t+1} \mid Z_0, \ldots, Z_t)$. From the above considerations we have

$$k' \leq \int_{S_t} \lambda_r \int_{[0,1]^2} \lambda_r\, \mathbf{1}_{\|x-y\|\leq 2r}\, dy\, dx. \tag{8}$$

This is an overestimate, as we are double-counting overlapping disks whose centres both lie within S_t. Now, S_t is a union of at most $2k$ disks of radius $2r$. Thus

$$\begin{aligned} k' &\leq 2k\lambda_r^2 \int_{D_{2r}(\mathbf{0})} \int_{\mathbb{R}^2} \mathbf{1}_{\|x-y\|\leq 2r}\, dy\, dx \\ &= 2k\lambda_r^2 \int_{D_{2r}(\mathbf{0})} \int_{D_{2r}(x)} dy\, dx \\ &= 2k\lambda_r^2 \times 4\pi r^2 \times 4\pi r^2 \\ &= 32\lambda^2 k. \end{aligned} \tag{9}$$

There are further sources of slack here: there may be fewer than $2k$ disks, the disks comprising S_t certainly overlap, and, for points x near the boundary, some of disks $D_{2r}(x)$ will lie partly outside the unit square. (The last of these presumably has no effect asymptotically, as $r \to 0$.) Setting $\bar{\lambda} = 1/(4\sqrt{2}) = 0.17677+$, we see that $\alpha = k/k' > 1$ for any $\lambda < \bar{\lambda}$, and $(Z_t)_{t=0}^{\infty}$ is a supermartingale, as required. ◀

The constant $\bar{\lambda}$ may seem quite small. Note, however, that classical rejection sampling cannot achieve any $\bar{\lambda} > 0$. The argument goes as follows. Divide $[0,1]^2$ into $r \times r$ squares. If there are two points in the same square then they will certainly be less than distance $2r$ apart. The number of points in each square is Poisson distributed with mean λ/π. Thus for any $\lambda > 0$ the probability that a particular square has at least two points is bounded away from zero. The number of points in each square is independent of all the others. It follows that the running time of classical rejection sampling is exponential in r^{-2}.

The above derivation for $\bar{\lambda}$ is quite crude and can be improved.

▶ **Lemma 6.** *The constant $\bar{\lambda}$ in Lemma 5 can be taken to be* 0.21027.

Proof. For each of the $2k$ disks, the right-hand side of inequality (9) counts pairs of points (x, y) with x in the disk, and y anywhere within distance $2r$ of x. Since a bad event is determined by an *unordered* pair of points, this gives rise to significant double counting.

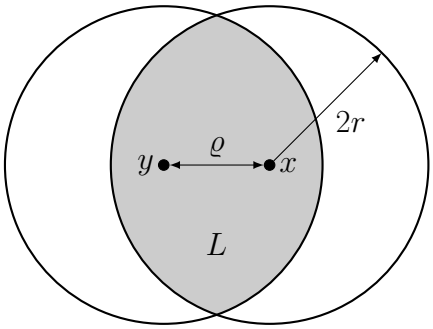


Figure 1 An illustration of double counting.

In particular, pairs (x, y) with x and y lying in the same ball are double counted. We can subtract off these pairs to obtain a better estimate.

For a single ball, the correction is

$$C = \frac{1}{2} \int_{D_{2r}(\mathbf{0})} \lambda_r \int_{D_{2r}(\mathbf{0})} \lambda_r \mathbf{1}_{\|x-y\| \le 2r} \, dy \, dx.$$

(The initial factor of one half arises because we want to count unordered pairs.) With the change of variables $x = 2rx'$ and $y = 2ry'$ this expression simplifies to

$$\begin{aligned} C &= \frac{1}{2} \times 16 r^4 \lambda_r^2 \int_{D_1(\mathbf{0})} \int_{D_1(\mathbf{0})} \mathbf{1}_{\|x'-y'\| \le 1} \, dy' \, dx' \\ &= 8\lambda_r^2 r^4 \int_{D_1(\mathbf{0})} L(\|x'\|) \, dx', \end{aligned}$$

where $L(\|x'\|)$ is the area of the "lens" $D_1(\mathbf{0}) \cap D_1(x')$. Letting ϱ denote the offset of the centres of the two disks, the area of the lens is given by

$$L(\varrho) = 2 \arccos(\varrho/2) - \tfrac{1}{2} \varrho \sqrt{4 - \varrho^2}.$$

(This is by elementary geometry: the lens is the intersection of two sectors, one from each of the disks, and its area can be computed by inclusion-exclusion.) An illustration (before shifting y to $\mathbf{0}$) is given in Figure 1. The shaded area is L.

Translating to polar coordinates (ϱ, θ),

$$\begin{aligned} C &= 8\lambda_r^2 r^4 \int_{D_1(\mathbf{0})} L(\|x'\|_2) \, dx' \\ &= 8\lambda_r^2 r^4 \int_0^{2\pi} \int_0^1 \varrho \, L(\varrho) \, d\varrho \, d\theta \\ &= \frac{8\lambda^2}{\pi^2} \times 2\pi \int_0^1 \varrho \, L(\varrho) \, d\varrho \\ &= \frac{16\lambda^2}{\pi} \left[\frac{\pi}{2} + (\varrho^2 - 1) \arccos\left(\frac{\varrho}{2}\right) - \left(\frac{\varrho}{4} + \frac{\varrho^3}{8}\right) \sqrt{4 - \varrho^2} \right]_0^1 \\ &= \left(8 - \frac{6\sqrt{3}}{\pi}\right) \lambda^2. \end{aligned}$$

(The integral was evaluated using the Maple computer algebra system.) Our revised upper bound on k' is thus

$$k' \le 2k(16\lambda^2 - C) = 2k\lambda^2 \left(8 + \frac{6\sqrt{3}}{\pi}\right), \tag{10}$$

Solving

$$\bar{\lambda}^2\left(16+\frac{12\sqrt{3}}{\pi}\right)=1$$

yields the improved bound of $\bar{\lambda}=0.21027+$. ◀

There are other factors that could in principle be used to increase $\bar{\lambda}$ further — each disk necessarily overlaps with at least one other disk, some bad events are triple or quadruple counted — but the computational difficulties rapidly increase when attempting to account for these.

5 Three or more dimensions

In higher dimensions, the hard disk model is known as the hard spheres model. Everything in Sections 3 and 4 carries across to $d>2$ dimensions with little change. For general d, the appropriate scaling for the intensity is $\lambda_{r,d}=\lambda/(v_d r^d)$, where v_d is the volume of a ball of unit radius in d dimensions. Note that in a realisation of a Poisson point process with intensity $\lambda_{r,d}$, the expected number of points in a ball of radius r is λ.

The analogue of equation (8) is

$$k' \le \int_{S_t} \lambda_{r,d} \int_{[0,1]^d} \lambda_{r,d}\, \mathbf{1}_{\|x-y\|\le 2r}\, dy\, dx,$$

which leads to

$$k' \le 2^{2d+1}\lambda^2 k.$$

So setting $\bar{\lambda}=2^{-(d+\frac{1}{2})}$ we find that $\alpha=k/k'>1$ for any $\lambda<\bar{\lambda}$. It follows that the running time of partial rejection sampling is $O(\log r)$ for any $\lambda<\bar{\lambda}$.

By a result of Jenssen, Joos and Perkins [6], we lose just a constant factor when translating from intensity λ to packing density α. (It is partly to connect with their work, we measure intensity in terms of the expected number of points in a ball of radius r.) In the proof of [6, Thm 2], the following inequality is derived:

$$\alpha \ge \inf_z \max\left\{\lambda e^{-z},\, 2^{-d}\exp[-2\cdot 3^{d/2}\lambda]\cdot z\right\}.$$

Assuming $\lambda\le\bar{\lambda}$, which holds in the range of validity of our algorithm, we have $\sqrt{2}\lambda\le 2^{-d}$ and hence

$$\begin{aligned}\alpha &\ge \inf_z \max\left\{\lambda e^{-z},\, \sqrt{2}\lambda\exp[-\sqrt{2}(3/4)^{d/2}]\cdot z\right\}\\ &= c_d\lambda,\end{aligned}$$

where

$$c_d=\inf_z \max\left\{e^{-z},\, \sqrt{2}\exp[-\sqrt{2}(3/4)^{d/2}]\cdot z\right\}.$$

Note that (c_d) is monotonically increasing, with $c_2=0.42220+$, and $\lim_{d\to\infty} c_d=0.63724+$. It follows that we can reach expected packing density $\Omega(2^{-d})$ with $O(\log r^{-1})$ expected iterations. This is currently the best that can be achieved by any provably correct sampling algorithm with polynomial (in $1/r$) runtime [7]. The asymptotically best packing density currently rigorously known is $d2^{-d}$, but achieving this would require λ to grow exponentially fast in d. This is clearly beyond the capability of partial rejection sampling, but also beyond the capability of any known efficient sampling algorithm.

References

1. Henry Cohn. A conceptual breakthrough in sphere packing. *Notices Amer. Math. Soc.*, 64(2):102–115, 2017.
2. Henry Cohn, Abhinav Kumar, Stephen D. Miller, Danylo Radchenko, and Maryna Viazovska. The sphere packing problem in dimension 24. *Ann. of Math. (2)*, 185(3):1017–1033, 2017.
3. Heng Guo, Mark Jerrum, and Jingcheng Liu. Uniform sampling through the Lovasz local lemma. In *STOC*, pages 342–355, 2017.
4. Thomas C. Hales. A proof of the Kepler conjecture. *Ann. of Math. (2)*, 162(3):1065–1185, 2005.
5. Thomas P. Hayes and Cristopher Moore. Lower bounds on the critical density in the hard disk model via optimized metrics. *CoRR*, abs/1407.1930, 2014.
6. Matthew Jenssen, Felix Joos, and Will Perkins. On the hard sphere model and sphere packings in high dimensions. *ArXiv*, abs/1707.00476, 2017.
7. Ravi Kannan, Michael W. Mahoney, and Ravi Montenegro. Rapid mixing of several Markov chains for a hard-core model. In *ISAAC*, pages 663–675, 2003.
8. Wilfrid S. Kendall. Perfect simulation for the area-interaction point process. In *Probability towards 2000 (New York, 1995)*, volume 128 of *Lect. Notes Stat.*, pages 218–234. Springer, New York, 1998.
9. Wilfrid S. Kendall and Jesper Møller. Perfect simulation using dominating processes on ordered spaces, with application to locally stable point processes. *Adv. Appl. Probab.*, 32(3):844—-865, 2000.
10. Torgny Lindvall. On Strassen's theorem on stochastic domination. *Electron. Comm. Probab.*, 4:51–59, 1999.
11. Hartmut Löwen. Fun with hard spheres. In Klaus R. Mecke and Dietrich Stoyan, editors, *Statistical Physics and Spatial Statistics*, pages 295–331, 2000.
12. Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21(6):1087–1092, 1953.
13. S. B. Moka, S. Juneja, and M. R. H. Mandjes. Perfect sampling for Gibbs processes with a focus on hard-sphere models. *ArXiv*, abs/1705.00142, 2017.
14. Will Perkins. Birthday inequalities, repulsion, and hard spheres. *Proc. Amer. Math. Soc.*, 144(6):2635–2649, 2016.
15. Chris Preston. Spatial birth-and-death processes (with discussion). *Bull. Inst. Internat. Statist.*, 46(2):371–391, 405–408 (1975), 1975.
16. V. Strassen. The existence of probability measures with given marginals. *Ann. Math. Statist.*, 36:423–439, 1965.
17. David J. Strauss. A model for clustering. *Biometrika*, 62(2):467–475, 1975.
18. Maryna S. Viazovska. The sphere packing problem in dimension 8. *Ann. of Math. (2)*, 185(3):991–1015, 2017.

Non-Preemptive Flow-Time Minimization via Rejections

Anupam Gupta
Carnegie Mellon University

Amit Kumar
IIT Delhi

Jason Li
Carnegie Mellon University

Abstract

We consider the online problem of minimizing weighted flow-time on unrelated machines. Although much is known about this problem in the resource-augmentation setting, these results assume that jobs can be preempted. We give the first constant-competitive algorithm for the non-preemptive setting in the rejection model. In this rejection model, we are allowed to reject an ε-fraction of the total weight of jobs, and compare the resulting flow-time to that of the offline optimum which is required to schedule all jobs. This is arguably the weakest assumption in which such a result is known for weighted flow-time on unrelated machines. While our algorithms are simple, we need a delicate argument to bound the flow-time. Indeed, we use the dual-fitting framework, with considerable more machinery to certify that the cost of our algorithm is within a constant of the optimum while only a small fraction of the jobs are rejected.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Scheduling, Rejection, Unrelated Machines, Non-Preemptive

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.70

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.09602`.

1 Introduction

Consider the problem of scheduling jobs for weighted flow-time minimization. Given a set of m unrelated machines, jobs arrive online and have to be processed on one of these machines. Each job j is released at some time r_j, has a potentially different processing requirement (size) p_{ij} on each machine i, and a weight w_j which is a measure of its importance. The objective function is the *weighted flow time* (or *response time*): if the job j completes its processing at time C_j, the flow/response time is $(C_j - r_j)$, i.e., the time the job spends in the system. The goal is now to minimize the weighted sum $\sum_j w_j(C_j - r_j)$.

The problem of flow-time minimization has been extensively studied both from theoretical and practical perspectives. The theoretical analyses have to assume that the jobs can be pre-empted in order to prove any meaningful competitive ratio, and it is easy to see why. If we schedule a long low-weight job and a large number of short high-weight items arrive meanwhile, we cannot afford to delay the latter (else we suffer large flow-time), so the only solution would be to preempt the former (See [14] for strong lower bounds.) And even with pre-emption, the problem turns out to be difficult for multiple machines: e.g., [11] show no bounded competitive ratio is possible for the case of unrelated machines. Hence, it is natural to consider models with "resource augmentation" where the algorithm has slightly more

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 70; pp. 70:1–70:13


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

resources than the adversary. E.g., in the speed-augmentation setting, where the algorithm uses machines of speed $(1+\varepsilon)$-times those of the adversary, Chadha et al. [7] showed how to get a preemptive schedule with weighted flow time at most poly$(1/\varepsilon)$ times the optimal flow time.

A different model of resource augmentation was proposed by Choudhury et al. [8] in the context of load balancing and maximum weighted flow-time, where we are allowed to reject at most ε-fraction of the total weight of the incoming jobs, but we compare with the optimum off-line algorithm which is required to process all the jobs. The motivation was two-fold: (a) the model is arguably more natural, since it does not involve comparing to an imaginary optimal schedule running on a slower machine, and (b) even with speed-augmentation, there are problems, e.g. on-line load balancing, where even a constant factor speed-up does not suffice to give meaningful results. Indeed, getting a non-preemptive schedule for weighted flow-time is one of these problems. Consider for example the following input: a job of unit size and unit weight at time 0 arrives. As soon as the algorithm schedules it, the adversary releases L jobs of size $\varepsilon \ll 1/L^2$. The optimal off-line flow-time is $O(1)$, but the algorithm will incur total flow-time of $\Omega(L)$. The model of job rejection is intuitively more powerful than speed-augmentation (although no such formal connection is known): loosely, the speed-augmentation model only allows us to uniformly reject an ε-fraction of each job, whereas the rejection model allows us to "non-uniformly" reject an arbitrary subset of jobs, as long as they contribute only an ε-fraction of the total weight.

1.1 Our Results

We consider the problem of non-preemptive scheduling on unrelated machines where the objective is to minimize total weighted flow-time of jobs. Our main result is the following:

▶ **Theorem 1** (Main Theorem). *For the problem of online weighted flow-time minimization on unrelated machines, there is a deterministic algorithm that rejects at most an ε-fraction of the total weight of incoming jobs, and ensures that the total weighted flow time for the remaining jobs is at most an $O(1/\varepsilon^3)$ factor times the optimal weighted flow time without rejections.*

Note that we compare with the off-line optimum which is allowed to be preemptive (in fact, migratory), but is required to process all the jobs. Our guarantees are, in fact, stronger. Define the notion of a "departure time" D_j for the job, which is the time at which either the job completes non-preemptively (in which case $D_j = C_j$) or is the time at which the job is rejected. A different natural definition of the total weighted response time in the presence of rejections would be the following:

$$\text{total weighted response time} := \sum_j w_j(D_j - r_j).$$

Keeping this quantity small forces us to decide on jobs early, and discourages us from letting jobs linger in the system for a long time, only to reject them at some late date. (Such a behaviour would be very undesirable for a scheduling policy, and would even be considered "unprofessional" in real-world settings.)

In fact the bulk of our work is in handling the single machine case. For this case, we get a slightly stronger bound.

▶ **Theorem 2** (Single Machine). *For the problem of online weighted flow-time minimization on a single machine, there is a deterministic algorithm that rejects at most an ε-fraction of the total weight of incoming jobs, and ensures that the total weighted flow time for the remaining*

jobs is at most $O(1/\varepsilon^2)$ factor times the optimal weighted flow time without rejections even when the offline optimum is given $(1+\varepsilon)$-extra speedup.

The fact that we can compare with an optimum offline algorithm which has faster machine allows us to use known immediate-dispatch algorithms for the setting of unrelated machines in a black-box manner [7, 2].

1.2 Our Techniques

Let us first focus on the single-machine case. Our algorithm rejects jobs in two different ways: some of the jobs are rejected immediately upon arrival, and others are rejected after receiving some processing. Moreover, assume for the moment that we are running a preemptive schedule, but without speed-augmentation. The high-level idea is to reject a "random" ε-fraction of jobs that come in. At an intuitive level, this rejects only ε-fraction of the weight (although this only in expectation, whereas we want this to hold deterministically at all times), and should create the effect of ε-speed augmentation. To implement this, let α_j be the "effect" of job j on the system – i.e., the increase in the total flow-time of the jobs currently in the system (assuming no future jobs arrive). The value of α_j also naturally corresponds to settings of dual variables for a natural flow-time LP. Using this we can (more-or-less) show that (a) the α_j values of the rejected jobs give us a lower bound on OPT, whereas (b) the α_j values of the non-rejected jobs upper-bound our cost. Hence, our goal becomes: at each time cancel *at most* an ε-fraction of the total weight $\sum_j w_j$, while cancelling *at least* an ε-fraction (say) of the total "dual" value $\sum_j \alpha_j$.

A little thought shows that this abstract task is hopeless in general for any deterministic strategy (say, if the α values rise very sharply), so we have to take the structure of the α_j values into account. We do this in two steps: we break the α_j contribution into α_j^+, the effect of job j on items denser than j, and α_j^-, its effect on less-dense items. Now we put jobs into buckets based on having the same (α^+, w) or (α^-, w) values, and rejecting each $1/\varepsilon^{th}$ job in each bucket. (The actual bucketing is a little finer, see §3.) Moreover, we reject the first job in each (α^+, w) bucket. The complications arise because we are more aggresive for each such (α^+, w) bucket, and because we may not have rejected any jobs in the (α^-, w) if it had less than $1/\varepsilon$ items. In §4.3.1 we perform a delicate charging to relate our aggressive rejections for the former to the total running time of the jobs, and show that (i) this aggressive rejection does not reject too much weight, and (b) also compensates for our timid rejections in the latter bucketing.

This high-level argument was done assuming preemptions. Since we want a non-preemptive schedule, only immediate rejections do not suffice, and we also must reject some jobs which we have started processing – indeed, if a large number of high-density ("important") jobs arrive right after we start processing some long low-density job j, delaying these more important jobs would cause large flow-time. So we must reject job j. However, as long as the total weight of these new jobs is w_j/ε, we can charge the rejection to these new jobs. This rejection makes the schedule very "unstable" and hence complicates the analysis. To get around this problem, we *mark* the job j as "preemptible". We then run a version of HDF with some preemptible and other non-preemptible jobs, and show that its performance can also be related to the LP variables.

Finally, for the multiple machines case we can perform a modular reduction to the single-machines case. We first use the *immediate dispatch* algorithm of Anand et al. [2] to assign jobs to machines, assuming speed augmenation. We then show our algorithm does well even compared to a stronger benchmark (i.e., where the offline schedule – instead of the online schedule – gets the speed augmentation). This gives us the theorem for the unrelated machines.

1.3 Related Work

There has been considerable work on the problem of minimizing total flow-time in the online setting, though most of it is in the preemptive setting. Several logarithmic competitive algorithms are known for unweighted flow-time on identical machines setting [15, 3], and in the related machines setting [10, 1], but there are strong lower bounds for the case of weighted flow-time even on a single machine [5]. In the restricted assignment settings with preemption, the unweighted flow-time problem becomes considerably harder even for 3 machines [7]. The situation for non-preemptive flow-time is much harder. Kellerer et al. [14] showed that one cannot achieve $o(n)$-competitive algorithm even for a single machine.

Much stronger results are known in the speed augmentation model, where machines in the online algorithm have ε-fraction more speed than the corresponding machines in the offline setting. This model was first proposed by Kalyanasundaram and Pruhs [13] for the problem of non-clairvoyant preemptive total flow-time minimization on a single machine. They gave an $O(1/\varepsilon)$-competitive algorithm for this problem. Chadha et al. [7] gave $O(1/\varepsilon^2)$-competitive preemptive algorithm for weighted flow-time in the unrelated machines setting. This was extended to the non-clairvoyant setting by Im et al. [12]. However, the non-preemptive weighted flow-time problem has strong lower bounds in the speed augmentation model even on a single machine [16].

The rejection model was proposed by Choudhury et al. [8] in the context of load balancing and maximum weighted flow-time in the restricted assignment setting. Lucarelli et al. [16] considered the non-preemptive scheduling problem of minimizing weighted flow-time in the unrelated machines setting. They showed that one can get $O(1/\varepsilon)$-competitive algorithm if we allow both $(1+\varepsilon)$-speed augmentation and rejection of jobs of total weight ε-times the total weight. Assuming both, we can design a much simpler algorithm and use the dual fitting techniques developed for speed augmentation models to give a simple analysis of this algorithm. Independently of us, Lucarelli et al. [17] recently announced an algorithm where they can remove the speed augmentation assumption for the simpler unweighted setting.

In the *prize-collection* model, one is allowed to incur a penalty term for the rejected jobs. This model has been widely studied, see e.g. Bartal et al. [6], Eppstein et al. [9], and Bansal et al. [4], though is considerably different from our model because here one can reject a large fraction of the jobs.

2 Definitions and Preliminaries

We consider the unrelated machine scheduling problem, as defined in §1. Our schedules will be non-preemptive. For a schedule $\mathcal{S}$, let $C_j^{\mathcal{S}}$ denote the completion time of j. We use $F_j^{\mathcal{S}}$ to denote the flow-time of j, and the objective function is given by $F^{\mathcal{S}} := \sum_j w_j \cdot F_j^{\mathcal{S}}$. We may remove the superscript $\mathcal{S}$ if it is clear from the context. We use $\mathcal{O}$ to denote the optimal off-line schedule. In Section 3, when considering the special case of a single machine, we use p_j to denote the *processing time* of job j (on this machine). Define the *density* ρ_j of a job as the ratio w_j/p_j. We assume that the parameter ε satisfies $\varepsilon^2 \leq 1/2$, and that $1/\varepsilon \in \mathbb{Z}$.

Fractional weighted flow-time. Given a schedule $\mathcal{A}$, let $p_j(t)$ denote the remaining processing time of job j at time t (assuming $t \geq r_j$). The remaining weight of j at time t is defined as $w_j(t) := \rho_j \cdot p_j(t)$. The weighted flow-time of j in this schedule is defined as $w_j(C_j - r_j)$, where C_j is the completion time of j. The fractional weighted flow-time of j is defined as $\sum_{t \geq r_j} w_j(t)$. Since $w_j(t) = 0$ for $t \notin [r_j, C_j]$, and $w_j(t) \leq w_j$ for any time t, it is clear that the fractional weighted flow-time is at most the (integral) weighted flow-time of j. The following claim is easy to check.

▶ **Claim 3.** If a job j is processed without interruption during $[t, t+p_j]$, then its fractional weighted flow-time is $w_j(t-r_j) + {}^{w_j p_j}/_2$. Moreover, if a job j gets rejected at time t', its weighted fractional flow-time is at least ${}^{w_j(t'-r_j)}/_2$.

Since the integral weighted flow-time of a job as in the claim above is $w_j(t-r_j)+w_j p_j$, we see the integer and fractional flow times are within factor of 2 of each other. Thus, for jobs which do not get preempted, we can argue about weighted fractional flow-time.

3 Algorithm for Single-Machine Weighted Flow Time

In this section, we consider the single-machine setting. For ease of algorithm description, we assume that all quantities are integers so that we can schedule jobs at the level of integer time-slots. We first describe an algorithm $\mathcal{A}$ which both rejects and preempts jobs. We subsequently show how to modify this algorithm (in an online manner) to another schedule which only rejects jobs, and does no preemptions. During our algorithm, we shall say that a job j is *active* at time t if it has been released by time t, but has not finished processing until time t, and has not been rejected. Let $A(t)$ denote the set of active jobs at time t in our algorithm. A subset of these jobs, denoted by $L(t)$, will be special – these jobs are allowed to be preempted (at time t). Once a job enters the set $L(t)$ at some time t, it stays in $L(t')$ for all subsequent times $t' \geq t$ until it finishes processing.

For a job $j \in A(t)$ and time t, recall that $p_j(t)$ denotes the remaining processing time. At every point of (integer) time t, the algorithm performs the following steps (in this order):

1. If job j arrives at time t, the algorithm may choose to reject it immediately upon arrival. We will call such rejections *immediate rejections*. If the job is not rejected, it gets added to the active set $A(t)$. For the moment, this is the only way in which a job gets rejected.
2. Let j be the job getting processed just before time t (i.e., in the time-slot $[t-1, t]$). If job j was not already in the set $L(t)$, the algorithm may move it to the set $L(t)$ if "many" jobs smaller than j have arrived during its execution. We will specify the precise rule soon. Recall that once added, the job j will remain in the set $L(t)$ until it finishes.
3. If the job j getting processed in the time-slot $[t-1, t]$ did not finish at time t and it is not in $L(t)$, the algorithm will continue to process j during the next time-slot $[t, t+1]$. Otherwise, if j finishes or $j \in L(t)$, the algorithm chooses a job in $A(t)$ which has the highest density (the HDF rule) and processes it during $[t, t+1]$.

Note that if multiple jobs arrive at a time t, we consider them in arbitrary order, and carry out the first two steps above iteratively for each such job, before executing step 3. This completes the description of the algorithm, except that we have not specified the rules for the first two steps.

We first explain the rule for adding a job to $L(t)$. Suppose the algorithm processes a job j during $[t-1, t]$, and suppose $j \notin L(t-1)$. Let t' be the time when the algorithm started processing j. Since it was not allowed to preempt j, it must have processed j without interruption during $[t', t]$. If the total weight of jobs arriving during $(t', t]$ exceeds w_j/ε, we add job j to the set $L(t)$. The intuition behind this rule is simple – the final algorithm will eventually reject all jobs which get added to the set $L(t)$, for all t. We can charge the weight of the rejected job j to the weight of the jobs which arrived during $[t', t]$. Moreover, consider a job j that does not get added to $L(t)$ over its lifetime. In a preemptive setting, we may have preempted such a job j on the arrival of a new shorter job, whereas here we perform such a preemption only when enough shorter jobs arrive. Since j was not added to $L(t)$, the total weight of such shorter jobs waiting on j is at most w_j/ε, so we can pay for the additional flow-time incurred by these shorter jobs (up to an $1/\varepsilon$ factor) by the flow-time of j.

The rule for immediate rejections is more involved. We maintain two tables T^+ and T^-. Each arriving job may get assigned to either T^+ or T^-, or both. We refer to each entry of these tables as a *bucket*. At a high level, every $(1/\varepsilon)^{th}$ job arriving in each bucket in either table suffers immediate rejection, though the details differ for the two tables. Let us elaborate on this further.

With every newly arriving job j, we specify a quantity α_j, which is the increase in the total flow-time of all the jobs in the system, assuming (i) no further jobs arrive after job j, and (ii) the scheduling algorithm follows the preemptive HDF policy from r_j onwards for *all* the jobs in $A(r_j)$. As in [2], we can write an expression for α_j as follows.

$$\alpha_j := \Big(w_j \sum_{j' \in A(r_j):\ \rho_{j'} \geq \rho_j} p_{j'}(r_j)\Big) \ + \ {}^{w_j p_j}/_2 \ + \ \Big(p_j \sum_{j' \in A(r_j):\ \rho_{j'} < \rho_j} w_{j'}(r_j)\Big). \tag{1}$$

We establish the convention that $A(r_j)$ does not contain job j. Moreover, if multiple jobs are released at time r_j, we consider them in arbitrary but fixed order, and add only those jobs to $A(r_j)$ which are considered before j.

For $x \in \mathbb{R}$, let $\llbracket x \rrbracket$ denote the largest integer i such that $2^i \leq x$. For a job j, define its *density-class* as $\llbracket \rho_j \rrbracket$. We partition jobs in $A(r_j)$ depending on their density-class as follows:

$$D_j^+ := \{j' \in A(r_j) \mid \llbracket \rho_{j'} \rrbracket \geq \llbracket \rho_j \rrbracket\} \qquad \text{and} \qquad D_j^- := \{j' \in A(r_j) \mid \llbracket \rho_{j'} \rrbracket < \llbracket \rho_j \rrbracket\}. \tag{2}$$

Now let α_j^+ be the terms in the expression for α_j involving jobs in D_j^+, and define α_j^- similarly. In other words,

$$\alpha_j^+ := \Big(w_j \sum_{j' \in D_j^+ :\ \rho_{j'} \geq \rho_j} p_{j'}(r_j)\Big) + \Big(p_j \sum_{j' \in D_j^+ :\ \rho_{j'} < \rho_j} w_{j'}(r_j)\Big); \tag{3}$$

$$\alpha_j^- := \Big(p_j \sum_{j' \in D_j^- :\ \rho_{j'} < \rho_j} w_{j'}(r_j)\Big). \tag{4}$$

Clearly, $\alpha_j = \alpha_j^+ + {}^{w_j p_j}/_2 + \alpha_j^-$. We now specify the definitions of the two tables.

- Table T^+: Buckets in this table are indexed by ordered pairs of integers (κ, λ). If an arriving job j satisfies $\alpha_j^+ \geq w_j p_j/\varepsilon$, we assign it to the bucket indexed $(\llbracket {}^{\alpha_j^+}/_{w_j} \rrbracket, \llbracket w_j \rrbracket)$ in this table, and add it to the set J^+ of jobs assigned to T^+. For each bucket, we cancel the first job that is assigned to that bucket, and then every $(1/\varepsilon)^{th}$ subsequent job assigned to it.
- Table T^-: Buckets in this table are indexed by ordered triplets of integers (γ, δ, η). Each arriving job which satisfies $\alpha_j^- > w_j p_j/\varepsilon$ is assigned to the bucket indexed $(\llbracket \alpha_j^- \rrbracket, \llbracket \rho_j \rrbracket, \llbracket p_j \rrbracket)$, and added to the set J^- of jobs assigned to T^-. For each bucket, cancel every $(1/\varepsilon)^{th}$ job assigned to this bucket. Note the subtle difference with respect to T^+: here the first job to be canceled in a bucket is the $(1/\varepsilon)^{th}$ job assigned to it.

3.1 The Final Algorithm $\mathcal{B}$

The actual online algorithm $\mathcal{B}$ is almost the same as $\mathcal{A}$, except when the algorithm $\mathcal{A}$ processes a job in $L(t)$ during time-slot $[t, t+1]$, the algorithm $\mathcal{B}$ idles, leaving this slot empty. In other words, when a job being executed is added to $L(t)$, the algorithm $\mathcal{B}$ rejects the job instead of eventually finishing it, perhaps after some preemptions. (We can think of this as being a *delayed rejection*, as opposed to the *immediate rejection* that $\mathcal{A}$ performs based on the above bucketing strategy.) Clearly, we can implement $\mathcal{B}$ in an online manner.

4 Analyzing the Single-Machine Algorithm

In this section, we provide the analysis of our single-machine algorithm $\mathcal{B}$. Naturally, the two main steps are to show that (i) an $O(\varepsilon)$ fraction of jobs by weight get rejected, and (ii) the total flow time is competitive with the optimal offline algorithm.

Showing (i) is relatively straightforward: a rejected job is either immediately rejected or is later rejected in $\mathcal{B}$ due to its preemption in $\mathcal{A}$. We will show that the rejected jobs falling under each of the two categories is an $O(\varepsilon)$ fraction by weight, with a separate analysis for each category. Both of the analyses are in Section 4.1.

To show flow time competitiveness of algorithm $\mathcal{B}$, we instead focus on bounding the tota flow time of algorithm $\mathcal{A}$. By Claim 3, the total (integer) flow-time of jobs that $\mathcal{B}$ does not reject is within a factor of two of their fractional flow-time in $\mathcal{A}$, since these are precisely the jobs that $\mathcal{A}$ does not preempt. Therefore, to prove Theorem 2, it suffices to show that $\mathcal{A}$ is $O(1/\varepsilon^2)$ factor competitive with the optimal offline algorithm.

Let $J^{\texttt{immed}}$ denote the set of jobs which get rejected immediately upon arrival, and let $\mathcal{O}$ denote the optimal offline schedule and $F^{\mathcal{O}}$ its fractional weighted flow time. Roughly speaking, our goal is to establish the following chain of approximate inequalities:

$$\varepsilon F^{\mathcal{A}} \lesssim \varepsilon \sum_j \alpha_j \lesssim \sum_{j \in J^{\texttt{immed}}} \alpha_j \lesssim F^{\mathcal{O}}, \tag{5}$$

where $\lesssim$ hides additive $\sum_j {}^{w_j p_j}/_{\varepsilon^{O(1)}}$ factors. Since $F^{\mathcal{O}} \geq \sum_j w_j p_j/2$, these additive losses still provide a ${}^1/_{\varepsilon^{O(1)}}$ competitive ratio.

For the first inequality, we will bound the flow time of algorithm $\mathcal{A}$, modulo an additive $\sum_j {}^{w_j p_j}/_{\varepsilon}$ factor, by the sum of α_j over all jobs $j \notin J^{\texttt{immed}}$, which are precisely the jobs that are finished by $\mathcal{A}$. We do so by exploiting the facts that the α_j values indicate an increase in flow time to an HDF algorithm, and that $\mathcal{A}$ is "approximately" an HDF algorithm. The details are in Lemma 5.

The second inequality is the most technically involved section of the paper. Not only does the immediate rejection scheme reject an $O(\varepsilon)$ fraction of jobs, but it also rejects jobs constituting an ε fraction of the total α_j value. The analysis is in Section 4.3.

Finally, the last inequality relates the optimal offline flow time to the sum of the α_j values of immediately rejected jobs. It is restated as Lemma 6 and proved in the appendix.

4.1 Bounding Weight of Rejected Jobs

In this section, we show that the total weight of rejected jobs is only an $O(\varepsilon)$ fraction of total. Recall that jobs either suffer immediate rejection, or are added to $L(t)$ for some time t, and hence suffer delayed rejection.

Let us first bound the total weight of the set $L := \cup_t L(t)$. For a job j in $L(t)$, let s_j be the first time when it gets processed and l_j be the time at which it enters the set $L(t)$. Since j must be processed uninterrupted in this interval $(s_j, l_j]$, the intervals associated with different jobs are disjoint. Moreover job j entered $L(t)$ because the total weight of jobs released during $(s_j, l_j]$ is at least w_j/ε. Thus the total weight of jobs in L can be upper bounded by ε times the weight of all the jobs.

We now account for the weight of jobs which are rejected immediately on arrival. For job j, let $[\![w_j]\!]$ denote the *weight-class* of this job. Jobs assigned to a bucket in T^+ have the same weight-class, by construction of the buckets. Jobs assigned to a bucket in T^- have the same $[\![\rho_j]\!]$ and $[\![p_j]\!]$, which pins down their weight $w_j = \rho_j \cdot p_j$ up to a factor of 4. This gives us the following facts:

- Since we reject every $(1/\varepsilon)^{th}$ job in each bucket of T^-, the total weight of jobs in J^- which get rejected immediately is at most 4ε times the weight of all jobs in J^-.
- Let J_f^+ be the subset of jobs in J^+ which happen to be the first jobs to be assigned to their respective buckets in T^+. Then the weight of all jobs in $J^+ \setminus J_f^+$ which get rejected immediately on arrival is at most 2ε times the total weight of all the jobs in J^+.

So it remains to account for the items items in J_f^+, which are all rejected. Recall that a job in J^+ is assigned to the bucket indexed $(\llbracket \alpha_j^+/w_j \rrbracket, \llbracket w_j \rrbracket)$ in T^+. Jobs in J_f^+ are assigned to distinct buckets in T^+. Fix an integer γ, and let J^γ denote the jobs in J_f^+ which are mapped to a bucket indexed (γ, κ) for some κ. The jobs in J^γ have distinct weight-classes and so it suffices to bound the weight of the highest weight job in J^γ – let this heaviest job be j^γ. Let S denote the set of such jobs j^γ as we range over all γ. Jobs in S have distinct $\llbracket \alpha_j^+/w_j \rrbracket$ values. Let $\Gamma = \{\gamma_1 < \gamma_2 < \ldots < \gamma_k\}$ be the integers γ for which there is a job $j^\gamma \in S$, and let the corresponding jobs in S be called $j_1, j_2, \ldots, j_k$.

Now starting from the smallest index in Γ, we charge each job $j_r \in S$ to a subset of jobs of total weight at least w_{j_r}/ε. The job j_r may charge to a job fractionally – if it charges to a fraction δ of some job j, then it can only use δw_j amount of weight of j for its charging (and we say that "j_r *charges to* δp_j *size of this job* j"). Of course, we need to ensure that the total fraction charged to a job is at most 1. We inductively maintain the following invariant for all $r \in 1 \ldots k$:

- The job j_r charges to jobs of total (fractional) weight at least $w_{j_r}/8\varepsilon$.
- Jobs $j_1, \ldots, j_r$ charge to jobs of total (fractional) size at most 2^{γ_r}.

Assuming these invariants hold for $r-1$, we show that they hold for r as well. Let $\rho^\star := \llbracket \rho_{j_r} \rrbracket$ be the density class for job j_r. By j_r's choice of bucket, $\llbracket \alpha_{j_r}/w_{j_r} \rrbracket = \gamma_r$, so

$$\alpha_{j_r}^+ \geq 2^{\gamma_r} \cdot w_{j_r}. \tag{6}$$

Recall from (2) that $D_{j_r}^+$ is the set of jobs of density class $\rho^\star$ or higher which are active at the time j_r is released. Let $P_r := \sum_{j \in D_{j_r}^+} p_j$ be the total processing time of these jobs. By (3), it follows that

$$\alpha_{j_r}^+ \leq w_{j_r} P_r. \tag{7}$$

Combining (6) and (7), $P_r \geq 2^{\gamma_r}$. By the second invariant, the first r jobs $j_1, \ldots, j_{r-1}$ have only charged to jobs of total size at most $2^{\gamma_{r-1}}$, so we can find jobs in $D_{j_r}^+$ of total (fractional) size $2^{\gamma_r} - 2^{\gamma_{r-1}} \geq 2^{\gamma_r - 1}$ which have not been charged yet, and charge to them. This proves the second invariant.

To prove the first invariant, we know that $\alpha_{j_r}^+ \geq w_{j_r} p_{j_r}/\varepsilon$, else j_r would not be assigned to T^+. Moreover, $\alpha_{j_r}^+ \leq w_{j_r} 2^{\gamma_r + 1}$ by the bucketing, so $2^{\gamma_r} \geq p_{j_r}/2\varepsilon$. Consequently, we charge to jobs of total size at least $2^{\gamma_r - 1} \geq p_{j_r}/4\varepsilon$, and these jobs have density class at least $\rho^\star$. Since $2\rho^\star \geq \rho_{j_r} = w_{j_r}/p_{j_r}$, we get their total (fractional) weight is at least $w_{j_r}/8\varepsilon$. This proves the first invariant, and hence the following theorem.

▶ **Theorem 4** (Few Rejections). *The weight of jobs suffering immediate rejection, plus those in $\cup_t L(t)$, is at most an $O(\varepsilon)$ fraction of the weight of all jobs released.*

4.2 Bounding the Weighted Fractional Flow-time

Next we show that the total fractional flow-time of $\mathcal{A}$ can be bounded in terms of total α_j values. We first focus on relating $F^{\mathcal{A}}$ to the sum of the α_j values, as described in (5).

Observe that α_j denotes the increase in objective function due to the arrival of j if we had followed the preemptive HDF policy for all the jobs from time r_j onwards. However, we follow a slightly different policy – if j' denotes the job that was running on the machine at time j's release time r_j, we let j' run until it finishes, or else until j' belongs to the set $L(t')$ at some time $t' \geq r_j$. If no further jobs are released after j, the HDF policy after this time t' would be non-preemptive. Thus, we would still expect that the total fractional weighted flow-time of our algorithm to be close to $\sum_j \alpha_j$. We formalise this intuition now. For every job j, we define a job $\phi(j)$ as follows: let j' be the job which was running just before time r_j (i.e., in the slot $[r_j - 1, r_j]$). If $j' \notin L(r_j)$, we define $\phi(j)$ to be j', otherwise we leave $\phi(j)$ undefined. Our policy for adding a job to the set $L(t)$ ensures that for every job j, $w(\phi^{-1}(j))$ is at most w_j/ε. [1] Recall that $J^{\texttt{immed}}$ is the set of jobs which get rejected immediately upon arrival. The following lemma, whose proof is deferred to the full version, states that the fractional weighted flow-time of the algorithm can be charged to the α_j values of the jobs which get immediately rejected.

▶ **Lemma 5.** *The fractional weighted flow-time of $\mathcal{A}$ is at most $\sum_{j:j\notin J^{\texttt{immed}}} \alpha_j + \sum_j w_j p_j/\varepsilon$.*

Proof. Jobs in $J^{\texttt{immed}}$ get rejected immediately, so their flow-time is 0. We now consider the jobs which are not immediately rejected in the rest of the proof. Consider the jobs in order of increasing release times. Let Δ_j denote the increase in the objective function value due to arrival of j. In other words, if J_1 is the set of jobs released before j, then Δ_j equals the total fractional weighted flow-time of $\mathcal{A}$ on the input $J_2 := J_1 \cup \{j\}$ minus that on the input J_1. The total weighted flow time of $\mathcal{A}$ on the entire input would be $\sum_j \Delta_j$, the sum of these increases. We now show that

$$\Delta_j \leq \alpha_j + w_j p_{\phi(j)}. \tag{8}$$

Since $w(\phi^{-1}(j')) \leq w_{j'}/\varepsilon$, we get that $\sum_j w_j p_{\phi(j)} = \sum_{j'} w(\phi^{-1}(j'))p_{j'} \leq \sum_{j'} w_{j'} p_{j'}/\varepsilon$. Hence, summing (8) over all j which are not in $J^{\texttt{immed}}$ proves the lemma.

Now we prove (8). Since we will be dealing with two inputs, J_1 and J_2, we parameterise all quantities by J_1 or J_2 to clarify which input we refer to. For example, $A(J_k, t), k = 1, 2$ will refer to the active set $A(t)$ on input J_k. Let $F(J_k, t)$ denote the fractional weighted flow-time of jobs in $A(J_k, t)$ *beyond* time t, i.e., $F(J_k, t) := \sum_{t' \geq t} \sum_{j \in A(J_k,t')} w_j(t')$.

There are two cases when job j arrives. If $\phi(j)$ is undefined, the job j' running in slot $[r_j - 1, r_j]$ belongs to $L(r_j)$. Hence the algorithm $\mathcal{A}$ on both inputs J_1, J_2 just runs HDF starting at time r_j. The difference between the corresponding flow times is precisely α_j, by definition.

Otherwise $\phi(j)$ is well-defined. Since j is the latest arrival, the job $\phi(j)$ will not be preempted, and runs to completion. Say job $\phi(j)$ completes at time t'. During the time $[r_j, t']$ the difference in fractional weighted flow-time between the two runs is precisely $w_j \cdot (t' - r_j) \leq w_j p_{\phi(j)}$. After time t' we run HDF on the remaining jobs, and the difference in the fractional weighted flow-time of the two runs is precisely what α_j would have been had j arrived at time t' instead of time r_j. In other words, if $J' := A(J_k, r_j) \setminus \{\phi(j)\}$,

$$\begin{aligned} F(J_2, t') - F(J_1, t') &= w_j p_j/2 + \sum_{j' \in J' : \rho_{j'} \geq \rho_j} w_j p_{j'}(t') + \sum_{j' \in J' : \rho_{j'} < \rho_j} w_{j'}(t') p_j \\ &= w_j p_j/2 + \sum_{j' \in J' : \rho_{j'} \geq \rho_j} w_j p_{j'}(r_j) + \sum_{j' \in J' : \rho_{j'} < \rho_j} w_{j'}(r_j) p_j \end{aligned}$$

[1] For a set S of jobs, let $w(S)$ denote the total weight of jobs in S.

But this is a subset of the terms of α_j: indeed, we're just missing the term corresponding to job $\phi(j)$. Hence, the total difference is at most $\alpha_j + w_j p_{\phi(j)}$, proving (8). ◀

To bound our flow time against the optimum using this lemma, note that $\sum_j w_j p_j/\varepsilon \leq 2F^{\mathcal{O}}/\varepsilon$, where we recall that $\mathcal{O}$ denotes the optimal offline schedule, and $F^{\mathcal{O}}$ its fractional weighted flow time. So we just need to bound $\sum_j \alpha_j = \sum_j {}^{w_j p_j}/_2 + \sum_j \alpha_j^+ + \sum_j \alpha_j^-$. The first term is again bounded by $F^{\mathcal{O}}$, so the work is in bounding the other two terms. We first record a convenient lemma – its proof is based on LP duality arguments and construction of dual variables are similar to those in [2], and is deferred to the full version.

▶ **Lemma 6** (Duality-based Lower Bound on OPT). $\sum_{j \in J^{\texttt{immed}}} \alpha_j \leq F^{\mathcal{O}} + \sum_j {}^{w_j p_j}/_{\varepsilon}$.

4.3 Controlling the α Terms

In this section, our goal is to establish the approximate inequality $\varepsilon \sum_j \alpha_j \lesssim \sum_{j \in J^{\texttt{immed}}} \alpha_j$, introduced in (5).

▶ **Lemma 7.** $\sum_j \alpha_j^+ \leq O(1/\varepsilon) \cdot \big(\sum_j w_j p_j + \sum_{j \in J^{\texttt{immed}}} \alpha_j^+\big)$.

Proof. The definition of J^+ implies that $\sum_{j \notin J^+} \alpha_j^+ \leq \sum_{j \notin J^+} w_j p_j/\varepsilon$. It remains to bound $\sum_{j \in J^+} \alpha_j^+$. We do an accounting per bucket in T^+. Fix a bucket B indexed by a pair (κ, λ), i.e., all jobs j in this bucket have $\llbracket \alpha_j^+/w_j \rrbracket = \kappa$, and $\llbracket w_j \rrbracket = \lambda$. Hence, if j is any job in this bucket, then $2^\kappa \leq \alpha_j^+/w_j \leq 2^{\kappa+1}$, and $2^\lambda \leq w_j \leq 2^{\lambda+1}$. Multiplying, $2^{\kappa+\lambda} \leq \alpha_j^+ \leq 4 \cdot 2^{\kappa+\lambda}$, i.e., the α_j^+ values of any two jobs in this bucket differ by a factor of at most 4.

Let J_B denote the jobs in J^+ assigned to this bucket B, and n_B denote their cardinality $|J_B|$. Since we reject the first job and then every subsequent $(1/\varepsilon)^{th}$ job in J_B, we immediately reject at least $\varepsilon\, n_B$ jobs in J_B. Therefore,

$$\sum_{j \in J_B} \alpha_j^+ \leq \frac{4}{\varepsilon} \cdot \sum_{j \in J_B \cap J^{\texttt{immed}}} \alpha_j^+.$$

Summing over all buckets, the lemma follows. ◀

▶ **Lemma 8.** $\sum_j \alpha_j^- \leq O(1/\varepsilon) \cdot \big(\sum_j w_j p_j + \sum_{j \in J^{\texttt{immed}}} \alpha_j\big)$.

Proof. The argument is similar to Lemma 7 in spirit, but technically more involved. The reason is that we do not remove any jobs from a bucket of T^- until it has $1/\varepsilon$ jobs assigned to it. Hence, for a bucket B, if J_B is non-empty but $|J_B| \leq 1/\varepsilon$, we have $J_B \cap J^{\texttt{immed}} = \varnothing$. However, if J_f^- is the set of jobs in J^- which are the first jobs assigned to their corresponding buckets in T^-, then we get (as in the proof of Lemma 7) that

$$\sum_j \alpha_j^- \leq O(1/\varepsilon) \cdot \left(\sum_j w_j p_j + \sum_{j \in J^{\texttt{immed}}} \alpha_j^- + \sum_{j \in J_f^-} \alpha_j^- \right). \tag{9}$$

It remains to bound $\sum_{j \in J_f^-} \alpha_j^-$, which we accomplish via the following claim. Since the proof is more technical, we defer it to the next section.

▶ **Claim 9.** $\sum_{j \in J_f^-} \alpha_j^- \leq O(\varepsilon) \cdot \big(\sum_j w_j p_j + \sum_j \alpha_j^+\big)$.

Combining this with (9) and Lemma 7, using that $\alpha_j^+ + {}^{w_j p_j}/_2 + \alpha_j^- = \alpha_j$, the lemma follows. ◀

Combining Lemmas 7 and 8 along with Lemma 5 and Lemma 6, we get

▶ **Theorem 10.** *The fractional weighted flow-time of the non-rejected jobs in $\mathcal{A}$ is $O(F^{\mathcal{O}}/\varepsilon^2)$.*

4.3.1 Proof of Claim 9

Define $\Lambda^+ := \sum_j \alpha_j^+$. Recall that for a job j, its density class is given by $\lfloor \rho_j \rfloor = \lfloor w_j/p_j \rfloor$. For each density class $\delta \in \mathbb{Z}$, let us define some notation:

- Let $A^\delta(t) := \{j \in A(t) \mid \lfloor \rho_j \rfloor = \delta\}$ denote jobs in $A(t)$ whose density class is δ.
- Let $P^\delta(t) := \sum_{j \in A^\delta(t)} p_j(t)$ and $W^\delta(t) := \sum_{j \in A^\delta(t)} w_j(t)$ be the total processing time and residual weight of jobs in $A^\delta(t)$, respectively. Since all jobs in this set have the same density class, observe that $\frac{W^\delta(t)}{P^\delta(t)}$ also lies in the range $[2^\delta, 2^{\delta+1})$.
- Define $P^\delta := \max_t P^\delta(t)$ and $W^\delta := \max_t W^\delta(t)$.

Our proof shows that $\sum_\delta P^\delta W^\delta$ is small; then we bound $\sum_{j \in J_f^-} \alpha_j^-$ by $\sum_\delta P^\delta W^\delta$. The proof of the following technical lemma is deferred to the full version.

▶ **Lemma 11.** $\sum_\delta P^\delta W^\delta \le O(1) \cdot \left(\sum_j w_j p_j + \Lambda^+ \right)$.

▶ **Lemma 12.** $\sum_{j \in J_f^-} \alpha_j^- \le O(\varepsilon) \cdot \sum_\delta P^\delta W^\delta$.

Proof. Let us first give a general method for bounding α_j^- of any job $j \in J^-$, and then we can apply it to the jobs in $J_f^- \subseteq J^-$. Recall that the jobs which contribute to α_j^- are the ones with a strictly smaller density class than that of j. We now show that one need not look at jobs of all such classes, and a subset of these classes suffice. Fix a job $j \in J^-$ of density class δ, and define an index set $\mathbb{I}_j$ as $\{\theta < \delta \mid P^\theta(r_j) \ge (1.5)^{\delta-\theta} p_j/8\varepsilon\}$.

▶ **Claim 13.** For any job $j \in J^-$ with density class δ, $\alpha_j^- \le 4p_j \cdot \sum_{\theta \in \mathbb{I}_j} W^\theta$.

Proof. Let j' be a job in $A(r_j)$ of strictly lower density class than j. Its contribution towards α_j^- is $p_j w_{j'}(r_j)$. Therefore, α_j^- is at most

$$\sum_{\theta<\delta} p_j W^\theta(r_j) = p_j \cdot \sum_{\theta \in \mathbb{I}_j} W^\theta(r_j) + p_j \cdot \sum_{\theta \notin \mathbb{I}_j, \theta<\delta} W^\theta(r_j). \tag{10}$$

Let us bound the summation from the second expression.

$$\sum_{\theta \notin \mathbb{I}_j, \theta<\delta} W^\theta(r_j) \le \sum_{\theta \notin \mathbb{I}_j, \theta<\delta} 2^{\theta+1} P^\theta(r_j) \le \sum_{\theta<\delta} \frac{(1.5)^{\delta-\theta}}{2^{\delta-\theta}} \cdot \frac{2^\delta p_j}{4\varepsilon} \le \frac{3w_j}{4\varepsilon}. \tag{11}$$

Substituting (11) into (10), and using that $\alpha_j^- \ge w_j p_j/\varepsilon$ for all jobs $j \in J^-$, we get that $\alpha_j/4 \le p_j \sum_{\theta \in \mathbb{I}_j} W^\theta(r_j) \le p_j \sum_{\theta \in \mathbb{I}_j} W^\theta$, which proves the desired result. ◀

Recall that job $j \in J^-$ is mapped in table T^- to the bucket indexed by $(\lfloor \alpha_j^- \rfloor, \lfloor \rho_j \rfloor, \lfloor p_j \rfloor)$. For a fixed pair (δ, η), consider the jobs in J_f^- which are mapped to buckets indexed (γ, δ, η) with various values of γ, and denote these jobs by $J_{(\delta,\eta)}$. Since J_f^- only contains the first job in each bucket, the $\lfloor \alpha_j^- \rfloor$ values of the various jobs in $J_{(\delta,\eta)}$ are all distinct. It follows that if $j^\star$ is the job in $J_{(\delta,\eta)}$ with the highest α_j^- value, then $\sum_{j \in J_{(\delta,\eta)}} \alpha_j^- \le 4\alpha_{j^\star}^-$. Thus, we just need to worry about one job per $J_{(\delta,\eta)}$ – let S denote this set of jobs.

The ordered pairs $(\lfloor \rho_j \rfloor, \lfloor p_j \rfloor)$ corresponding to jobs $j \in S$ are all distinct. For density class δ, let S^δ denote the jobs in S with density class δ. Using Claim 13,

$$\sum_{j \in S^\delta} \alpha_j^- \le 4 \sum_{j \in S^\delta} p_j \sum_{\theta \in \mathbb{I}_j} W^\theta = 4 \sum_{\theta<\delta} W^\theta \sum_{j \in S^\delta : \theta \in \mathbb{I}_j} p_j. \tag{12}$$

The jobs in S^δ also have different $[\![p_j]\!]$ values, so the sum $\sum_{j\in S^\delta:\theta\in\mathbb{I}_j} p_j \le 4p_{j'}$ for the job $j' := \arg\max\{p_j \mid j \in S^\delta, \theta \in \mathbb{I}_j\}$. By definition of $\mathbb{I}_j$, $p_{j'} \le 8\varepsilon P^\theta/(1.5)^{\delta-\theta}$. Substituting into (12),

$$\sum_{j\in S^\delta} \alpha_j^- \le 16 \sum_{\theta<\delta} \frac{8\varepsilon\, W^\theta P^\theta}{(1.5)^{\delta-\theta}}. \tag{13}$$

To complete the argument, $\sum_{j\in J_f^-} \alpha_j^-$ is at most

$$4\sum_\delta \sum_{j\in S^\delta} \alpha_j^- \overset{\text{eq.(13)}}{\le} 2^9\varepsilon \sum_\delta \sum_{\theta<\delta} \frac{W^\theta P^\theta}{(1.5)^{\delta-\theta}} = 2^9\varepsilon \sum_\theta W^\theta P^\theta \cdot \sum_{\delta>\theta} \frac{1}{(1.5)^{\delta-\theta}} = O\Big(\varepsilon \sum_\theta W^\theta P^\theta\Big).$$

This completes the proof of Lemma 12. ◀

Combining Lemmas 11 and 12 completes the proof of Claim 9, and hence for Theorem 10. In the full version of the paper we show that the algorithm is competitive even against an optimal algorithm that is allowed $(1+\varepsilon)$-speed augmentation – and hence prove Theorem 2.

5 Extension to Unrelated Machines

The extension of our result on single machine to the more general scenario of unrelated machines can be done very modularly. We shall use the following result from [7, 2].

▶ **Theorem 14.** *There is an online algorithm $\mathcal{D}$ which dispatches each arriving job j immediately upon arrival to one of the m machines such that the following property holds: if $J^{(i)}$ is the set of jobs which are dispatched to machine i and $\mathcal{O}^{\varepsilon',i}$ is the optimal solution to $J^{(i)}$ when we have only one machine with speed $(1+\varepsilon')$, then $\sum_i F^{\mathcal{O}^{\varepsilon',i}}$ is at most $1/\varepsilon'$ times the optimal weighted flow-time of J.*

The algorithms in [7, 2] actually build a schedule as well and use this schedule to immediately dispatch a job. The algorithm $\mathcal{D}$ can build this schedule in the *background* and use it to dispatch jobs, but not use it for actual processing. It follows from Theorem 14 and our result showing that our algorithm is also competitive against an optimal algorithm that is allowed $(1+\varepsilon)$-speed augmentation (which we defer to the full version), that if we run our algorithm on each of the machines i (with input $J^{(i)}$ arriving on-line) independently, then the total weighted flow-time of non-rejected jobs in our algorithm is at most $O(1/\varepsilon^3)$ times the optimal value.

References

1 S Anand. *Algorithms for flow time scheduling.* PhD thesis, Indian Institute of Technology, Delhi, 2013.

2 S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA'12*, pages 1228–1241. ACM, New York, 2012.

3 Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *SPAA*, pages 11–18, 2003.

4 Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Kedar Dhamdhere. Scheduling for flow-time with admission control. In *Proc. ESA, 2003*, pages 43–54. Springer, 2003. doi:10.1007/978-3-540-39658-1_7.

5 Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *SODA*, pages 1238–1244, 2009.

6 Yair Bartal, Stefano Leonardi, Alberto Marchetti-Spaccamela, Jiri Sgall, and Leen Stougie. Multiprocessor scheduling with rejection. *SIAM J. Discrete Math.*, 13(1):64–78, 2000.

7 Jivitej S. Chadha, Naveen Garg, Amit Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation. In *STOC'09*, pages 679–683. ACM, New York, 2009.

8 Anamitra Roy Choudhury, Syamantak Das, Naveen Garg, and Amit Kumar. Rejecting jobs to minimize load and maximum flow-time. *J. Comput. System Sci.*, 91:42–68, 2018.

9 Leah Epstein and Hanan Zebedat-Haider. Preemptive online scheduling with rejection of unit jobs on two uniformly related machines. *J. Scheduling*, 17(1):87–93, 2014.

10 Naveen Garg and Amit Kumar. Better algorithms for minimizing average flow-time on related machines. In *ICALP*, volume 4051, pages 181–190. 2006.

11 Naveen Garg and Amit Kumar. Minimizing average flow-time : Upper and lower bounds. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 603–613, 2007.

12 Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 531–540, 2014.

13 Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

14 Hans Kellerer, Thomas Tautenhahn, and Gerhard J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. *SIAM J. Comput.*, 28(4):1155–1166, 1999.

15 Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *Journal of Computer and Systems Sciences*, 73(6):875–891, 2007.

16 Giorgio Lucarelli, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online non-preemptive scheduling in a resource augmentation model based on duality. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 63:1–63:17, 2016.

17 Giorgio Lucarelli, Nguyen Kim Thang, Abhinav Srivastav, and Denis Trystram. Online min-sum flow scheduling with rejections. In *In 13th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP 2017), 2017*, 2017.

Maximizing Profit with Convex Costs in the Random-order Model*

Anupam Gupta[1]
Carnegie Mellon University, Pittsburgh, USA
anupamg@cs.cmu.edu

Ruta Mehta[2]
University of Illinois Urbana-Champaign, Champaign, USA
rutameht@illinois.edu

Marco Molinaro[3]
PUC-Rio, Rio de Janeiro, Brazil
mmolinaro@inf.puc-rio.br

Abstract

Suppose a set of requests arrives online: each request gives some value v_i if accepted, but requires using some amount of each of d resources. Our cost is a convex function of the vector of total utilization of these d resources. Which requests should be accept to maximize our profit, i.e., the sum of values of the accepted demands, minus the convex cost?

We consider this problem in the random-order a.k.a. secretary model, and show an $O(d)$-competitive algorithm for the case where the convex cost function is also *supermodular*. If the set of accepted demands must also be independent in a given matroid, we give an $O(d^3\alpha)$-competitive algorithm for the supermodular case, and an improved $O(d^2\alpha)$ if the convex cost function is also separable. Here α is the competitive ratio of the best algorithm for the submodular secretary problem. These extend and improve previous results known for this problem. Our techniques are simple but use powerful ideas from convex duality, which give clean interpretations of existing work, and allow us to give the extensions and improvements.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online algorithms, secretary problem, random order, convex duality

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.71

Related Version A full version of the paper is available at `https://arxiv.org/abs/1804.08172`.

1 Introduction

The problem we consider is a basic convex optimization problem in the online setting: n items appear one-by-one. Each item/element e has a *d-dimensional size* $s(e) \in \mathbb{R}^d_+$ and a *value* $v(e) \in \mathbb{R}_+$, which are both revealed to us when the item arrives.We must either accept

* This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing.

[1] Computer Science Department, Carnegie Mellon University, Pittsburgh, USA. Supported in part by NSF awards CCF-1536002, CCF-1540541, and CCF-1617790.
[2] Supported in part by NSF award CCF-1750436, and the Simons Institute for the Theory of Computing.
[3] Supported in part by CNPq grants Universal #431480/2016-8 and Bolsa de Produtividade em Pesquisa #310516/2017-0, and a Microsoft Research Fellowship at the Simons Institute for the Theory of Computing.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 71; pp. 71:1–71:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

or reject an item when it arrives, before seeing the future items. If we accept a certain subset $A \subseteq [n]$ of the items, we get their total value $v(A) := \sum_{e \in A} v_e$, but incur a production cost $g(s(A)) := g(\sum_{e \in A} s(e))$, where $g : \mathbb{R}^d_+ \to \mathbb{R}_+$ is a non-decreasing *convex cost* function with $g(0) = 0$. Optionally, we may also be given a downwards-closed family of subsets $\mathscr{F} \subseteq 2^{[n]}$, and now the accepted set of elements A must lie in $\mathscr{F}$. More formally, we want to solve

$$\max_{A \in \mathscr{F}} \text{ profit } \pi(A) := \big[v(A) - g(s(A))\big]. \tag{1.1}$$

This question arises, e.g., when we are selling some service that depends on d commodities, where the value is the amount of money customer e is willing to pay for the service, and the size vector $s(e)$ is the amount of resources she will require. The cost function $g(\cdot)$ captures our operating expenses; its convexity models *diseconomies of scale* that arise when dealing with scarce commodities. In particular, it can capture d-dimensional knapsack constraints, by setting $g(z) = 0$ until the knapsack size, and ∞ afterwards. When the cost function is linear $g(z) = \langle a, z \rangle$, we want to pick a max-weight subset from $\mathscr{F}$ using item weights $v(e) - \langle a, s(e) \rangle$, which is tractable/approximable for $\mathscr{F}$ being a matroid, p-system, etc.

Blum et al. [6] defined this problem in the adversarial model, and gave posted-price algorithms for "low-degree" *separable* cost functions g, that is, of the form $g(z) = \sum_{i=1}^{d} g_i(z_i)$ for 1-dimensional functions g_i's. This result was tightened by Huang and Kim [16], still for separable functions with additonal growth control. More recently, Azar et al. [3] studied this problem for more general *supermodular* non-separable convex functions g (see also [9]). A differentiable function g is supermodular if for any vectors $x \leq x'$ we have $\nabla g(x) \leq \nabla g(x')$. Equivalently, if g is twice-differentiable, it is supermodular if $\frac{\partial^2 g}{\partial x_i \partial x_j} \geq 0$ for all $i \neq j$, i.e., increasing the consumption of a resource cannot decrease the marginal cost for another. However, to handle the worst-case ordering, Azar et al. also require the cost functions to have essentially low-degree.

Can we do better by going beyond the worst-case model? In this paper, we focus on the random-order or "secretary" setting, where the set of items is fixed by an adversary but they arrive in random order. In the single-dimensional case $d = 1$, it is easy to see that a solution that learns a "good" threshold λ and picks all further items with density $v(e)/s(e)$ at least λ essentially gives a constant approximation, much like in the secretary and knapsack secretary problems [13, 4]. The multi-dimensional case is much more challenging. This was studied by Barman et al. [5], again assuming a separable cost function $g(z) = \sum_{i=1}^{d} g_i(z_i)$. They give an $O(d)$-competitive algorithm for the unconstrained case, and an $O(d^5\alpha)$-competitive algorithm for the problem with a downward closed constraint set $\mathscr{F}$, where α is the competitive ratio for the $\mathscr{F}$-secretary problem. Their main idea is to perform a clever decomposition of the value of each item into "subvalues" $v_i(e)$ for each of the coordinate cost functions g_i's; this effectively decomposes the problem into d 1-dimension problems with values v_i's and costs g_i's. Unfortunately, since their solution explicitly relies on the decomposability of the cost function, it is unclear how to extend it to general supermodular functions. We note that when the cost function is supermodular, the profit function is a *submodular* set function (Section 2.1). However, the profit can take *negative values*, and then existing algorithms for submodular maximization break down.

Our work is then motivated by trying to better understand the multi-dimensional nature of this problem, and provide a more principled algorithmic approach.

1.1 Our Results

We use techniques from convex duality to re-interpret, simplify, and improve the existing results. First, we obtain the first approximation for non-separable supermodular cost functions. (We omit some mild regularity conditions for brevity; see Section 3 for full details.)

▶ **Theorem 1** (Unconstrained & Supermodular). *For the unconstrained problem with* supermodular *convex cost functions g, we give an $O(d)$-competitive randomized algorithm in the random-order model.*

This result generalizes the $O(d)$-approximation of Barman et al. [5] to the non-separable case. The factor d seems unavoidable, since our problem inherits the (offline) $\Omega(d^{1-\varepsilon})$ hardness of the d-dimensional knapsack, assuming $NP \neq ZPP$ [7].

Next, we consider the constrained case. For simplicity, we focus on the most interesting case where $\mathscr{F}$ is a matroid constraint; more general results can be obtained from the results and techniques in Section 5.

▶ **Theorem 2** (Constrained & Separable). *For the constrained problem with $\mathscr{F}$ being a matroid constraint, and the cost function g being* separable*, we get an $O(d^2 \log\log \mathit{rank})$-competitive randomized algorithm in the random-order model.*

This improves by a factor of d^3 the $O(d^5 \log\log \mathrm{rank})$-approximation given by [5]. Finally, we give a general reduction that takes an algorithm for *separable* functions and produces an algorithm for *supermodular* functions, both with respect to a matroid constraint, implying:

▶ **Theorem 3** (Constrained & Supermodular). *For the constrained problem with $\mathscr{F}$ being a matroid constraint, and the cost function g being* supermodular*, we get an $O(d^3 \log\log \mathit{rank})$-competitive randomized algorithm in the random-order model.*

On conceptual contributions are in bringing techniques from convex duality to obtain, in a principled way, *threshold-based* algorithms for non-linear secretary problems. Since this is a classical and heavily used algorithmic strategy for secretary problems [13, 4, 18, 2, 20] we hope that the perspectives used here will find use in other contexts.

1.2 Other Related Work

There is a vast literature on secretary problems [13]. Closest to our setting, Agrawal and Devanur study an online convex optimization problem in the random order model, and give a powerful result showing strong regret bounds in this setting [1]. They extend this result to give algorithms for online packing LPs with "large" right-hand sides. However, it is unclear how to use their algorithm to obtain results in our setting. Other algorithms solving packing LPs with large right-hand sides appear in [2, 8, 20, 17, 14, 10].

Feldman and Zenklusen [12] show how to transform any algorithm for (linear) matroid secretary into one for *submodular* matroid secretary. They give an $O(\log\log \mathrm{rank})$-algorithm for the latter, based on results of [19, 11]. All these algorithms critically assume the submodular function is non-negative everywhere, which is not the case for us, since picking too large a set may cause the profit function to go negative. Indeed, one technical contribution is a procedure for making the profit function non-negative while preserving submodularity (Section 4.1), which allows us to use these results as part of our solution.

1.3 Structure of the paper

Section 3 develops the convex duality perspective used in the paper for the offline version of the unconstrained case, hopefully in an manner accessible to non-experts. Section 4 gives the small changes required to extend this to the constrained case. Section 5 shows how transform these into online algorithms. Section 6 shows how to convert an algorithm for separable functions into one for supermodular functions, both subject to matroid constraints. To improve the presentation, we make throughout mild assumptions, which are discharged in the full version of the paper.

2 Preliminaries

Elements from a universe U of size n are presented in random order. Each element e has value $v(e) \in \mathbb{R}_+$ and size $s(e) \in \mathbb{R}^d_+$. We are given a convex cost function $g : \mathbb{R}^d_+ \to \mathbb{R}_+$. On seeing each element we must either accept or discard it. A downwards-closed collection $\mathscr{F} \subseteq 2^U$ of feasible sets is also given. When $\mathscr{F} = 2^U$, we call it the *unconstrained* problem. The goal is to pick a subset $A \in \mathscr{F}$ to maximize the *profit*

$$\pi(A) := \sum_{e \in A} v(e) - g\big(\sum_{e \in A} s(e)\big). \tag{2.2}$$

We often use vectors in $\{0,1\}^n$ to denote subsets of U; χ_A denotes the indicator vector for set A. Hence, $\mathscr{F} \subseteq \{0,1\}^n$ is a down-ideal on the Boolean lattice, and we can succinctly write our problem as

$$\max_{x \in \mathscr{F}} \quad \pi(x) := \langle v, x \rangle - g(Sx), \tag{2.3}$$

where columns of $S \in \mathbb{R}^{d \times n}$ are the item sizes. Let opt denote the optimal value. For a subset $A \subseteq U$, $v(A)$ and $s(A)$ denote $\sum_{e \in A} v(e) = \langle v, \chi_A \rangle$ and $\sum_{e \in A} s(e) = S\chi_A$ respectively.

▶ **Definition 4** (Exceptional). Item $e \in U$ is *exceptional* if $\arg\max_{\theta \in [0,1]} \{\theta\, v(e) - g(\theta\, s(e))\} \in (0,1)$.

▶ **Definition 5** (Marginal Function). Given $g : \mathbb{R}^d \to \mathbb{R}$, define the i^{th} *marginal function* $g_i : \mathbb{R} \to \mathbb{R}$ as $g_i(x) := g(x\mathbf{e}_i)$, where $\mathbf{e}_i$ is the i^{th} standard unit vector.

▶ **Definition 6** (Convex Dual). For any function $g : \mathbb{R}^d \to \mathbb{R}$, its *convex dual* is the function $g^\star : \mathbb{R}^d \to \mathbb{R}$ given by $g^\star(y) := \sup_x \big[\langle y, x \rangle - g(x)\big]$.

2.1 Supermodular Functions

While supermodular functions defined over the Boolean lattice are widely considered, one can define supermodularity for all real-valued functions.

▶ **Definition 7** (Supermodular). Let $X \subseteq \mathbb{R}^d$ be a lattice. A function $f : X \to \mathbb{R}$ is *supermodular* if for all $x, y \in X$, $f(x) + f(y) \le f(x \wedge y) + f(x \vee y)$, where $x \wedge y$ and $x \vee y$ are the component-wise minimum and maximum operations.

This corresponds to the usual definition of (discrete) supermodularity when $X = \{0,1\}^d$. For proof of the lemma below and other equivalent definitions, see, e.g., [21].

▶ **Lemma 8** (Supermodularity and Gradients). *A convex function $f : \mathbb{R}^d_+ \to \mathbb{R}$ is supermodular if and only if any of the following are true.*

- *∇f is increasing in each coordinate, if f is differentiable.*
- *$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \ge 0$ for all i, j, if f is twice-differentiable.*

▶ **Lemma 9** (Superadditivity). *If $f : \mathbb{R}^d_+ \to \mathbb{R}$ is differentiable, convex, and supermodular, then for $x, x', y \in \mathbb{R}^d_+$ such that $x' \le x$, $f(x' + y) - f(x') \le f(x + y) - f(x)$. In particular, if $f(0) = 0$, setting $x' = 0$ gives $f(x) + f(y) \le f(x + y)$.*

▶ **Corollary 10** (Subadditivity of profit). *The profit function π is subadditive.*

The next fact shows that the cost g is also supermodular when seen in a discrete way.

▶ **Fact 11** (Continuous vs. Discrete Supermodularity). *Given a convex supermodular function $g : \mathbb{R}^d \to \mathbb{R}$ and n items with sizes $s_1, \ldots, s_n \in \mathbb{R}^d_+$, define the function $h : \{0,1\}^n \to \mathbb{R}$ as $h(v) = g(\sum_i s_i v_i) = g(Sv)$. Then $h(\cdot)$ is a (discrete) supermodular function.*

3 The Offline Unconstrained Problem

We first present an offline algorithm for supermodular functions in the **unconstrained** case (where $\mathscr{F} = \{0,1\}^n$). We focus on the main techniques and defer some technicalities and all computational aspects for now. Just for this section, we assume item sizes are "infinitesimal". We make the following assumptions on the cost function g and the elements.

▶ **Assumption 12.** *We assume that cost function g is non-negative, strictly convex, closed, and differentiable. We assume $g(0) = 0$, g is supermodular, and that gradients of g go to ∞ along every positive direction. We assume elements are in general positionand that there are no exceptional items. We also assume that every individual item has profit at most $M := \mathsf{opt}/\eta d$ for $\eta \geq 10^4$ (see the full version to remove these assumptions).*

Classifiers. The offline algorithm will be based on *linear classifiers*, where a set of weights is used to aggregates the multidimensional size of an item into a scalar, and the algorithm picks all items that have high-enough value/aggregated-size ratio.

▶ **Definition 13** (Classifiers and Occupancy)**.** Given a vector $\lambda \in \mathbb{R}^d_+$ (a "classifier"), define the set of items picked by λ as $U_\lambda := \{e \in U \mid v(e) \geq \langle \lambda, s(e)\rangle\}$. Let $\mathsf{occ}_\lambda := \sum_{e:v(e)\geq\langle\lambda,s(e)\rangle} s(e)$ denote the multidimensional occupancy induced by choosing items in U_λ.

To understand the importance of classifier-based solutions it is instructive to consider the problem with single-dimensional size. A little thought shows that an optimal solution is to pick items in decreasing order of value density $v(e)/s(e)$. Adding these items causes the total occupancy – and hence the incurred cost – to increase, so we stop when the value density of the current item becomes smaller than the derivative of the cost function at the current utilization. That is, we find a density threshold λ such that $g'(\textit{total size of items having } v(e) \geq \lambda\, s(e)) \approx \lambda$, and take all these high-density items. Thus, the optimal solution is one based on the classifier λ.

To see that this holds in the multi-dimensional case, express g in terms of linearizations

$$g(z) = \max_{\lambda \in \mathbb{R}^d_+} (\langle \lambda, z\rangle - g^\star(\lambda)), \tag{3.4}$$

where $g^\star$ is its Fenchel dual. (Note we are maximizing over positive, but this is WLOG.) Then our unconstrained problem (2.2) becomes a minimax problem:

$$\max_{x\in\{0,1\}^n} \min_{\lambda\in\mathbb{R}^d_+} \left[\langle v, x\rangle - \Big(\langle \lambda, Sx\rangle - g^\star(\lambda)\Big)\right].$$

Consider an optimal pair (x^*, λ^*); i.e., a pair that is a saddle-point solution, so neither x^* nor λ^* can be improved keeping the other one fixed. This saddle-point optimality implies:

(a) Since $\lambda^* = \operatorname{argmax}_{\lambda\in\mathbb{R}^d_+} (\langle \lambda, Sx^*\rangle - g^\star(\lambda))$, it is the right linearization of g at Sx^* and thus $\lambda^* = \nabla g(Sx^*)$ (see [15, Theorem E.1.4.1] and [15, Corollary E.1.3.6]).

(b) x^* is such that $x^*_i = 1$ if $v_i > \langle \lambda^*, S^i\rangle$ and $x^*_i = 0$ if $v_i < \langle \lambda^*, S^i\rangle$, with S^i being the i^{th} column of S and the size of the i^{th} item.

From part (b) we see the optimal solution x^* is essentially the one picked by the classifier λ^* (ignoring coordinates with "0 marginal value" $v_i = \langle \lambda^*, S^i\rangle$). The converse also holds.

▶ **Claim 14.** *For a classifier $\lambda \in \mathbb{R}^d_+$, let x be the items picked by it. If we have $\lambda = \nabla g(Sx) \stackrel{def}{=} \nabla g(\mathsf{occ}_\lambda)$, then x is an optimal solution.*

Proof. For any solution x',

$$\pi(x') = \langle v, x' \rangle - g(Sx') \le \langle v, x' \rangle - \langle \lambda, Sx' \rangle + g^\star(\lambda)$$
$$\le \langle v, x \rangle - \langle \lambda, Sx \rangle + g^\star(\lambda) \overset{(\lambda = \nabla g(Sx))}{=} \langle v, x \rangle - g(Sx) = \pi(x),$$

where the second inequality holds since, by definition, x maximizes $\langle v, x \rangle - \langle \lambda, Sx \rangle$. ◀

Restricting the Set of Classifiers. The existence of such good classifiers is not enough, since we need to *find* them online. This is difficult not only because of d degrees of freedom and no control over the magnitude of the values/sizes (to be exploited in concentration inequalities), but also because picking too few or too many items could lead to low profits.

So we restrict the set of candidate classifiers to be a *monotone*[4] *1-dimensional* curve $\mathcal{C} \subseteq \mathbb{R}^d_+$, satisfying additional properties given below. The main motivation is that it imposes a total ordering on the set of items picked by the classifiers: given $\lambda \le \mu$ on such a curve $\mathcal{C}$, the sets of items picked satisfy the inclusion $U_\lambda \supseteq U_\mu$. This allows us to select a "minimally good" classifier in $\mathcal{C}$ in a robust way, avoiding classifiers that select too many items.

To design the curve $\mathcal{C}$ so it contains a classifier with profit $\approx \frac{\mathsf{opt}}{d}$, we relax the condition $\nabla g(\mathsf{occ}_\lambda) = \lambda$ from Claim 14 (too much to ask) and require the existence of $\lambda \in \mathcal{C}$ satisfying:

(P1) (don't pick too many items) $\nabla g(\mathsf{occ}_\lambda) \le \lambda$.
(P2) (partial gradient equality) There is a coordinate i^* where $(\nabla g(\mathsf{occ}_\lambda))_{i^*} = \lambda_{i^*}$.
(P3) (balanced curve) $g_i^\star(\lambda_i) = g_j^\star(\lambda_j) \quad \forall i, j \in [d]$.

Property (P1) enforces half of the equality in Claim 14, and (P2) guarantees that equality holds for *some* coordinate. Now for property (P3). Since $\lambda \ne \nabla g(\mathsf{occ}_\lambda)$ the optimality proof of Claim 14 does not go though, since $g(\mathsf{occ}_\lambda) \ne \langle \lambda, \mathsf{occ}_\lambda \rangle - g^\star(\lambda)$. As we prove later, the difference between these terms can be at most $g^\star(\lambda) \le \sum_i g_i^\star(\lambda_i)$. Property (P3) is used to control this sum, by charging it to the coordinate i^* where we know we have "the right linearization" (by property (P2)). Reinterpreting the construction of [5] in our setting, we then define $\mathcal{C}$ as any monotone curve where every $\lambda \in \mathcal{C}$ satisfies (P3).

▶ **Lemma 15.** *The curve $\mathcal{C}$ exists and contains a λ satisfying properties (P1)-(P3).*

Proof. We first show existence, that is, the set $\{\lambda \in \mathbb{R}^d_+ \mid g_i^\star(\lambda_i) = g_j^\star(\lambda_j) \ \ \forall i, j\}$ contains a monotone curve. Notice that this set is the union of the box $\{\lambda \in \mathbb{R}^d_+ \mid g_i^\star(\lambda_i) = 0 \ \ \forall i\} = \prod_i [0, g_i'(0)]$ (range of slopes where we can swivel around $g_i(0) = 0$) and a monotone curve $\{\lambda(\tau) \mid \tau > 0\}$, where $\lambda(\tau)$ is the unique vector satisfying $g_i^\star(\lambda_i(\tau)) = \tau$; uniqueness follows from the fact $g_i^\star$ stays at value zero in the interval $[0, g_i'(0)]$, but after that is strictly increasing due to its convexity, and monotonicity of this curve also follows from monotonicity of the $g_i^\star$'s. Thus, $\mathcal{C}$ is this curve plus any monotone curve extending it to the origin.

To see that $\mathcal{C}$ satisfies properties (P1) and (P2), we note that since the $g_i^\star$'s are increasing and not identically 0, $\mathcal{C}$ is unbounded in all coordinates. Thus, a sufficiently large $\lambda \in \mathcal{C}$ satisfies (P1), and we can start with such λ and move down the curve (decreasing in each coordinate) until we obtain $\lambda' \in \mathcal{C}$ with $\lambda' = \nabla g(\mathsf{occ}_{\lambda'})$, since the g has increasing gradients. (The equality in this final step uses the assumption that item sizes are infinitesimal, which we made for simplicity in this section). ◀

Making the above discussion formal, we show that $\mathcal{C}$ has a high-value classifier. Recall that U_λ is the set of items picked by λ (Definition 13).

[4] A curve $\mathcal{C}$ is *monotone* if for every pair $\lambda, \lambda' \in \mathcal{C}$, one is coordinate-wise smaller than the other.

▶ **Theorem 16.** *Given Assumption 12, let λ^* be a classifier in $\mathcal{C}$ satisfying properties (P1)-(P3). Then for all $x' \in [0,1]^n$ we have $\pi(U_{\lambda^*}) \geq \frac{1}{d+1} \cdot \pi(x')$.*

Proof. Let $x^* = \chi_{U^*_\lambda}$ be the solution picked by the classifier λ^*, and note that $\mathsf{occ}_{\lambda^*} = Sx^*$. Let $L(y,\mu) := \langle v, y\rangle - [\langle \mu, Sy\rangle - g^\star(\mu)]$ be the linearization of $\pi(y)$ at some slope μ. From (3.4) we know $g(y) \geq L(y,\mu)$ for all $\mu \geq 0$. Since x^* is optimal for the linearization $L(y,\lambda^*)$ (because $x^*_i = 1$ iff $v_i - \langle \lambda^*, S^i\rangle \geq 0$), we have

$$L(x^*,\lambda^*) \geq L(x',\lambda^*) \geq \pi(x') \qquad \text{for all } x' \in [0,1]^n. \tag{3.5}$$

Now we relate the true profit $\pi(x^*)$ to this linearized value. Observe that

$$\begin{aligned}\pi(x^*) = \langle v, x^*\rangle - g(Sx^*) &= \langle v, x^*\rangle - [\langle \nabla g(Sx^*), Sx^*\rangle - g^\star(\nabla g(Sx^*))] \\ &\geq \underbrace{\langle v, x^*\rangle - \langle \lambda^*, Sx^*\rangle}_{\geq 0} + \underbrace{g^\star(\nabla g(Sx^*))}_{\geq 0},\end{aligned} \tag{3.6}$$

where the inequality uses that $\lambda^* \geq \nabla g(Sx^*)$ by property (P1) and $Sx^* \geq 0$. The first term is non-negative because we only pick items for which $v_i - \langle \lambda, S^i\rangle \geq 0$. The second term is non-negative because $g(0) = 0$. We can now prove three lemmas that imply the theorem.

▶ **Lemma 17.** *For any $x' \in [0,1]^n$, $\pi(x^*) \geq L(x^*,\lambda^*) - g^\star(\lambda^*) \geq \pi(x') - g^\star(\lambda^*)$.*

Proof. Drop the second term from (3.6), then use the definition of $L(\cdot,\cdot)$ and (3.5). ◀

▶ **Lemma 18.** $g^\star(\lambda^*) \leq d \cdot g^\star_{i^*}(\lambda^*_{i^*})$.

Proof. Using the superadditivity of g, one can show $g^\star(\lambda^*) \leq \sum_i g^\star_i(\lambda^*_i)$. Now from property (P3) of the classifier λ^*, all the terms in the sum are equal. ◀

▶ **Lemma 19.** $\pi(x^*) \geq g^\star_{i^*}(\lambda^*_{i^*})$.

Proof. We claim that $g^\star(\nabla g(Sx^*)) \geq g^\star_{i^*}(\lambda^*_{i^*})$; plugging this into (3.6) proves the lemma. For the claim, define $\lambda' = \nabla g(Sx^*)$. By Property (P2), $\lambda'_{i^*} = \lambda^*_{i^*}$, so we want to show $g^\star(\lambda') \geq g^\star_{i^*}(\lambda'_{i^*}) = g^\star(\lambda'_{i^*}\mathbf{e}_{i^*})$. This follows because $g^\star$ is monotone. ◀

This completes the proof of Theorem 16. ◀

4 The Offline Constrained Case

Having built up tools and intuition in the unconstrained case, we turn to the case where there is a downwards-closed constraint $\mathscr{F} \subseteq \{0,1\}^n$, and the goal is to maximize the profit subject to $x \in \mathscr{F}$. We again work with Assumption 12, but do not assume anything about items sizes. We discuss computational aspects at the end of this section.

The general idea is again to use classifiers $\lambda \in \mathbb{R}^d_+$, and only consider items in U_λ, namely those with "high-enough" value $v_i \geq \langle \lambda, S^i\rangle$. However, because of the constraints $\mathscr{F}$ we may no longer be able to pick all these items. Thus, we need to consider the most profitable solution from $\mathscr{F}$ in this filtered feasible set U_λ (whose quality is less clear how to analyze).

Again we restrict to the 1-dimensional curve $\mathcal{C}$ defined in the previous section; however, it only satisfies slightly modified versions of properties (P1)-(P2), since we do not assume the item sizes to be infinitesimal anymore. To make this precise, define the "open" set $U^\circ_\lambda := \{e \in U \mid v(e) > \langle \lambda, s(e)\rangle\}$; note the strict inequality. Under the assumption of items being in general position, there is at most one "threshold" item with $v_i = \langle \lambda, S^i\rangle$, i.e., $|U_\lambda \setminus U^\circ_\lambda| \leq 1$. Now a "good" classifier is one that satisfies the following:

(P1') For all binary x with $\text{support}(x) \subseteq U^\circ_\lambda$ and $x \in \mathscr{F}$, $\nabla g(Sx) \le \lambda$.

(P2') There exists a binary x^{occ} with $\text{support}(x^{occ}) \subseteq U_\lambda$ and $x^{occ} \in \mathscr{F}$, and index i^* such that $(\nabla g(Sx^{occ}))_{i^*} \ge \lambda_{i^*}$. (Note that if $\text{support}(x^{occ}) \subseteq U^\circ_\lambda$, then by property (P1') the above inequality holds at equality; else x^{occ} contains the unique element in $U_\lambda \setminus U^\circ_\lambda$.)

(P3') This is the same as before: $g^\star_i(\lambda_i) = g^\star_j(\lambda_j) \quad \forall i, j \in [d]$.

The arguments of Lemma 15 show the following.

▶ **Lemma 20.** *Given Assumption 12, the curve $\mathcal{C}$ defined in the previous section contains a λ satisfying properties (P1')-(P3').*

Next, we show that for a good classifier $\lambda \in \mathcal{C}$, the maximum profit solution from $\mathscr{F}$ contained within U°_λ essentially gives an $O(1/d)$-approximation.

▶ **Theorem 21** (Offline Approach)**.** *Suppose Assumption 12 holds. Let λ^* be a classifier in $\mathcal{C}$ satisfying properties (P1')–(P3'). Then the better of the two solutions: (a) the maximum profit solution in $\mathscr{F}$ containing elements only from $U^\circ_{\lambda^*}$, and (b) the optimal single element in U_{λ^*}, has profit at least $\pi(x')/(2d+1)$ for any vector $x' \in \text{Conv}(\mathscr{F}) \subseteq [0,1]^n$.*

Proof. The idea is to follow the development in Theorem 16. There same solution x^* satisfied the value lower bounds of Lemmas 17 and 19; to satisfy the first lemma, we needed the solution to be optimal for the linearization of π using "slope" λ^*; to satisfy the second, we needed to satisfy (P2). Here, we construct two solutions in $\mathscr{F}$ intersect U_{λ^*} to satisfy these lemmas separately:

$x^{lin} := \text{argmax}\{\langle v, y\rangle - \langle \lambda^*, Sy\rangle \mid y \subseteq U^\circ_{\lambda^*}, y \in \mathscr{F}\}$

$x^{occ} :=$ the solution promised by property (P2').

Since property (P1') and (P3') holds for x^{lin}, Lemmas 17 and 18 hold essentially unchanged, and thus for any vector $x' \in \text{Conv}(\mathscr{F})$ we have

$$\pi(x^{lin}) \ge \pi(x') - d \cdot g^\star_{i^*}(\lambda^*_{i^*}). \tag{4.7}$$

The solution x^{occ} may not belong to the set $U^\circ_{\lambda^*}$, since it may contain the threshold item $e^\circ = \langle \lambda^*, s(e^\circ)\rangle$, if it exists (let $x^\circ = \chi_{\{e^\circ\}}$ be its characteristic vector, all 0's vector if does not exists). Let $x^{rest} = x^{occ} - x^\circ$.

▶ **Lemma 22.** *These solutions satisfy $\pi(x^{rest}) + \pi(x^\circ) \ge g^\star_{i^*}(\lambda^*_{i^*})$.*

Proof. Property (P1') gives $\nabla g(Sx^{rest}) \le \lambda^*$, and Property (P2') implies $\nabla g(S(x^{rest}+x^\circ)) = \nabla g(Sx^{occ})$ is at least λ^* at some coordinate i^*. Since g is convex and differentiable, the gradients are continuous [15, Remark D.6.2.6], so there is $\delta \in [0,1]$ where the vector $\widehat{x} := x^{rest} + \delta x^\circ$ satisfies $\nabla g(S\widehat{x}) \le \lambda^*$ and $\nabla g(S\widehat{x})_{i^*} = \lambda^*_{i^*}$ for some coordinate i^*. Due to these properties, the proof of Lemma 19 holds for $\widehat{x}$ and shows $\pi(\widehat{x}) \ge g^\star_{i^*}(\lambda^*_{i^*})$.

The assumption of no exceptional items gives $\pi(\delta x^\circ) \le \pi(x^\circ)$. From subadditivity of profit π, $g^\star_{i^*}(\lambda^*_{i^*}) \le \pi(\widehat{x}) \le \pi(x^{rest}) + \pi(\delta x^\circ) \le \pi(x^{rest}) + \pi(x^\circ)$. This concludes the proof. ◀

Combining Lemma 22 with inequality (4.7) we have $\pi(x') \le \pi(x^{lin}) + d\,\pi(x^{rest}) + d\,\pi(x^\circ)$ for any $x' \in \mathscr{F}$. Since x^{lin}, x^{rest} are feasible for (a) in the theorem statement, and x° is feasible for (b), the best of them gives a $(2d+1)$-approximation. This proves Theorem 21. ◀

Picking the most profitable singleton is trivial offline, and well-approximable online by the secretary algorithm [13]. Moreover, we need to approximately optimize the *submodular* function π (Fact 11) over $\mathscr{F}|_{U^\circ_{\lambda^*}}$ (i.e., the sets in $\mathscr{F}$ with only elements of $U^\circ_{\lambda^*}$). For several constraint structures (e.g., matroids, p-systems), there are known algorithms for approximately optimizing *non-negative* (and sometimes also monotone) submodular functions. Unfortunately, our profit function π may take negative values, so we cannot directly use these algorithms. Simply considering the truncated function $\max\{\pi(z), 0\}$ does not work because it may be non-submodular. In the next section, when g is *separable*, we introduce a way of making our profit function non-negative everywhere, while maintaining submodularity and preserving the values at the region of interest $\mathscr{F}|_{U^\circ_{\lambda^*}}$.

4.1 Making the Profit Function π Non-negative

We first show that π already satisfies the desired properties over the sets in $\mathscr{F}|_{U^\circ_{\lambda^*}}$.

▶ **Lemma 23.** *The profit function π is non-negative monotone over $\mathscr{F}|_{U^\circ_{\lambda^*}}$.*

Proof. Since $\pi(\emptyset) = 0$ it suffices to show monotonicity. Consider $x \in \mathscr{F}|_{U^\circ_{\lambda^*}}$ and let χ_e be the indicator os an item in x. Comparing the costs with and without e we have

$$g(Sx) \overset{\text{(convexity)}}{\leq} g(S(x-\chi_e)) + \langle \nabla g(Sx), S\chi_e \rangle \overset{\text{(Property (P1'))}}{\leq} g(S(x-\chi_e)) + \langle \lambda^*, s(e) \rangle.$$

Since $x \in U^\circ_{\lambda^*}$, we have $v(e) > \langle \lambda^*, s(e) \rangle$ and thus $\pi(x) > \pi(x - \chi_e)$, i.e., monotonicity. ◀

However, to run algorithms that approximately optimize π over $\mathscr{F}|_{U^\circ_{\lambda^*}}$ in a black-box fashion, non-negativity over the feasible sets $\mathscr{F}|_{U^\circ_{\lambda^*}}$ is not enough, even if the algorithm only probes π over these sets, since their *proof of correctness* may require this property outside of feasible sets. Thus, we need to modify π to ensure non-negativity outside of $\mathscr{F}|_{U^\circ_{\lambda^*}}$.

For that, the idea is to truncate the gradient of the cost g so $\nabla g(Sx)$ becomes at most λ^* for all subsets $x \subseteq U^\circ_{\lambda^*}$ (i.e., so Property (P1') holds for all subsets); this was the crucial element for the monotonicity (and hence non-negativity) proof above. Notice that since Property (P1') guarantees already $\nabla g(Sx) \leq \lambda^*$ for all $x \in \mathscr{F}|_{U^\circ_{\lambda^*}}$, this does not change the value of π over these points. The proof of the lemma is deferred to the full version.

▶ **Lemma 24.** *If g is separable, there is a submodular function π^+ satisfying the following:*

(i) *π^+ is non-negative and monotone over all subsets of $U^\circ_{\lambda^*}$, and*

(ii) *$\pi^+(x) = \pi(x)$ for every $x \in \mathscr{F}|_{U^\circ_{\lambda^*}}$.*

4.2 The Offline Algorithm: Wrap-up

Using this non-negativization procedure, we get an $O(d)$-approximation *offline* algorithm for constrained profit maximization for *separable* cost functions g; this is an offline analog of Theorem 2. For the unconstrained case, Lemma 23 implies that the profit function π it itself monotone, so we get an $O(d)$-approximation offline algorithm for the *supermodular* case. In the next section we show how to convert these algorithms into online algorithms.

One issue we have not discussed is the computational cost of finding λ^* satisfying (P1')–(P3'). In the full version of the paper, we show that for any $\varepsilon > 0$ we can efficiently find a λ^* satisfying (P1'), (P2'), and a slightly weaker condition: $|g^\star_i(\lambda^*_i) - g^\star_j(\lambda^*_j)| \leq 2\varepsilon$ for all $i, j \in [d]$. Using this condition in Theorem 21 means we get a profit of at least $\frac{\mathsf{opt}-2d\varepsilon}{2d+1} \geq [\mathsf{opt}/(2d+1)] - \varepsilon$; the running time depends on $\log \varepsilon^{-1}$ so we can make this loss negligible.

5 The Online Algorithm

In the previous sections we were working offline: in particular, in computing the "good" classifier $\lambda \in \mathcal{C}$, we assumed knowledge of the entire element set. We now present the online framework for the setting where elements come in random order. Recall the definition of the curve $\mathcal{C}$ from §3, and the fact that there is a total order among all $\lambda \in \mathcal{C}$. Recall that for simplicity we restrict the constraints $\mathscr{F}$ to be matroid constraints.

For a subset of elements $A \subseteq U$, let $\mathsf{opt}(A)$ and $\mathsf{fopt}(A)$ denote the integer and fractional optimal profit for $\mathscr{F}|_A$, the feasible solutions restricted to elements in A. Note that in the fractional case this means the best solution in the convex hull $\mathrm{Conv}(\mathscr{F}|_A)$. Clearly, $\mathsf{fopt}(A) \geq \mathsf{opt}(A)$. We use opt and fopt to denote $\mathsf{opt}(U)$ and $\mathsf{fopt}(U)$ for the entire instance U.

Again we work under Assumption 12. We will also make use of any algorithm for maximizing submodular functions over $\mathscr{F}$ in the random-order model satisfying the following.

▶ **Assumption 25.** *Algorithm SubmodMS takes a nonnegative monotone submodular function f with $f(\emptyset) = 0$, and a number N. When run on a sequence X of N elements presented in random order, it returns a (random) subset $X_{alg} \in \mathscr{F}$ with expected value $\mathbb{E}[f(X_{alg})] \geq \frac{1}{\alpha} \max_{X' \in \mathscr{F}} f(X)$. Moreover, the it only evaluates the function f on **feasible** sets.*

Our algorithm is very simple:

Algorithm 5.1 Online Algorithm for Profit Maximization

1: $L \leftarrow$ first Binomial$(n, 1/2)$ items.
2: $\mu \leftarrow$ largest vector on curve $\mathcal{C}$ s.t. $\mathsf{fopt}(L_\mu) \geq \frac{1}{12d}\,\mathsf{fopt}(L)$.
3: $R \leftarrow$ remaining instance, namely the last $n - |L|$ items.
4: $R^\circ_\mu \leftarrow \{e \in R \mid v(e) > \langle \mu, s(e) \rangle\}$ be the (strictly) "filtered" remaining instance.
5: **Un-constrained:** Select items in R°_μ not decreasing the current value of the solution.
Constrained: Run algorithm SubmodMS on R°_μ using profit function π, selecting items according to this algorithm, but do not add items that decrease the current value of the solution.

Note that L_μ denotes the set of items in the sample L picked by μ (Definition 13). In Step 2, we can use the Ellipsoid method to find fopt within negligible error. Moreover, we must do this for several sets L_μ and pick the largest one on $\mathcal{C}$ using a binary-search procedure. We defer the technical details to the full version of the paper.

5.1 Analysis

To analyze the algorithm, we need to show that the classifier μ learned in Step 2 is large enough that we do not waste space with useless items, but low enough that we admit enough useful items. For that we need the following concentration bound.

▶ **Lemma 26.** *Consider a submodular function $f : 2^{\mathcal{U}} \to \mathbb{R}$. Consider a set $Y \subseteq \mathcal{U}$ such that f is non-negative over all of its subsets, and that for some M:*

$$\text{For all } Y' \subseteq Y \text{ and element } e \in Y', \quad |f(Y') - f(Y' - e)| \leq M. \tag{5.8}$$

Let $\mathbf{Y}$ be the random subset obtained by picking each element from Y independently with some probability (possibly different for each item). Then $\Pr(|f(\mathbf{Y}) - \mathbb{E}[f(\mathbf{Y})]| \geq t) \leq \frac{2M\,\mathbb{E}[f(\mathbf{Y})]}{t^2}$.

We then also need π to satisfy (5.8) on the optimal solutions of any given sub-instance. For a vector $y \in \mathbb{R}^n$ and subset $A \subseteq U$, let y_A be the same as y on A, and zero outside A.

▶ **Claim 27.** *Consider any $U' \subseteq U$, and let y be an optimal fractional solution on $\mathscr{F}|_{U'}$ (so $\pi(y) = \mathsf{fopt}(U')$). Then for any $B \subseteq A \subseteq U'$ with $|A \setminus B| = 1$, we have $|\pi(y_A) - \pi(y_B)| \leq M$, where M is an upper bound on the profit from any single item.*

From Section 4, recall $\lambda^* \in \mathbb{R}^d_+$ is a classifier that satisfies properties (P1')–(P3').

▶ **Lemma 28.** *Given Assumption 12, the classifier μ of Line 2 of Algorithm 5.1 satisfies:*

(a) *(Not too small) $\mu \geq \lambda^*$, with probability at least $19/20$.*

(b) *(Not too big) $\mathsf{fopt}(U_\mu) \geq \frac{\mathsf{fopt}}{48d}$ with probability at least $1 - 1/20d \geq 19/20$.*

Proof sketch. For the first part, we show that the classifier λ^* satisfies the properties needed in Line 2 with probability $1 - 1/20$; since μ is the largest such vector, we get $\mu \geq \lambda^*$. Using Theorem 21 and the assumption that no item has large profit, we have $\mathsf{fopt}(U_{\lambda^*}) \geq \frac{\mathsf{fopt}}{3d}$. Moreover, the sample obtains at least half of this profit in expectation, i.e., $\mathbb{E}\,\mathsf{fopt}(L_{\lambda^*}) \geq \frac{\mathsf{fopt}}{3d}$. Then using Lemma 26 with the Lipschitz property of Claim 27 and the no-high-profit-item assumption, we have $\mathsf{fopt}(L_{\lambda^*}) \geq \frac{\mathsf{fopt}}{12d} \geq \frac{\mathsf{fopt}(L)}{12d}$ with probability at least $19/20$. Thus, with this probability λ^* satisfies the properties needed in Line 2 of the algorithm, as desired.

For the part (b) of the lemma, notice that for each scenario $\mathsf{fopt}(U_\mu) \geq \mathsf{fopt}(L_\mu)$, since feasible solutions for the sample are feasible for the whole instance. Next, by definition of μ, $\mathsf{fopt}(L_\mu) \geq \frac{\mathsf{fopt}(L)}{12d}$. Finally, if x is the fractional optimal solution on U with $\pi(x) = \mathsf{fopt}$, then $\mathbb{E}[\pi(x_L)] \geq \mathsf{fopt}/2$, since g is superadditive. Again using Lemma 26, the profit $\pi(x_L)$ is at least $\frac{\mathsf{fopt}}{4}$ with probability at least $(1 - 1/20d)$. Of course, $\mathsf{fopt}(L) \geq \pi(x_L)$. Chaining these inequalities, $\mathsf{fopt}(U_\mu) \geq \frac{\mathsf{fopt}}{48d}$ with this probability. ◀

In view of Theorem 21, we show the filtered out-of-sample instance R°_μ behaves like $U^\circ_{\lambda^*}$.

▶ **Lemma 29.** *The filtered out-of-sample instance R°_μ satisfies the following w.p. $19/20$:*

(a) *For all $e \in R^\circ_\mu$, $v(e) \geq \langle \lambda^*, s(e) \rangle$.*

(b) *For all x with support$(x) \subseteq R^\circ_\mu$ such that $x \in \mathscr{F}$, $\nabla g(Sx) \leq \lambda^*$.*

(c) $\mathsf{fopt}(R^\circ_\mu) \geq \frac{\mathsf{fopt}}{200d}$.

Proof. By Lemma 28(a), threshold $\mu \geq \lambda^*$ with probability $19/20$. When that happens, $U^\circ_\mu \subseteq U^\circ_{\lambda^*}$. Since the first two properties hold for $U^\circ_{\lambda^*}$, they also hold for U°_μ, and by downward-closedness, also for R°_μ.

For the third part, let λ^+ be the largest threshold in $\mathcal{C}$ such that $\mathsf{fopt}(U_{\lambda^+}) \geq \frac{\mathsf{fopt}}{48d}$. From Lemma 28(b), with good probability we have $\mu \leq \lambda^+$. Since μ is a smaller threshold, the instance U_{λ^+} is contained in the instance U_μ, which implies that for every scenario $\mathsf{fopt}(R_\mu) \geq \mathsf{fopt}(R_{\lambda^+})$. Next we show that that with good probability $\mathsf{fopt}(R_{\lambda^+}) \geq \frac{\mathsf{fopt}}{200d}$, and hence get the same lower bound for $\mathsf{fopt}(R_\mu)$. If y is the optimal fractional solution for U_{λ^+}, then y_R is feasible for R_{λ^+} with $\mathbb{E}[\pi(y_R)] = \frac{1}{2}\mathsf{fopt}(U_{\lambda^+}) \geq \frac{\mathsf{fopt}}{96d}$. Moreover, using the concentration bound again, we get that $\pi(y_R) \geq \frac{\mathsf{fopt}}{192d}$ with probability at least $19/20$. Finally, by the assumption of general position, there is at most one item in $R_\mu \setminus R^\circ_\mu$. Dropping this item from the solution y to get y° reduces the value by at most $M = \frac{\mathsf{fopt}}{10^4 d}$; here we use subadditivty of the profit, and that there are no exceptional items. Hence, with probability at least $19/20$: $\mathsf{fopt}(R^\circ_\mu) \geq \mathsf{fopt}(R^\circ_{\lambda^+}) \geq \pi(y^\circ_R) \geq \frac{\mathsf{fopt}}{196d} - M \geq \frac{\mathsf{fopt}}{200d}$. ◀

Finally, we are ready to prove the main theorems in the online setting.

▶ **Theorem 30** (Unconstr. Case: Supermodular Cost)**.** *Algorithm 5.1 gives an $O(d)$-approximation in expectation for the unconstrained case, if the cost function is supermodular.*

Proof. Define the event $\mathcal{E}$ that Lemmas 28 and 29 hold; $\Pr(\mathcal{E}) \geq 17/20$. Now, by Lemma 29(c), the optimal fractional solution for R°_μ has profit at least $\mathsf{fopt}/200d$. Moreover, since there are no constraints, the profit function is monotone submodular over all of $U^\circ_{\lambda^*}$ by Lemma 23. Conditioning on the good event $\mathcal{E}$, Lemma 28(a) gives that $R^\circ_\mu \subseteq U^\circ_{\lambda^*}$, so the algorithm to maximize the monotone submodular function (both integrally and fractionally) is to pick all elements. Hence, conditioned on $\mathcal{E}$, the profit we get is at least $\mathsf{fopt}/200d$. In the other case, we never pick an item that gives negative marginal value, so our solution is always non-negative. Hence our expected profit is at least $\Pr[\mathcal{E}] \cdot \mathsf{opt}(R_\mu) = \Omega(\mathsf{fopt}/d) \geq \Omega(\mathsf{opt}/d)$. ◀

The analysis of the algorithm for the constrained separable-cost case is similar, only using the constrained offline guarantees of Theorem 21, and the non-negativization Lemma 23 to argue that SubmodMS maintains its guarantees.

▶ **Theorem 31** (Constr. Case: Separable Cost)**.** *Suppose algorithm SubmodMS satisfies Assumption 25 and is α-competitive in expectation. Then Algorithm 5.1 gives a $O(\alpha d^2)$-approximation in expectation.*

6 Separability versus Supermodularity

In this section, we show that an β-approximation algorithm for the separable-cost case gives a $O(d\beta)$-approximation for a slight generalization of the supermodular-cost case. Consider the problem of picking a set A to solve

$$\pi(A) := \max_{A \in \mathscr{F}} \left(v(A) - g\Big(\sum_{e \in A} s(e)\Big) \right),$$

where $v(A)$ is a (discrete) submodular function over $\{0,1\}^n$ with $v(\emptyset) = 0$, g is a convex, (continuous) supermodular function over $\mathbb{R}^d$, and $\mathscr{F}$ is some downward-closed constraint set. We show that for the case of matroid constraints, this problem can be reduced to the setting where the cost function is separable over its d coordinates, suffering a loss of $O(d)$.

▶ **Theorem 32** (Reduction)**.** *Given an β-approximation algorithm for profit-maximization for* separable *convex cost functions under matroid constraints, we can get an $d(\beta + 2\mathrm{e}d)$-approximation algorithm for the profit-maximization problem with* supermodular costs g*,* submodular values v*, and $\mathscr{F}$ being a* matroid *constraint.*

The reduction is the following:

1. Define separable costs $\overline{g}(y) := 1/d \sum_{i=1}^d g_i(dy_i)$, where g_i are marginal functions for g.
2. W.p. $p = \frac{\beta}{\beta + 2\mathrm{e}d}$, run single-secretary algorithm to return element with maximum profit.
3. W.p. $1 - p = \frac{2\mathrm{e}d}{\beta + \mathrm{e}d}$, run algorithm for value function $v(\cdot)$ and separable cost fn. $\overline{g}(\cdot)$.

This reduction relies on the following simple but perhaps surprising observation that relates separability with supermodularity, which may find other applications.

▶ **Lemma 33.** *Given a monotone convex superadditive function g with $g(0) = 0$, let g_i be the marginal functions. Then for all $y \in \mathbb{R}^d_+$:*

1. $g(y) \geq \sum_i g_i(y_i)$
2. $g(y) \leq \frac{1}{d} \sum_i g_i(dy_i) = \overline{g}(y)$.

Proof. The first property follows from the superadditivity of g, and the second follows from Jensen's inequality. ◀

While the full proof of Theorem 32 is deferred to the full version of the paper, the main idea is clean. Given an optimal integer solution x^* for the original problem (with the original cost function), we use Lemma 33 and the Lovász (convex) extension of submodular functions to show that x^*/d is a good fractional solution for the separable cost function. Now using polyhedral properties of d-dimensional faces of the matroid polytope, and other properties of the Lovász extension, we show the existence of a good integer solution to the separable problem. Combining this reduction with Theorem 2 proves Theorem 3.

References

1. Shipra Agrawal and Nikhil R. Devanur. Fast algorithms for online stochastic convex programming. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1405–1424. SIAM, Philadelphia, PA, 2015. doi:10.1137/1.9781611973730.93.
2. Shipra Agrawal, Zizhuo Wang, and Yinyu Ye. A dynamic near-optimal algorithm for online linear programming. *Oper. Res.*, 62(4):876–890, 2014. doi:10.1287/opre.2014.1289.
3. Yossi Azar, Niv Buchbinder, T-H. Hubert Chan, Shahar Chen, Ilan R. Cohen, Anupam Gupta, Zhiyi Huang, Ning Kang, Viswanath Nagarajan, Joseph (Seffi) Naor, and Debmalya Panigrahi. Online algorithms for covering and packing problems with convex objectives. In *57th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2016, New Brunswick, NJ, USA, October 9-11, 2016*, 2016.
4. Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *APPROX-RANDOM*, pages 16–28, 2007. doi:10.1007/978-3-540-74208-1_2.
5. Siddharth Barman, Seeun Umboh, Shuchi Chawla, and David Malec. Secretary problems with convex costs. In *Automata, languages, and programming. Part I*, volume 7391 of *Lecture Notes in Comput. Sci.*, pages 75–87. Springer, Heidelberg, 2012. doi:10.1007/978-3-642-31594-7_7.
6. Avrim Blum, Anupam Gupta, Yishay Mansour, and Ankit Sharma. Welfare and profit maximization with production costs. In *FOCS*, pages 77–86, Nov 2011. doi:10.1109/FOCS.2011.68.
7. Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Adaptivity and approximation for stochastic packing problems. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*, pages 395–404, 2005. URL: http://dl.acm.org/citation.cfm?id=1070432.1070487.
8. Nikhil R. Devanur, Kamal Jain, Balasubramanian Sivan, and Christopher A. Wilkens. Near optimal online algorithms and fast approximation algorithms for resource allocation problems. In Yoav Shoham, Yan Chen, and Tim Roughgarden, editors, *ACM Conference on Electronic Commerce*, pages 29–38. ACM, 2011. doi:10.1145/1993574.1993581.
9. Reza Eghbali and Maryam Fazel. Designing smoothing functions for improved worst-case competitive ratio in online optimization. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3279–3287, 2016. URL: http://papers.nips.cc/paper/6073-designing-smoothing-functions-for-improved-worst-case-competitive-ratio-in-online-optimization.
10. Reza Eghbali, Jon Swenson, and Maryam Fazel. Exponentiated subgradient algorithm for online optimization under the random permutation model. *CoRR*, abs/1410.7171, 2014. URL: http://arxiv.org/abs/1410.7171, arXiv:1410.7171.

11 Moran Feldman, Ola Svensson, and Rico Zenklusen. A simple $O(\log \log(\text{rank}))$-competitive algorithm for the matroid secretary problem. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1189–1201. SIAM, Philadelphia, PA, 2015. doi:10.1137/1.9781611973730.79.

12 Moran Feldman and Rico Zenklusen. The submodular secretary problem goes linear. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science—FOCS 2015*, pages 486–505. IEEE Computer Soc., Los Alamitos, CA, 2015.

13 P. R. Freeman. The secretary problem and its extensions: a review. *Internat. Statist. Rev.*, 51(2):189–206, 1983. doi:10.2307/1402748.

14 Anupam Gupta and Marco Molinaro. How the experts algorithm can help solve lps online. *Math. Oper. Res.*, 41(4):1404–1431, 2016. doi:10.1287/moor.2016.0782.

15 Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Fundamentals of convex analysis*. Grundlehren Text Editions. Springer-Verlag, Berlin, 2001. doi:10.1007/978-3-642-56468-0.

16 Zhiyi Huang and Anthony Kim. Welfare maximization with production costs: a primal dual approach. In *26th SODA*, pages 59–72. SIAM, Philadelphia, PA, 2015. doi:10.1137/1.9781611973730.6.

17 Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. Primal beats dual on online packing LPs in the random-order model. In *STOC'14—Proceedings of the 2014 ACM Symposium on Theory of Computing*, pages 303–312. ACM, New York, 2014.

18 Robert Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '05, pages 630–631, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics. URL: http://portal.acm.org/citation.cfm?id=1070432.1070519.

19 Oded Lachish. O(log log rank) competitive ratio for the matroid secretary problem. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 326–335, 2014. doi:10.1109/FOCS.2014.42.

20 Marco Molinaro and R. Ravi. The geometry of online packing linear programs. *Math. Oper. Res.*, 39(1):46–59, 2014. doi:10.1287/moor.2013.0612.

21 Donald M. Topkis. *Supermodularity and complementarity*. Frontiers of Economic Research. Princeton University Press, Princeton, NJ, 1998.

Generic Single Edge Fault Tolerant Exact Distance Oracle

Manoj Gupta
IIT Gandhinagar, Gandhinagar, India
gmanoj@iitgn.ac.in

Aditi Singh
IIT Gandhinagar, Gandhinagar, India
aditi.singh@iitgn.ac.in

Abstract

Given an undirected unweighted graph G and a source set S of $|S| = \sigma$ sources, we want to build a data structure which can process the following query $\mathrm{Q}(s, t, e)$: find the shortest distance from s to t avoiding an edge e, where $s \in S$ and $t \in V$. When $\sigma = n$, Demetrescu, Thorup, Chowdhury and Ramachandran (SIAM Journal of Computing, 2008) designed an algorithm with $\tilde{O}(n^2)$ space[1] and $O(1)$ query time. A natural open question is to generalize this result to any number of sources. Recently, Bilò et. al. (STACS 2018) designed a data-structure of size $\tilde{O}(\sigma^{1/2}n^{3/2})$ with the query time of $O(\sqrt{n\sigma})$ for the above problem. We improve their result by designing a data-structure of size $\tilde{O}(\sigma^{1/2}n^{3/2})$ that can answer queries in $\tilde{O}(1)$ time.

In a related problem of finding fault tolerant subgraph, Parter and Peleg (ESA 2013) showed that if detours of *replacement* paths ending at a vertex t are disjoint, then the number of such paths is $O(\sqrt{n\sigma})$. This eventually gives a bound of $O(n\sqrt{n\sigma}) = O(\sigma^{1/2}n^{3/2})$ for their problem. *Disjointness of detours* is a very crucial property used in the above result. We show a similar result for a subset of replacement path which **may not** be disjoint. This result is the crux of our paper and may be of independent interest.

2012 ACM Subject Classification Theory of computation → Shortest paths, Theory of computation → Graph algorithms analysis, Theory of computation → Data structures design and analysis

Keywords and phrases Fault Tolerant Algorithms, Graph Algorithms, Distance Oracles, Data-Structures

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.72

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.00190`.

1 Introduction

Real life graph networks like communication network or road network are prone to link or node failure. Thus, algorithms developed for these networks must be resilient to failure. For example, the shortest path between two nodes may change drastically even if a single link fails. So, if the problem forces us to find shortest paths in the graph, then it should find the next best shortest path after a link failure. There are many ways to model this process: one of them is *fault-tolerant graph algorithm*. In this model, we have to preprocess a graph G and build a data-structure that can compute a property of the graph after any k edges/vertices

[1] $\tilde{O}(\cdot)$ hides poly $\log n$ factor.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 72; pp. 72:1–72:15


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of the graph have failed. Note the difference between this model and *dynamic graph model*. In a dynamic graph algorithm, we have to *maintain* a property of a continuously changing graph. However, in the fault tolerant model, we expect the failure to be repaired readily and restore our original graph.

In this paper, we study the shortest path problem in the fault tolerant model. Formally, we are given an undirected and unweighted graph G and a source set S of $|S| = \sigma$ sources. We want to build a data structure which can process the following query $\mathrm{Q}(s, t, e)$: find the shortest distance from s to t avoiding an edge e, where $s \in S$ and $t \in V$. Such a data-structure is also called a *distance oracle*. When there are n sources, Demetrescu et al. [9] designed an oracle that can find the shortest path between any two vertices in G after a single vertex/edge failure in $\tilde{O}(n^2)$ space and $O(1)$ query time. Recently, Bilò et. al. [5] generalized this result to any number of sources. They designed a data-structure of size $\tilde{O}(\sigma^{1/2}n^{3/2})$ with the query time of $O(\sqrt{n\sigma})$ for the above problem.

To understand our problem, we should also understand a closely related problem of finding *fault tolerant subgraph*. Here, we have to find a subgraph of G such that BFS tree from $s \in S$ is preserved in the subgraph after any edge deletion. In an unweighted graph, a BFS tree preserves the shortest path from s to all vertices in G. Parter and Peleg [17] showed that a subgraph of size $O(\sigma^{1/2}n^{3/2})$ is both necessary and sufficient to solve the above problem. The above result indicates that there should be a better fault-tolerant distance oracle for any value of σ.

Inspired by this result, we generalize the result of [9] to any number of sources – by showing that there exists a distance oracle of size $\tilde{O}(\sigma^{1/2}n^{3/2})$ which can answer queries in $\tilde{O}(1)$ time. Note that our result nearly matches the space bound achieved by Parter and Peleg[17] – up to polylog n factors. We now state the main result of this paper formally:

▶ **Theorem 1.** *There exists a data-structure of size $\tilde{O}(\sigma^{1/2}n^{3/2})$ for multiple source single fault tolerant exact distance oracle that can answer each query in $\tilde{O}(1)$ time.*

This generalization turns out to be much more complex than the result in [9]. Indeed, the techniques used by Demetrescu et al. [9] are also used by us to weed out *easy replacement* paths. To take care of other paths, we take an approach similar to Parter and Peleg[17]. They used the following trick: if the *detour* of replacement paths are *disjoint*, then the number of such paths can be bounded easily by a *counting argument*. The main challenge is then to show that paths in question are indeed disjoint – which is also easy in their problem. We use a technique similar to above – however, our paths are not disjoint, they may intersect. We believe that this technique can be of independent interest and may be used in solving closely related fault tolerant subgraph problems.

1.1 Related Work

Prior to our work, the work related to fault tolerant distance oracle was limited to two special cases, $\sigma = 1$ or $\sigma = n$. As stated previously, Demetrescu et al. [9] designed a single fault tolerant distance oracle of size $\tilde{O}(n^2)$ with a query time of $O(1)$. The time to build the data-structure is $O(mn^2)$ – which was improved to $O(mn \log n)$ by Bernstein and Karger [4]. The above result also works for a directed weighted graph. Pettie and Duan [10] were able to extend this result to two vertex faults. The size and query time of their distance oracle is $\tilde{O}(n^2)$ and $\tilde{O}(1)$ respectively. If the graph is weighted, then Demetrescu et al. [9] showed that there exists a graph in which a single vertex fault tolerant distance oracle will take $\Omega(m)$ space. Recently, Bilò et. al. [5] designed the following data-structure: for every $S, T \subseteq V$, a data-structure of size $\tilde{O}(n\sqrt{|S||T|})$ and query time $O(\sqrt{|S||T|})$, where the query asks for the shortest distance from $s \in S$ to $t \in T$ avoiding any edge. If $|S| = \sigma$ and $|T| = n$, then the size of their data-structure is $\tilde{O}(\sigma^{1/2}n^{3/2})$ and the query time is $O(\sqrt{n\sigma})$.

The next set of results are not *exact* but *approximate*, that is, they return an approximate distance (by a multiplicative *stretch* factor) between two vertices after an edge/vertex fault. Also, these oracles work for a single source only. Baswana and Khanna [14] showed that a 3-stretch single source single fault tolerant distance oracle of size $\tilde{O}(n)$ can be built in $\tilde{O}(m+n)$ time and a constant query time. Bilò et. al.[6] improved the above result: a distance oracle with stretch 2 of size $O(n)$ and $O(1)$ query time. In another result, Bilò et. al. [7] designed a k fault tolerant distance oracle of size $\tilde{O}(kn)$ with a stretch factor of $(2k+1)$ that can answer queries in $\tilde{O}(k^2)$ time. The time required to construct this data-structure is $O(kn\alpha(m,n))$, where $\alpha(m,n)$ is the inverse of the Ackermann's function. If the graph is unweighted, then Baswana and Khanna[14] showed that a $(1+\epsilon)$-stretch single source fault tolerant distance oracle of size $\tilde{O}(\frac{n}{\epsilon^3})$ can be built in $O(m\sqrt{n/\epsilon})$ time and a constant query time. Bilò et. al [6] extended this result for weighted graph by designing a distance oracle with stretch $(1+\epsilon)$ of size $O(\frac{n}{\epsilon}\log\frac{1}{\epsilon})$ and a logarithmic query time.

There is another line of work, called the *replacement path* problem. In this problem, we are given a source s and destination t and for each edge e on the shortest st path, we need to find shortest s to t path avoiding e. The problem can be generalized to finding k shortest s to t path avoiding e. The main goal of this problem is to find all shortest paths as fast as possible. Malik et al. [15] showed that in an undirected graphs, replacement paths can be computed in $O(m+n\log n)$ time. For directed, unweighted graphs, Roditty and Zwick [19] designed an algorithm that finds all replacement paths in $O(m\sqrt{n})$ time. For the k-shortest paths problem, Roditty [18] presented an algorithm with an approximation ratio 3/2, and the running time $O(k(m\sqrt{n}+n^{3/2}\log n))$. Bernstein [3] improved the above result to get an approximation factor of $(1+\epsilon)$ and running time $O(km/\epsilon)$. The same paper also gives an improved algorithm for the approximate st replacement path algorithm. See also [11, 22, 21].

As mentioned previously, a problem closely related to our problem is the fault tolerant subgraph problem. The aim of this problem is to find a subgraph of G such that BFS tree from $s \in S$ is preserved in the subgraph after any edge deletion. Parter and Peleg [17] designed an algorithm to compute single fault tolerant BFS tree with $O(n^{3/2})$ space. They also showed their result can be easily extended to multiple source with $O(\sigma^{1/2}n^{3/2})$ space. Moreover, their upper bounds were complemented by matching lower bounds for both their results. This result was later extended to dual fault BFS tree by Parter [16] with $O(n^{5/3})$ space. Gupta and Khan [12] extended the above result to multiple sources with $O(\sigma^{1/3}n^{5/3})$ space. All the above results are optimal due to a result by Parter [16] which states that a multiple source k fault tolerant BFS structure requires $\Omega(\sigma^{\frac{1}{k+1}}n^{2-\frac{1}{k+1}})$ space. Very recently, Bodwin et. al. [8] showed the existence of a k fault tolerant BFS structure of size $\tilde{O}(k\sigma^{1/2^k}n^{2-1/2^k})$.

Other related problems include fault-tolerant DFS and fault tolerant reachability. Baswana et al. [1] designed an $\tilde{O}(m)$ sized fault tolerant data structure that reports the DFS tree of an undirected graph after k faults in $\tilde{O}(nk)$ time. For single source reachability, Baswana et al. [2] designed an algorithm that finds a fault tolerant reachability subgraph for k faults using $O(2^k n)$ edges.

2 Preliminaries

We use the following notation throughout the paper:

- xy : Given two vertices x and y, let xy denote a path between x and y. Normally this path will be the shortest path from x to y in G. However, in some places in the paper, the use of xy will be clear from the context.
- $|xy|$: It denotes the number of edges in the path xy.

- $(\cdot \diamond \cdot)$: Given two paths sx and xt, $sx \diamond xt$ denotes the concatenation of paths sx and xt.
- *after or below/before or above* x : We will assume that the st path (for $s \in S$ and $t \in V$) is drawn from top to bottom. Assume that $x \in st$. The term *after or below* x on st path refers to the path xt. Similarly *before or above* x on st path refers to the path sx.
- *replacement path*: The shortest path that avoids any given edge is called a *replacement* path.

3 Our Approach

We will randomly select a set of terminals $\mathcal{T}$ by sampling each vertex with probability $\sqrt{\frac{\sigma}{n}}$. Note that the size of $\mathcal{T}$ is $\tilde{O}(\sqrt{\sigma n})$ with high probability. For a source s and $t \in V$, let t_s be the last terminal encountered on the st path. The following lemma is immediate:

▶ **Lemma 2.** *If* $|st| \geq c\sqrt{\frac{n}{\sigma}} \log n$ $(c \geq 3)$, *then* $|t_s t| = \tilde{O}(\sqrt{\frac{n}{\sigma}})$ *with a very high probability for all* $s \in S$ *and* $t \in V$.

Let G_p denote the graph where each edge is perturbed by a weight function that ensures unique shortest paths. Our st path is the shortest s to t path in G_p, let us denote its length by $|st|_p$. Note that G_p contains a unique shortest path between any two vertices, even the ones that avoid an edge – such a graph has been used before in related problems [4, 17, 13]. We can use G_p even to find all the replacement paths. However, we want our replacement paths to have other nice property, that is, *the length replacement paths(without perturbation) from* s *to* t *are different.* This property is not satisfied by replacement paths in G_p. We employ another simple strategy to find a replacement path. Following [12], we define preferred replacement paths:

▶ **Definition 3.** A path P is called a **preferred** replacement path from s to t avoiding e if (1) it diverges and merges the st path *just once* (2) it divergence point from the st path is as close to s as possible (3) it is the shortest path in G_p satisfying (1) and (2).

The replacement path has to diverge from the st path before e. Ideally, we want a replacement path that diverges from st path as close to s as possible. This is a crucial feature which will ensure that all replacement paths from s to t have different lengths. The first condition ensures that we do not diverge from st path just to get a higher point of divergence. If many shortest paths are diverging from a same vertex, the third condition is used to break ties. In the ensuing discussion, we will assume that we are always working with a preferred replacement path.

The initial st path is found out by finding the unique shortest path in G_p. Consider the query $Q(s, t, e)$. If the failed edge e does not lie on st path, then we can report $|st|$ as the shortest distance from s to t avoiding e. To this end, we should be able to check whether e lies in the shortest path from s to t. At this point, we will use the property of graph G_p. If $e(u, v)$ lies in st path, then we have to check if u and v lie on st path. To this end, we check if $|su|_p + |ut|_p = |st|_p$ and $|sv|_p + |vt|_p = |st|_p$. If both the above two equations are satisfied then the st path passes through e (as the shortest path from u to v is 1). We can also find whether u or v is closer to s on st path. Without loss of generality assume that u is closer to s than v on st path.

However, we do not have space to store all these distances. Specifically, the second term on the LHS of above two equations mandates that we store the distance of every pair of vertices in the graph. This implies that the size of our data structure is $O(n^2)$ which is not desirable.

To solve the above problem, we observe that if e lies in the $t_s t$ path, then we have just enough space to store this fact. So, given any e, we can easily find if $e \in t_s t$. If $e \in st_s$, then

we know that $|su|_p + |ut_s|_p + |t_st|_p = |st|_p$ and $|sv|_p + |vt_s|_p + |t_st|_p = |st|_p$. This equality is easier to check with the space at hand. So, we have the following two cases:

1. (Near Case) e lies on t_st.
2. (Far Case) e lies on st_s.

3.1 Handling the Near Case

For each $e(u,v) \in t_st$, let P_e be the preferred replacement path from s to t avoiding e. We put $(e, |P_e|)$ in a balanced binary search tree $\textsc{Bst}(s,t)$ with the key being e. Given any query $Q(s,t,e)$, we now need to check if e lies in $\textsc{Bst}(s,t)$. This can be done in $\tilde{O}(1)$ time and the length of the preferred replacement path can be reported.

The space required for $\textsc{Bst}(s,t)$ is directly proportional to the size of path t_st. By Lemma 2, we know that $|t_st| = \tilde{O}(\sqrt{\frac{n}{\sigma}})$. Thus, the size of $\textsc{Bst}(s,t) = \tilde{O}(\sqrt{\frac{n}{\sigma}})$. This implies that the cumulative size of all the associated binary search tree is $\cup_{t\in V} \cup_{s\in S} |t_st| = \tilde{O}(n\sigma\sqrt{\frac{n}{\sigma}}) = \tilde{O}(\sigma^{1/2}n^{3/2})$.

3.2 Handling the Far Case

We first need to check if $e \in st_s$. To this end we use the following data-structures.

- B_0: For each pair of vertices x and y where $x \in (S \cup \mathcal{T})$ and $y \in V$, the shortest path between x and y in G and G_p is stored in $B_0(x,y)$ and $B_0^p(x,y)$ respectively. The total size of B_0 is $\tilde{O}((\sigma + \sqrt{n\sigma})n) = \tilde{O}(\sigma^{1/2}n^{3/2})$.
- B_1: For each pair of vertices $s \in S$ and $t \in V$, $B_1(s,t)$ contains the vertex in $\mathcal{T}$ closest to t on st path, that is t_s. The total size of B_1 is $O(\sigma n) = \tilde{O}(\sigma^{1/2}n^{3/2})$.

To check if $e(u,v) \in st_s$, we first find $t_s \leftarrow B_1(s,t)$. Then we check if $B_0^p(s,u) + B_0^p(u,t_s) + B_0^p(t_s,t) = B_0^p(s,t)$ and $B_0^p(s,v) + B_0^p(v,t_s) + B_0^p(t_s,t) = B_0^p(s,t)$. If yes, then $e \in st_s$. We subdivide the far case into two more sub-cases:

1. The preferred replacement path avoiding e passes through t_s.
2. The preferred replacement path avoiding e avoids t_s.

The first case turns out to be a generalization of techniques used by Demetrescu et. al.[9] to solve the all pair distance oracle under single edge/vertex failure – we will use the compact version of this algorithm presented by Pettie and Duan [10]. The second case is a *new and unexplored* case. We will show that we can bound the number of preferred replacement paths in this case to $O(\sqrt{n\sigma})$ for a fixed vertex t. This would imply that the total number of such paths is $O(\sigma^{1/2}n^{3/2})$. We are able to bound the number of paths even though these paths may intersect with each other – this is a new feature of our analysis which is much different from the analysis done by Parter and Peleg [17] on a related problem.

Section 4 deals with the first case. In Section 5, we will apply our new approach to the special case when $\sigma = 1$, or there is a single source. In Section 6, we will discuss the potential problems in extending our approach to multiple sources. Section 7 and 8 extends our approach to multiple sources and in Section 9 we develop our data-structure that can answer queries in $\tilde{O}(1)$ time. To save space, we have omitted proofs in this extended abstract. The concerned reader may read the proof in the full version of the paper.

4 Preferred replacement path passes through t_s

Since this case is a generalization of the techniques developed by Demetrescu et. al. [9], concerned reader may read the full version of the paper for details – where we show that there exists a data-structure of size $\tilde{O}(\sigma^{1/2}n^{3/2})$ which takes $O(1)$ time to find a replacement path from s to t that avoids e but passes through t_s.

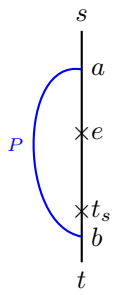


Figure 1 P does not intersect any path in $(> P)$.

Now, we move on to the harder case, that is, replacement paths avoid t_s too. For this, we will fix a vertex t. We will show that the query $Q(s, t, e(u, v))$ can be answered in $\tilde{O}(1)$ using $\tilde{O}(\sqrt{\sigma n})$ space. This immediately implies that we can answer exact queries in $\tilde{O}(1)$ time using $\tilde{O}(\sigma^{1/2} n^{3/2})$ space.

5 Preferred Replacement path avoids t_s

Handling preferred replacement paths that avoid t_s turns out to be a challenging and unexplored case. For better exposition, we will first solve the problem for the case when $\sigma = 1$, that is there is only one source. Let $\mathcal{R}$ be the set of all preferred replacement paths from s to t that do not pass through t_s. We make two important observations:

1. The size of $\mathcal{R}$ is $O(\sqrt{n})$.
2. Preferred replacement paths in $\mathcal{R}$ avoid one contiguous sub path of st.

Few remarks are in order. If the preferred replacement paths in $\mathcal{R}$ were disjoint, then bounding the size of $\mathcal{R}$ is easy. However, we are able to bound the size of $\mathcal{R}$ even if paths are intersecting. The second observation implies that we can build a balanced binary search tree containing paths in $\mathcal{R}$. Each node in this tree will contain a preferred replacement path P. The key for each node will be the start and end vertex of the sub path P avoids. We will use this BST to find an appropriate replacement path that avoids an edge e.

▶ **Definition 4.** (Detour of a replacement path) Let P be a preferred replacement path avoiding an edge e on st path. Then detour of P is defined as, $\textsc{Detour}(P) := P \setminus st$. That is, detour is a path the leaves st before e till the point it merges back to st again.

Since our replacement path P also avoids t_s, the following lemma is immediate by the definition of preferred path.

▶ **Lemma 5.** *Let P be a preferred replacement path in $\mathcal{R}$ that avoids e and t_s on st path, then (1) $\textsc{Detour}(P)$ cannot merge back to st_s path and (2) $\textsc{Detour}(P)$ is a contiguous path.*

▶ **Lemma 6.** *Let $P, P' \in \mathcal{R}$ avoid e and e' respectively on st_s path. Also assume that e is closer to s than e'. Then (1) P avoids e' (2) $\textsc{Detour}(P')$ starts after e on st_s path and (3) $|P| > |P'|$.*

The converse of the third part of the lemma is also true. Since we will be using it in future, we prove it now.

▶ **Lemma 7.** *Let P and P' be two preferred replacement paths that avoid e and e' on st path respectively. If $|P| > |P'|$, then e is closer to s than e'.*

By Lemma 6(3), we know that all preferred replacement paths in $\mathcal{R}$ have different lengths. In fact, it is the main reason we defined a preferred replacement path. We can thus arrange these paths in decreasing order of their lengths. Thus, we get the following corollary.

▶ **Corollary 8.** *Given a set $\mathcal{R}$ of preferred replacement paths from s to t (that also avoid t_s), we can arrange paths in decreasing order of their lengths.*

Given a path $P \in \mathcal{R}$, let $(< P)$ be the set of all preferred replacement paths with length less than P. Similarly, let $(> P)$ be the set of all preferred replacement paths with length greater than P. If P avoids e, then by Lemma 7, it also avoids all edges avoided by paths in $(< P)$. By Lemma 6, for any path $P' \in (< P)$, $\textsc{Detour}(P')$ starts after e on st_s path. We will now show a simple but important property of a path P in $\mathcal{R}$.

▶ **Lemma 9.** *Let $P \in \mathcal{R}$ be the shortest path from s to t avoiding e such that $|P| = |st| + \ell$ where $\ell \geq 0$, then the size of the set $(< P)$ is $\leq \ell$.*

▶ **Definition 10.** (Unique path of P) Let $\textsc{Unique}(P)$ be the prefix of $\textsc{Detour}(P)$ which does not intersect with any detours in $\cup_{P' \in (> P)} \textsc{Detour}(P')$.

We now arrange all preferred replacement paths in $\mathcal{R}$ in decreasing order of their lengths. Assume that we are processing a path P according to this ordering such that P avoids e on st path. If $|\textsc{Unique}(P)| \geq \sqrt{n}$, then we have associated $O(\sqrt{n})$ vertices on $\textsc{Unique}(P)$ to P. Else $\textsc{Unique}(P) < \sqrt{n}$ and we have the following two cases:

5.1 Detour(P) does not intersect with detour of any path in $(> P)$

Let $\textsc{Detour}(P)$ start at a and end at b – the vertex where it touches $t_s t$ path. Let ab denote the path from a to b on P. By our assumption $\textsc{Unique}(P) = ab$ and $|ab| < \sqrt{n}$. By Lemma 6, all replacement paths in $(< P)$ pass through e (as detour of these replacement paths start below e) and by Lemma 7, these replacement paths avoid edges that are closer to t than e. We can view the replacement paths as if they are starting from the vertex a. That is, consider paths $\{P \setminus sa\} \cup \{P' \setminus sa|\ P' \in (< P)\}$. These replacement paths avoid edges in at. $|P \setminus sa| = |ab| + |bt| \leq |ab| + |at| < |at| + \sqrt{n}$. Applying Lemma 9, we infer that the number of paths in $\{P' \setminus sa | P' \in (< P)\}$ is $\leq \sqrt{n}$

5.2 Detour(P) intersects with detour of a path in $(> P)$

Assume that P first intersects with $P' \in (> P)$. Let P' avoid e' and $\textsc{Detour}(P')$ start at a' and end at b' (see Figure 2). Let us assume that $\textsc{Detour}(P)$ starts at a and it intersects $\textsc{Detour}(P')$ at c. This implies that $\textsc{Unique}(P) = ac$.

Consider the path $sa' \diamond a'c \diamond ca \diamond at$. We claim that this path avoids e'. This is due to the fact that by Lemma 6, $\textsc{Detour}(P)$ starts after e' on st path. So, ca and at avoids e'. Since $P' = sa' \diamond a'c \diamond cb' \diamond b't$, length of P' must be $\leq$ length of the alternate path. Thus,

$$\begin{array}{lrcl} & |sa'| + |a'c| + |cb'| + |b't| & \leq & |sa'| + |a'c| + |ca| + |at| \\ \implies & |cb'| + |b't| & \leq & |ca| + |at| \\ \implies & |ac| + |cb'| + |b't| & \leq & 2|ca| + |at| \end{array}$$

On the left hand of the inequality, we have a path from a to t avoiding e. So, its length should be $\geq$ length of the preferred path $P \setminus sa$. Thus $|P \setminus sa| \leq 2|ca| + |at| \leq 2\sqrt{n} + |at|$. By Lemma 6, all replacement paths in $(< P)$ pass through e (as detour of these replacement paths start below e) and by Lemma 7, these replacement paths avoid edges that are closer to t than e. We can view the replacement paths as if they are starting from the vertex a. That is, consider paths $\{P \setminus sa\} \cup \{P' \setminus sa | P' \in (< P)\}$. Applying lemma 9, we infer that the number of paths in $\{P' \setminus sa | P' \in (< P)\}$ is $\leq 2\sqrt{n}$.

Our arguments above point to the following important observation: *Once we find a replacement path in $\mathcal{R}$ with unique path length $< \sqrt{n}$, then there are at most $2\sqrt{n}$ replacement paths in $\mathcal{R}$ left to process.* Since there can be at most $\sqrt{n}$ paths in $\mathcal{R}$ with unique path length $\geq \sqrt{n}$, we have proven the following lemma:

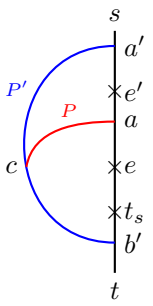


Figure 2 P intersects first with $P' \in (> P)$ at c.

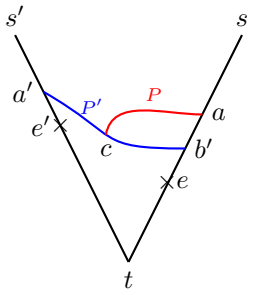


Figure 3 The bad case for us: $P' \in (> P)$ intersects with P and then passes through the edge e that P avoids

▶ **Lemma 11.** $|\mathcal{R}| = O(\sqrt{n})$.

We now build a data-structure which will exploit Lemma 11. However, we need another key but simple observation. By Lemma 6, if $|P| > |P'|$, then $\textsc{Detour}(P')$ starts below the edge avoided by P. This lemma implies that $\textsc{Detour}(P')$ starts below all edges avoided by P. Thus P avoids some contiguous path in st_s and detour of all replacement paths in $(< P)$ start below the last edge (which is closer to t_s) in this subpath. Thus, we have proved the second key lemma:

▶ **Lemma 12.** *A replacement path P avoids a contiguous subpath of st.*

Let $\textsc{First}(P)$ and $\textsc{Last}(P)$ denote the first and the last vertex of the contiguous path that P avoids. Given a vertex v, let $v.depth$ denote the depth of v in the BFS tree of s. We can store the depth of all vertices in an array (takes $O(n)$ space). Lastly, we build a balanced binary search tree $\text{BST}(t)$ in which each node represents a path P. The key used to search the node is the range: $[\textsc{First}(P).depth, \textsc{Last}(P).depth]$. By Lemma 12, all replacement paths avoid contiguous subpaths of st_s. These contiguous paths are also disjoint as there is only one preferred path avoiding an edge. Thus, the key we have chosen forms a total ordered set with respect to the relation $\{<, >\}$. The size of $\text{BST}(t)$ is $O(\sqrt{n})$ as the size of $\mathcal{R}$ is $O(\sqrt{n})$. We are now ready to process any query $\text{Q}(s, t, e(u, v))$. We just need to search for an interval in $\text{BST}(t)$ that contains $u.depth$ and $v.depth$. This can be done in $\tilde{O}(1)$ time. Thus we have proved the following theorem:

▶ **Theorem 13.** *There exists a data-structure of size $\tilde{O}(n^{3/2})$ for single source single fault tolerant exact distance oracle that can answer each query in $\tilde{O}(1)$ time.*

6 From Single Source to Multiple Sources

Unfortunately, the analysis for the single source case is not easily extendible to multiple source case. We identify the exact problem here. Consider the case described in Section 5.2. In this case, we show that if $|P'| > |P|$ and P intersects with P', then there is a path available for P (that is $ac \diamond cb' \diamond b't$). We can use this path because it also avoids e (the edge avoided by path P). First, we show that the above assertion is not true when we move to multiple source case. Consider the following example (See Figure 3). Here, P avoids e on st path and P' avoids e' on $s't$ path. Detour(P) starts at a and its intersects P' at c. Detour(P') starts at a' and it hits st path at b' and then passes through e. Note that the full path P from s to t is not shown in Figure 3. The reader can check that the path $ac \diamond cb' \diamond b't$ is not an alternate path for P as it passes through e. We say that such a path is a bad path because it breaks the easy analysis of single source case (we will formally define bad paths in Section 8.2). However, we are able to show that the total number of *good* paths (paths which are not bad) is $\geq$ the number of bad paths. Good paths exhibit properties similar to the set $\mathcal{R}$ in Section 5. This will help us in bounding them (and thus bad paths too). Once again we will fix a vertex t and show that the number of replacement paths from $s \in S$ to t that also avoids t_s is $O(\sqrt{\sigma n})$. Let BFS(t) denote the union of all shortest paths from t to $s \in S$. The reader can check that the union of these paths does not admit a cycle, so we can assume that its a tree rooted at t. Since BFS(t) has at most σ leaves, the number of vertices with degree > 2 in BFS(t) is $O(\sigma)$. We now contract all the vertices of degree 2 (except t and $s \in S$) in BFS(t) to get a tree that only contains leaves of BFS(t), the root t, all the sources and all other vertices with degree > 2 in BFS(t).

▶ **Definition 14.** (σ-BFS(t)) σ-BFS(t) is a tree obtained by contracting all the vertices with degree exactly 2 in BFS(t) except t and source $s \in S$.

▶ **Definition 15.** (Intersection vertex and segment in σ-BFS(t))
Each node σ-BFS(t) is called an intersection vertex. An edge $xy \in \sigma$-BFS(t) denotes a path between two vertices in BFS(t). We call such an edge in σ-BFS a segment. We use this term in order to differentiate between edges in BFS(t) and σ-BFS(t). Also, we will use the following convention: if xy is a segment, then y is closer to t than x.

σ-BFS(t) has at most σ vertices with degree ≤ 2. This implies that there are at most $O(\sigma)$ intersection vertices and segments in σ-BFS(t).

As in the single source case, we would like to find the preferred path for each avoided edge on the st path where $s \in S$. However, we don't have enough space to store all these paths. Also storing all paths seems wasteful. Consider two preferred replacement paths P and P' that start from s and s' respectively. These two paths meet at an intersection vertex x after which they are same, that is, they take the same detour to reach t. Storing both P and P' seems wasteful as they are essentially the same path once they hit x. To this end, we only store preferred path corresponding to each segment in σ-BFS(t). We now describe our approach in detail.

Let xy be a segment in σ-BFS(t). We divide replacement paths whose detour start in xy into two types:

$\mathcal{R}_1(xy)$: Preferred replacement paths from x to t whose detour starts in xy but the avoided edge lies in yt_x.

$\mathcal{R}_2(xy)$: Preferred replacement paths from x to t whose start of detour and avoided edge both lie strictly inside segment xy (that is, detour cannot start from x or y).

Let $\mathcal{R}_1 := \cup_{xy \in \sigma\text{-BFS}(t)} \mathcal{R}_1(xy)$ and $\mathcal{R}_2 := \cup_{xy \in \sigma\text{-BFS}(t)} \mathcal{R}_2(xy)$. The set $\mathcal{R}_1$ helps us to weed out simple preferred replacement paths. We will show that we can store preferred

replacement paths in $\mathcal{R}_1$ in $O(\sigma)$ space – one per segment in σ-BFS(t). The hardest case for us in $\mathcal{R}_2$, which contains bad paths. Let $\mathcal{B}$ denote the set of bad paths in $\mathcal{R}_2$. We will show that $|\mathcal{B}| \leq |\mathcal{R}_2 \setminus \mathcal{B}|$ (the number of bad paths is $\leq$ number of good paths in $\mathcal{R}_2$) and $|\mathcal{R}_2 \setminus \mathcal{B}| = O(\sqrt{n\sigma})$ (the number of good path is $O(\sqrt{n\sigma})$). This implies that $|\mathcal{R}_2| = O(\sqrt{n\sigma})$.

Since $\mathcal{R}_1$ and $\mathcal{R}_2$ are of size $O(\sqrt{n\sigma})$, we can make a data-structure of size $O(\sqrt{n\sigma})$. In this data-structure, we have stored a preferred path for each segment. However, we have to answer queries of type $\mathrm{Q}(s, t, e)$ where s is a source. In Section 9, we will see how to use preferred paths of segments to answer queries in $\tilde{O}(1)$ time.

7 Analysing preferred replacement paths in $\mathcal{R}_1$

We first show the following:

▶ **Lemma 16.** *For each segment $xy \in \sigma$-BFS(t), $|\mathcal{R}_1(xy)| = 1$*

The above lemma implies that $|\mathcal{R}_1| = \cup_{xy \in \sigma\text{-BFS}(t)} |\mathcal{R}_1(xy)| = O(\sigma) = O(\sqrt{n\sigma})$.

8 Analysing preferred replacement paths in $\mathcal{R}_2$

We first show that one special kind of path will never lie in $\mathcal{R}_2$. This characterization will help in analyzing bad paths in $\mathcal{R}_2$.

▶ **Lemma 17.** *Let P be a preferred path from x to t avoiding e on xt path. If P merges with any segment $x'y'$ and then diverges from $x't$ path, then $P \notin \mathcal{R}_2$.*

We will now analyze paths in $\mathcal{R}_2$. Consider two replacement paths P, P' avoiding edges e, e' (respectively) on $xy, x'y'$ segment respectively. Let a, a' be the starting vertex of $\textsc{Detour}(P), \textsc{Detour}(P')$ respectively. We say that $P \prec P'$ if $|at| < |a't|$. If $|at| = |a't|$, then the tie is broken arbitrarily.

Given a path $P \in \mathcal{R}_2$, let $(< P)$ be the set of all replacement paths in $\mathcal{R}_2$ that are $\prec P$ in the ordering. Similarly, $(> P)$ is the set of all replacement paths $P' \in \mathcal{R}_2$ for which $P \prec P'$. Define $\textsc{Unique}(P)$ according to this ordering (see definition 10). Assume that we are processing a replacement path P according to this ordering. If $|\textsc{Unique}(P)| \geq \sqrt{n/\sigma}$, then we can associate $O(\sqrt{n/\sigma})$ *unique* vertices to P. Otherwise $|\textsc{Unique}(P)| < \sqrt{n/\sigma}$ and we have the following two cases:

8.1 Detour(P) does not intersect with any other detour in $(> P)$

This case is similar to the first case in Section 5.1. We can show that once we get a replacement path $P \in \mathcal{R}_2(xy)$ with $|\textsc{Unique}(P)| < \sqrt{n/\sigma}$, then there are at most $O(\sqrt{n/\sigma})$ replacement paths in $\mathcal{R}_2(xy)$ remaining to be processed. This will bound the total number of such paths to $O(\sqrt{n\sigma})$. Please see the full version for details.

8.2 Detour(P) intersects with detour of a path in $(> P)$

We first give a formal definition of a bad path that was defined informally in Section 6.

▶ **Definition 18.** (Bad Path) A path $P \in \mathcal{R}_2$ is called a bad path if there exists another path $P' \in (> P)$ such that (1) $\textsc{Detour}(P)$ intersects with $\textsc{Detour}(P')$ and (2) $\textsc{Detour}(P')$ passes through the edge avoided by P after their intersection. We also say that P is a bad replacement path due to P' if P' satisfies the above two conditions.

A path that is not bad is called a good path. In Section 6, we saw that bad paths break the easy analysis of the single source case. So, we have two cases depending on whether the path is good or bad. Let us look at the easier case first.

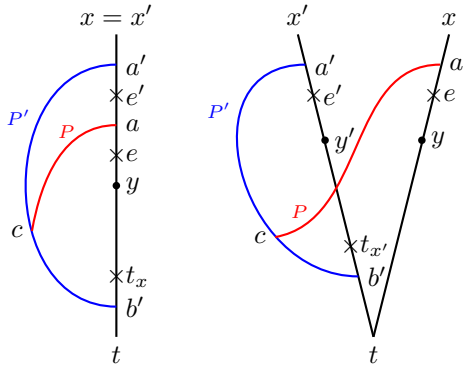


Figure 4 The figure shows two representative examples when ca and at does not pass through e'.

(1) *P is a good path.*
Assume that $P \in \mathcal{R}_2(xy)$ and it avoids an edge $e \in xy$. Assume that P intersects first with $P' \in (> P)$ and P' avoids e' on $x'y'$ segment. Note that x may be equal to x'. Let $\textsc{Detour}(P')$ start at a' and end at b'. Assume that $\textsc{Detour}(P)$ starts at a and it intersects $\textsc{Detour}(P')$ at c. Consider the path $x'a' \diamond a'c \diamond ca \diamond at$. Since $x'a' \diamond a'c$ is a part of P', it avoids e'. However, it is not clear whether $ca \diamond at$ avoids e' too. In Figure 4, we see two representative examples in which ca and at avoid e'.

In the full version of the paper, we show that ca and at cannot pass through e'. Thus, the path $x'a' \diamond a'c \diamond ca \diamond at$ is indeed a valid replacement path from x' to t avoiding e'. Since $P' = x'a' \diamond a'c \diamond cb' \diamond b't$, length of P' must be $\leq$ length of this alternate path. Thus,

$$\begin{aligned} & |x'a'| + |a'c| + |cb'| + |b't| \leq |x'a'| + |a'c| + |ca| + |at| \\ \implies & |cb'| + |b't| \leq |ca| + |at| \\ \implies & |ac| + |cb'| + |b't| \leq 2|ca| + |at| \end{aligned}$$

On the left hand of the inequality, we have a path from a to t avoiding e (since we know that P is a good path, so P' and thus $cb' \diamond b't$ does not pass through e). So, its length should be $\geq$ length of the preferred path $P \setminus xa$. Thus $|P \setminus xa| \leq 2|ca| + |at| \leq 2\sqrt{n/\sigma} + |at|$ (since $|\textsc{Unique}(P)| = |ac| < \sqrt{n/\sigma}$). Consider the following set of replacement paths $(< P)_x := \{P' \in (< P) \mid P' \text{ avoids an edge on } xy \text{ segment}\}$. By Lemma 6, all replacement paths in $(< P)_x$ pass through e (as detour of these replacement paths start below e) and by Lemma 7, these replacement paths avoid edges that are closer to y than e. Applying Lemma 9, we get that the number of replacement paths $(< P)_x$ is $\leq 2\sqrt{n/\sigma}$. Thus, once we get a replacement path $P \in \mathcal{R}_2(xy)$ with $|\textsc{Unique}(P)| < \sqrt{n/\sigma}$, then there are at most $2\sqrt{n/\sigma}$ replacement paths in $\mathcal{R}_2(xy)$ remaining to be processed. Thus, total number of paths $\in \mathcal{R}_2$ with $|\textsc{Unique}(P)| < \sqrt{n/\sigma}$ is $\sum_{xy \in \sigma\text{-BFS}(t)} 2\sqrt{n/\sigma} = O(\sqrt{n\sigma})$ (as there are $O(\sigma)$ segments in σ-BFS(t)).
(2) *P is a bad path.*
We now arrive at our hardest scenario. We will first show that the number of good paths in $\mathcal{R}_2$ is greater than the number of bad paths in $\mathcal{R}_2$. To this end, we will prove the following lemma:

▶ **Lemma 19.** *For each $P' \in \mathcal{R}_2$, there exists only one replacement path $P \in \mathcal{R}_2$ which is bad due to P'.*

The above lemma can be used to discard bad paths from $\mathcal{R}_2$. For each such discarded path, there exists at least one good path. And by the above lemma, each such good path can be used to discard at most one bad path. Thus the number of good paths in $\mathcal{R}_2$ is $\geq$

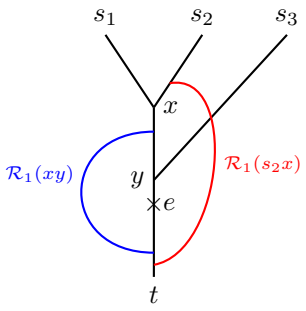


Figure 5 The shortest path from s_2 to t avoiding e can be $\mathcal{R}_1(xy)$ or $\mathcal{R}_1(s_2x)$.

number of bad paths in $\mathcal{R}_2$. We have already shown that the total number of good paths in $\mathcal{R}_2$ is $O(\sqrt{n\sigma})$. Thus the total number of paths in $\mathcal{R}_2$ is also $O(\sqrt{n\sigma})$.

9 Building the Data Structure

Let us first recognize a potential problem in using $\mathcal{R}_1(\cdot)$. Let s_1t and s_2t path meet at vertex x (See Figure 5). Another path s_3t meets s_2t path at y where y is closer to t. $\mathcal{R}_1(s_2x)$ is the shortest path from s_1 to t avoiding e and $\mathcal{R}_1(xy)$ is the shortest path from x to t avoiding $e \in yt$. This immediately leads to the following problem. Assume that the query is $\mathrm{Q}(s_2, t, e)$ and the preferred path avoiding e is in $\mathcal{R}_1$. Then there are two candidate paths that avoid e: one that goes from s_2 to the intersection vertex x and then take path $\mathcal{R}_1(xy)$ and the other $\mathcal{R}_1(s_2x)$. Thus, we need to check these two paths and return the minimum of the two. One can make a bigger example in which there are σ segments between s_2 and t and thus we have to check $O(\sigma)$ path before we can answer the query. The problem appears because we don't know from which segment the shortest path avoiding e started its detour. If this information is not there, then it seems that we have to look at all the segments between s_2 ans t. To end this dilemma, we use heavy light decomposition of σ-BFS(t) [20]. For any segment $xy \in \sigma$-BFS(t) (by our convention y is closer to t), x is a *heavy* child of y if the number of nodes in the subtree under x is $\geq 1/2$(number of nodes in the subtree under y) else it is called a *light child* (or light *segment* in our case). It follows that each intersection vertex has exactly one heavy child and each vertex is adjacent to atmost two heavy edges. A *heavy chain* is a concatenation of heavy edges. A *heavy subpath* is a subpath of a heavy chain. The following lemma notes a well known property of heavy-light decomposition.

▶ **Lemma 20.** *The path from a source s to t in σ-BFS(t) can be decomposed into $O(\log n)$ heavy subpaths and light segments .*

Given any source $s \in S$, by Lemma 20, the path from t to s may contain many heavy subpaths. Let $C(pq)$ be a heavy chain that starts at p and ends at q (where q is closer to t than p). A ts path may follow a heavy chain $C(pq)$ but may exit this chain from a vertex midway, say at r. Let $(C(pq), r)$ be a tuple associated with s such that the shortest path from t to s enters this heavy chain via q and leaves this chain at r. We keep a list $\textsc{Heavy}(s, t)$ which contains all the tuples $(C(pq), r)$ sorted according to the distance of heavy chain from t (that is distance qt). By Lemma 20, the size of $\textsc{Heavy}(s, t) = O(\log n)$. Similarly, we have one more list to store the light segments. $\textsc{Light}(s, t)$ contains all the light segments on the st path again ordered according to their distance from t in σ-BFS(t). Again by Lemma 20, the size of $\textsc{Light}(s, t) = O(\log n)$. Note that the size of these additional two data-structures is $\sum_{s\in S} O(\log n) = \tilde{O}(\sigma) = \tilde{O}(\sqrt{n\sigma})$.

Our main problem was that we have to find the minimum $\mathcal{R}_1(\cdot)$ of $O(\sigma)$ segments if there is a path of length σ between s and t. The trick we use here is that finding minimum on any heavy subpath takes $\tilde{O}(1)$ time. Since there are $O(\log n)$ heavy subpaths, the total time taken to find the minimum on heavy subpaths in $\tilde{O}(1)$. Also, since the number of light segments is also $O(\log n)$ finding the minimum among these also takes $\tilde{O}(1)$ time.

We now describe our intuition in detail. Let xy be a segment in a heavy chain $C(pq)$. We want to represent $\mathcal{R}_1(xy)$ in a balanced binary search tree $\textsc{Bst}(C)$. To this end, we will add a node with the tuple $(x.depth, |px \diamond \mathcal{R}_1(xy)|, |px|)$ in $\textsc{Bst}(C)$. The first element in this tuple is the depth of x in $\text{BFS}(t)$ – it also acts as the key in this binary search tree. The second element is the path $\mathcal{R}_1(xy)$ concatenated with px. This concatenation is done so that all paths in $\textsc{Bst}(C)$ start from p and comparing two paths in $\textsc{Bst}(C)$ is possible. The third element will be used to get the path length $\mathcal{R}_1(xy)$ (by subtracting it from the second element) when need arises. Now we can augment this tree so that the following range minimum query can be answered in $\tilde{O}(1)$ time: $\textsc{Rmq}(C(pq), [a, b])$: Find minimum of $\{|px \diamond \mathcal{R}_1(xy)| \mid xy$ is a segment in heavy chain $C(pq)$ and $x.depth \geq a.depth$ and $x.depth \leq b.depth\}$. The size of $\cup_{C \in \textsc{Heavy}(s,t)} \textsc{Bst}(C)$ is $O(\sigma) = O(\sqrt{n\sigma})$ as there are at most $O(\sigma)$ segments in σ-$\text{BFS}(t)$.

Given any edge $e(u, v)$ on st path, we can now find the shortest path in $\mathcal{R}_1$ from s to t avoiding e (Please refer to Algorithm 1 in the full version of the paper). We first find the first intersection vertex on the us path from u. Let this vertex be x. We will see that finding x is also not a trivial problem – we will say more about this problem later. Now, we will go over all possible replacement paths from u to s. Thus, we search if there exists any heavy chain in $\textsc{Heavy}(s, t)$ that contains x. To this end, we first check if x lies in some light segment (this can be checked in $\tilde{O}(1)$ time). If not, then x lies in some heavy chain. We now search each heavy chain in $\textsc{Heavy}(s, t)$ to find a node x' with the smallest depth such that $x'.depth > x.depth$. Let this node be x'. Thus we have found the segment $x'x$ where x is closer to t than x'. We can easily calculate $x.depth$ as $|st| - |sx|$ or $B_0(s, t) - B_0(s, x)$. Since there are $\tilde{O}(1)$ heavy chain in $\textsc{Heavy}(s, t)$, the time taken to find if $x'x$ exists in some heavy chain is $\tilde{O}(1)$.

Assume that we found out that $x'x \in C(pq)$, and ts path leaves the chain C at r, then we want to find the shortest replacement path from r to t avoiding e. This can be found out via the range minimum query $\textsc{Rmq}(C(p, q), [x, r])$. However, note that each replacement path in C starts from p. So, we need to remove $|pr|$ from the replacement path length returned by $\textsc{Rmq}$ query. The length pr can be found out in the node $r \in \textsc{Bst}(C)$. Finally, we add $|sr|$ to get the path from s to t.

Similarly, we can process a light segment in $O(1)$ time (please refer to Algorithm 1 in the full version). Thus, the time taken by Algorithm 1 is $\tilde{O}(1)$ as the while loop runs at most $O(\log n)$ times and each step in the while loop runs in $\tilde{O}(1)$ time.

9.1 Answering queries in $\tilde{O}(1)$ time

Given a query $Q(s, t, e(u, v))$, we process it as follows (assuming that e lies on st_s path (that is the *far case)* and v is closer to t than u)

1. Find the first intersection vertex on us path.
   In the full version of the paper, we show that we can find the first intersection vertex on us path in $\tilde{O}(1)$ time using $O(\sqrt{n\sigma})$ space.
2. Find the replacement path avoiding u if it lies in $\mathcal{R}_1$.
   To this end, we use our Algorithm 1. The first non-trivial part of this algorithm, that is, finding the first intersection vertex on the us path has already been tackled in the point

above. So we can find such a replacement path (if it exists) in $\tilde{O}(1)$ time and $\tilde{O}(\sqrt{n\sigma})$ space.

3. Find the replacement path avoiding $e(u, v)$ if it lies in $\mathcal{R}_2$.
   This part is similar to our data-structure in single source case. Let $x \leftarrow \textsc{Int}(u, t)$. Using $\textsc{Heavy}(s, t)$ and $\textsc{Light}(s, t)$, in $\tilde{O}(1)$ time, we can find the segment $xy \in \sigma\text{-BFS}(t)$ such that y is closer to t than x. In this case, we want to check if there exists any replacement path that starts in the same segment in which e resides. This replacement path first takes sx path and then takes the detour strictly inside the segment xy. All such paths are stored in $\mathcal{R}_2(xy)$ with the contiguous range of edges that they avoid on xy. We now just need to check if u and v lie in the range of some replacement path. To this end, we find $u.depth \leftarrow |st| - |su| = B_0(s, t) - B_0(s, u)$ and $v.depth \leftarrow |st| - |sv| = B_0(s, t) - B_0(s, v)$. Now we check if $u.depth$ and $v.depth$ lie in contiguous range of some replacement path in $\mathcal{R}_2(xy)$. If yes, then we return the length of that path concatenated with sx. Note that we have already stored $|sx|$ in $B_0(s, x)$. The time taken in this case is dominated by searching u and v in $\mathcal{R}_2(xy)$, that is $\tilde{O}(1)$.

Thus, the total query time of our algorithm is $\tilde{O}(1)$, and we can return the minimum of replacement paths found in Step 2 and 3 as our final answer. The reader can check that the space taken by our algorithm for a vertex t is $\tilde{O}(\sqrt{n\sigma})$. Thus the total space taken by our algorithm is $\tilde{O}(\sigma^{1/2}n^{3/2})$. Thus we have proved the main result, that is Theorem 1 of our paper.

References

1 Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. Dynamic DFS in Undirected Graphs: breaking the O(m) barrier. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 730–739, 2016.

2 Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant subgraph for single source reachability: generic and optimal. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 509–518, 2016.

3 Aaron Bernstein. A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 742–755. Society for Industrial and Applied Mathematics, 2010.

4 Aaron Bernstein and David Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 101–110. ACM, 2009.

5 Davide Bilò, Keerti Choudhary, Luciano Gualà, Stefano Leucci, Merav Parter, and Guido Proietti. Efficient oracles and routing schemes for replacement paths. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 13:1–13:15, 2018.

6 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Compact and fast sensitivity oracles for single-source distances. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 13:1–13:14, 2016.

7 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 18:1–18:14, 2016.

8 Greg Bodwin, Fabrizio Grandoni, Merav Parter, and Virginia Vassilevska Williams. Preserving distances in very faulty graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 73:1–73:14, 2017.

9 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.

10 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 506–515, 2009.

11 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 748–757. IEEE, 2012.

12 Manoj Gupta and Shahbaz Khan. Multiple source dual fault tolerant BFS trees. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 127:1–127:15, 2017.

13 John Hershberger and Subhash Suri. Vickrey prices and shortest paths: What is an edge worth? In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 252–259, 2001.

14 Neelesh Khanna and Surender Baswana. Approximate shortest paths avoiding a failed vertex: Optimal size data structures for unweighted graphs. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, pages 513–524, 2010.

15 Kavindra Malik, Ashok K Mittal, and Santosh K Gupta. The k most vital arcs in the shortest path problem. *Operations Research Letters*, 8(4):223–227, 1989.

16 Merav Parter. Dual failure resilient BFS structure. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 481–490, 2015.

17 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 779–790, 2013.

18 Liam Roditty. On the k-simple shortest paths problem in weighted directed graphs. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 920–928. Society for Industrial and Applied Mathematics, 2007.

19 Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Transactions on Algorithms (TALG)*, 8(4):33, 2012.

20 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

21 Oren Weimann and Raphael Yuster. Replacement paths via fast matrix multiplication. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 655–662. IEEE, 2010.

22 Virginia Vassilevska Williams. Faster replacement paths. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1337–1346. SIAM, 2011.

An Exponential Separation Between MA and AM Proofs of Proximity*

Tom Gur[1]
UC Berkeley, Berkeley, USA
tom.gur@berkeley.edu

Yang P. Liu
MIT, Cambridge, MA
yangpatil@gmail.com

Ron D. Rothblum[2]
MIT and Northeastern University, Cambridge, MA
ronr@mit.edu

Abstract

Interactive proofs of proximity allow a sublinear-time verifier to check that a given input is *close* to the language, using a small amount of communication with a powerful (but untrusted) prover. In this work we consider two natural *minimally interactive* variants of such proofs systems, in which the prover only sends a single message, referred to as the proof.

The first variant, known as MA-proofs of Proximity (MAP), is fully *non-interactive*, meaning that the proof is a function of the input *only*. The second variant, known as AM-proofs of Proximity (AMP), allows the proof to additionally depend on the verifier's (entire) random string. The complexity of both MAPs and AMPs is the total number of bits that the verifier observes – namely, the sum of the proof length and query complexity.

Our main result is an exponential separation between the power of MAPs and AMPs. Specifically, we exhibit an explicit and natural property Π that admits an AMP with complexity $O(\log n)$, whereas any MAP for Π has complexity $\tilde{\Omega}(n^{1/4})$, where n denotes the length of the input in bits. Our MAP lower bound also yields an alternate proof, which is more general and arguably much simpler, for a recent result of Fischer *et al.* (ITCS, 2014).

Lastly, we also consider the notion of *oblivious* proofs of proximity, in which the verifier's queries are oblivious to the proof. In this setting we show that AMPs can only be quadratically stronger than MAPs. As an application of this result, we show an exponential separation between the power of public and private coin for oblivious interactive proofs of proximity.

2012 ACM Subject Classification Theory of computation → Interactive computation

Keywords and phrases Property testing, Probabilistic proof systems, Proofs of proximity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.73

Related Version A full version of the paper is available at [31], `https://eccc.weizmann.ac.il/report/2018/083/`.

Acknowledgements We thank Oded Goldreich and Justin Thaler for very helpful discussions.

* Extended abstract. See full version [31].

[1] Research supported in part by the UC Berkeley Center for Long-Term Cybersecurity.
[2] Research supported in part by NSF Grants CNS-1350619 and CNS-1414119, Alfred P. Sloan Research Fellowship, Microsoft Faculty Fellowship and in part by the Defense Advanced Research Projects Agency (DARPA), the U.S. Army Research Office under contracts W911NF-15-C-0226 and W911NF-15-C-0236 and by the Cybersecurity and Privacy Institute at Northeastern University.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 73; pp. 73:1–73:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The field of property testing [47, 23] deals with sublinear algorithms for deciding whether a given object has a predetermined property or is far from any object having this property. Such algorithms, called testers, obtain local views of the object by performing queries; that is, the object is seen as a function and the tester receives oracle access to this function. The goal of the tester is to ascertain a global property of the function based only on its local view.

In the last couple of decades, the area of property testing has attracted much attention (see surveys [44, 45, 11] and recent textbook [22]). However, while much success was found in designing testers for a myriad of natural properties, which only make a small number of queries, many other natural properties were shown to require a very large number of queries to test (often linear in the length of the input).

Proofs of proximity, first considered by Ergün, Kumar and Rubinfeld [18], are both intrinsically interesting as a natural notion of proof systems for sublinear algorithms, as well as provide means to significantly reduce the number of queries that the tester needs to make in order to verify, rather than decide. These probabilistic proof systems can be viewed as augmenting testers with a help from a powerful, yet untrusted prover. In a recent line of works [46, 33, 20, 27, 36, 25, 43, 34, 9, 14] various types of interactive [46] and non-interactive proofs of proximity [35] were studied, including arguments of proximity [36], zero-knowledge proofs of proximity [9], and proofs of proximity for distribution testing [14].

In this work we study the relation between two types of proofs of proximity that are *minimally interactive*; namely, MA and AM proofs of proximity, which can be viewed as the property testing analogue of the class MA (i.e., "randomized NP") and AM, respectively, and are described in more detail next.

Informally speaking, an MA proof of proximity (MAP) protocol consists of a tester (or rather a verifier) that receives oracle access to an input function f but also receives explicit access to a short purported proof w. Based on the proof string and a few oracle queries to f, the verifier should decide whether f has some property Π (i.e., whether $f \in \Pi$). More specifically, after reading the proof w, the verifier tosses random coins, makes queries to the oracle f, and decides whether to accept or reject. We require the following completeness and soundness conditions: if $f \in \Pi$, then there exists a proof w that the verifier accepts with high probability, and if f is "far" (in Hamming distance) from any function in Π, then the verifier rejects with high probability. Following the literature, the complexity of an MAP is the total number of bits that the verifier observes - namely, the sum of its proof length and query complexity.[3]

The reason that the foregoing model is referred to as a "Merlin-Arthur" protocol is that we think of the prover as being Merlin (the all powerful magician) and the verifier as Arthur (a mere mortal). Then in the MAP model Merlin "speaks" first (i.e., sends the proof) and Arthur "speaks" second (i.e., tosses his random coins).

It is natural to ask what happens if we switch the order - letting Arthur toss his coins first and Merlin send his proof after seeing Arthur's coin tosses. This type of protocol is typically referred to as an "Arthur-Merlin" protocol. More precisely, an AM proof of proximity (AMP) is defined similarly to an MAP, except that now the proof oracle is a function of the verifier's entire random string. We view AMPs as *minimally interactive*, since given a common random

[3] Alternatively, one could view the running time of the verifier (which serves as an upper bound on the query and communication complexities) as the main resource to be minimized. However, for simplicity (and following the property testing literature), we focus on combinatorial resources. This only makes our lower bounds stronger, whereas for all of our upper bounds the verifier is also computationally efficient.

string, the protocol consists of a single message (i.e., the proof). Analogously to MAPs, the complexity of an AMP is the sum of its proof length and query complexity. We emphasize that the prover's message can depend on all of the verifier's coin tosses. Namely, the verifier cannot toss an additional coins after receiving its message from the prover.[4]

While the difference between these two proof systems may appear minor, MA-type and AM-type proofs naturally admit very different types of strategies. In particular, note that AM proofs provide the additional power of allowing the prover and verifier to jointly restrict their attention to a random subset of the input function's domain. On the other hand, the AM model also significantly hampers the power of the verifier to detect malicious prover strategies, since the prover knows the entire randomness of the verifier, and in particular the prover knows which queries the verifier will make.

At first glance, it may seem that AMPs are extremely limited, since the prover can predict exactly what the verifier will check (knowing the verifier's *entire* random string allows the prover to deduce which queries the verifier will make). However, it turns out that a straightforward adaptation of the classical MA $\subseteq$ AM inclusion [6] implies that any MAP can be emulated by an AMP at a quadratic cost. (More precisely, an MAP with proof complexity p and query complexity q can be emulated by an AMP with proof length p and query complexity $O(p \cdot q)$.[5])

It is natural to ask the following converse question:

Can any AMP *protocol be emulated by an* MAP*, or is there a gap between the power of these two models?*

Note that any AMP can be easily emulated by an MAP with at most an exponential overhead.[6] Thus, the question that we would really like to answer is whether such an exponential overhead is inherent.

1.1 Our results

Our main result shows that it is indeed the case that AMPs can be *exponentially* stronger than MAPs and so the foregoing emulation strategy is optimal, up to polynomial factors:

▶ **Theorem 1.** *There exists a property* $\Pi \subseteq \{f : [n] \to [n]\}$ *such that:*

- Π *has an* AMP *of complexity* $O(\log(n)/\varepsilon)$*, with respect to proximity parameter* $\varepsilon > 0$*; and*
- *Every* MAP *for* Π*, with respect to proximity parameter* $\varepsilon \leq \frac{1}{10}$*, must have complexity* $\Omega(n^{\frac{1}{4}})$.

The property Π that we use to prove Theorem 1 is actually very simple and natural. Specifically, Π is the set of all permutations over $[n]$; the goal of the verifier is to check

[4] In contrast, the complexity class AM is sometimes defined as any constant-round public-coin interactive proof-system. Indeed, if one does not care about polynomial factors, then by a result of Babai and Moran [6], any public-coin constant-round interactive proof can be reduced to just 2 messages.

[5] The idea is to first reduce the soundness error of the MAP to $2^{-O(p)}$ (by repetition). Now suppose that the verifier reveals its randomness to the prover before receiving the proof-string. For soundness, observe that when f is far from the having the property, for any fixed proof-string the probability that the verifier would accept is at most $2^{-\Omega(p)}$ and so by a union bound, with high probability there simply does not exist a proof-string that will make the verifier accept.

[6] Any AMP with proof complexity p and query complexity q can be emulated by a *tester* (i.e., an MAP which does not use a proof at all) with query complexity $q \cdot 2^p$ by simply trying all possible candidate proof strings.

whether a given function $f : [n] \to [n]$ is close to being a permutation by querying the function in a few locations and with a short interaction with the prover.

The AMP protocol for deciding whether a given function $f : [n] \to [n]$ is a permutation is extremely simple. The idea is that the random string specifies some random element $y \in [n]$ and the prover should specify an inverse x of y (under f). If f is a permutation such an element must exist whereas if f is ε-far from being a permutation, then with probability ε it holds that y simply does not have an inverse. We can repeat the base protocol $O(1/\varepsilon)$ times to get constant soundness error. This protocol can actually be traced back to a result of Bellare and Yung [7] who used it resolve a gap in the [19] construction of non-interactive zero-knowledge proofs for NP based on trapdoor permutations.

Our MAP lower bound is the technically more challenging part of this work, and is actually a special case of a more general MAP lower bound that we prove. We show that any property that satisfies a relaxed notion of k-wise independence requires MAPs with complexity roughly $\sqrt{k}$. This result generalizes a recent result of Fischer, Goldhirsh and Lachish [20] which can be interpreted as an MAP lower bound of $\sqrt{k}$ for properties that are exactly k-wise independent.[7] Our proof is also (arguably) significantly simpler than that of [20] and in particular uses only elementary arguments, see further discussion in Section 1.2.

1.1.1 Oblivious Proofs of Proximity

Having established Theorem 1, we revisit the MA versus AM problem within the context of proofs of proximity *with prover-oblivious queries* (a notion first considered in [46] and further explored in [35]), or in short *oblivious proofs of proximity*. These are proofs of proximity that have the special feature that the queries that the verifier makes are independent of the proof. Viewed from a temporal perspective, in these proof systems the verifier *first* makes its queries to the input function, and only after making all of its queries does it receive the proof. One reason that makes this feature appealing is because it allows the verifier to probe the object and obtain a certificate, which can then be used later when interacting with a prover, even if the object is no longer accessible. Another reason is that many of the interactive proof systems from the literature (e.g., the sumcheck protocol of [39]) are *oblivious*.

Surprisingly, it turns out that the gap between the power of *oblivious* AMPs and MAPs is dramatically smaller than the one exhibited in Theorem 1. Loosely speaking, we show that oblivious AMPs can only be *quadratically* stronger than oblivious MAPs, and in fact, standard testers (that do not use a proof).

▶ **Theorem 2.** *For any property* Π*, if there exists an* oblivious AMP *for* Π *with proof complexity* p *and query complexity* q*, then there also exists a tester (i.e.,* MAP *with proof complexity* 0*) for* Π *with query complexity* $O(p \cdot q)$*.*

As an application, we use Theorem 2 to derive lower bounds on public-coin oblivious interactive proofs of proximity, and show an exponential separation between public-coin and private-coin protocols in this setting. See further discussion in the full version [31].

1.2 Related works

The notion of proofs of proximity was originally proposed by Ergün, Kumar and Rubinfeld. Ben Sasson *et al.* [8] and Dinur and Reingold [17] considered such proofs in the context of

[7] More precisely, [20] show that any linear code with large dual distance requires MAPs of complexity that is roughly square root of the code's blocklength.

PCPs. Rothblum, Vadhan and Wigderson [46], considered *interactive* proofs of proximity and showed that every language computable by a low-depth circuit has an interactive proof of proximity (IPP) with a sublinear time verifier. Reingold, Rothblum, and Rothblum [43] showed *constant-round* IPPs for any language computable in polynomial-time and bounded polynomial-space. Goldreich and Gur [24, 25] showed general-purpose IPP with only 3 rounds, albeit for a much smaller class.

Proofs of proximity were further studied by [27] who showed more efficient constructions for certain restricted complexity classes, such as functions accepted by small read-once branching programs and context-free languages. Gur and Rothblum proved a round hierarchy theorem for IPPs [34], showing that the power of IPPs gradually increases with the number of rounds of interaction. Several works focused on studying non-interactive (MA) proofs of proximity [35, 20, 26] (see also [30]). In addition, recent works studied (computationally sound) interactive *arguments* of proximity [36], zero-knowledge proofs of proximity [9], and proofs of proximity for distribution testing [14]. Proofs of proximity have also found applications to property testing and related models [33, 21, 34]. We remark that a concurrent work of Berman *et al.* [9], utilizes our results (specifically Theorem 1) to derive a separation between the power of MAPs and zero-knowledge IPPs.

The notion of MA and AM proofs plays a central role in the study of proofs system in various computational models, other than in the setting of polynomial-time Turing machines in which they were originally conceived [6]. For example, in quantum computation, the class QMA (quantum MA proofs) captures the most fundamental type of quantum proof systems (since quantum algorithms are inherently randomized) and it has been extensively studied in the last couple of decades (see survey [4]). Of particular relevance, Aaronson [1] considered the problem of deciding whether a function is close to a permutation to derive a quantum query complexity separation between the class QMA and the class of statistical zero knowledge SZK, showing that every QMA query complexity algorithm with a w-qubit witness and query complexity q must satisfy $q + w = \Omega(n^{1/6})$.

In addition, MA and AM proof systems received much attention in the setting of communication complexity [5, 37, 41, 38, 29, 48] and streaming algorithms [15, 16, 12, 32, 13, 49]. The former also has an interesting connection to the algebrization barrier [2] and recently found important applications to distributed PCPs and hardness of approximation [3].[8] The latter can be viewed as the property testing analogue of online annotated data streams (there, instead of oracle access to the input, the algorithm has one-pass sequential access to the input, and the goal is to minimize *space* complexity rather than *query* complexity). Indeed, part of our results concerning oblivious proofs of proximity are inspired by the techniques for online annotated data streams in [13].

Perhaps most relevant to us, the notion of MA and AM proofs for decision tree complexity (or the "query complexity model"), which can be thought of as property testing for *exact* (rather than approximate) decision problems, is closely related to proofs of proximity, though the query complexity model is much simpler to analyze than property testing. We remark that the high-level approach of our main lower bound for MAPs is inspired by the work of Raz *et al.* [42].

[8] We remark that there are several similarities between MA and AM proof systems in the setting of property testing and communication complexity. In particular, simulating MA communication complexity protocols by their AM counterparts can also be done while only incurring a quadratic blow-up in complexity, and on the other hand AM protocols can also be exponentially more powerful than MA protocols [38]. In addition, oblivious MA proofs of proximity can be viewed as analogous to *online* MA communication complexity protocols [13].

Comparison with the techniques in [20]. As we discussed above, our MAP lower bound generalizes the main result of Fischer, Goldhirsh, and Lachish [20]. The latter result can be interpreted as an MAP lower bound for any k-wise independent property. Our lower bound extends to a natural generalization of this family. We stress that this extension is *crucial* for our main result, as the permutation property (with respect to which we prove Theorem 1) is *not* k-wise independent, but does satisfy our more general notion.[9]

The proof in [20] is technically quite involved and includes several subtle and non-trivial arguments. For example, while typically property testing lower bounds are shown by exhibiting two distributions that are chosen only as a function of the property, the argument in [20] crucially relies on distributions that are functions of both the property and the description of the specific analyzed algorithm. This entails the usage of several complex mechanisms. For example, they rely on an involved treatment of adaptivity, which consists of procedures for "grafting" decision trees, and use a special type of algorithms (called "readers") that expose low-entropy portions. Perhaps the most significant complication is that their argument uses a delicate information theoretic analysis to handle MAPs that have a two-sided error.

In contrast, our proof is much shorter and consists purely of a combinatorial argument, which does not require any special treatment of adaptivity and two-sided error, and does not use information theory.

1.3 Organization

In Section 2, we introduce the notations and definitions that we use throughout this work. In Section 3, we prove our main technical contribution, which is an MAP lower bound for relaxed k-wise independent properties. Finally, in Section 4, we conclude with a discussion and raise open problems. See full version [31] for the proof of our main result: an exponential separation between MAPs and AMPs, as well as for our results regarding oblivious proofs of proximity.

2 Preliminaries

In this section we establish the definitions and notions that we will need throughout this work.

2.1 Properties and Distance

We focus on testing properties of *functions* and identify a "property" with the set of functions having that property. More accurately, for each $n \in \mathbb{N}$, let D_n and R_n be sets. Let $\mathcal{F}_n$ be the set of functions from D_n to R_n. We define a property as an ensemble $\Pi = \bigcup_n \Pi_n$, where $\Pi_n \subseteq \mathcal{F}_n$ for all n.

For an alphabet Σ, we denote the Hamming distance between two strings $x, y \in \Sigma^n$ by $\Delta(x,y) := |\{x_i \neq y_i : i \in [n]\}|$. If $\Delta(x,y) \leq \varepsilon \cdot n$, we say that x is ε-close to y, otherwise we say that x is ε-far from y. For a non-empty set $S \subseteq \Sigma^n$, we similarly define $\Delta(x,S) := \min_{y \in S} \Delta(x,y)$. Again, if $\Delta(x,S) \leq \varepsilon \cdot n$, we also say that x is ε-close to S and otherwise x is ε-far from S. We extend these definitions to functions by identifying functions with their truth tables (viewed as strings).

[9] Jumping ahead, we remark that our relaxed notion of k-wise independence refers to distributions for which the probability that any subset of k indices is equal to any given sequence of k values is upper bounded by the same probability given the uniform distribution *up to a multiplicative constant* (whereas the standard (i.e., non-relaxed) notion requires exact equality). See further details in Section 3.

Integrality. Throughout this work, for simplicity of notation, we use the convention that all (relevant) integer parameters that are stated as real numbers are implicitly rounded to the closest integer.

2.2 Proofs of Proximity

We recall the definitions of MA and AM proofs of proximity (i.e., MAPs and AMPs), following [35]. Throughout, for an algorithm V we denote by $V^f(n, \varepsilon, w)$ the output of V given oracle access to a function f and explicit access to inputs n, ε, and w; if V is a probabilistic algorithm, we write $\Pr[V^f(n, \varepsilon, w) = z]$ to represent the probability *over the internal randomness of* V that this outcome is z.

▶ **Definition 3** (MAP). A Merlin-Arthur proof of proximity (MAP) for a property $\Pi = \bigcup_n \Pi_n$ consists of a probabilistic algorithm V, called the verifier, that is given as explicit inputs an integer $n \in \mathbb{N}$, a proximity parameter $\varepsilon > 0$, and a proof string $w \in \{0,1\}^*$; in addition, it is given oracle access to a function $f \in \mathcal{F}_n$. The verifier satisfies the following conditions.

1. **Completeness:** For every $n \in \mathbb{N}$ and $f \in \Pi_n$, there exists a string w (the proof) such that for every $\varepsilon > 0$ the verifier accepts with high probability; that is,

$$\Pr\left[V^f(n, \varepsilon, w) = 1\right] \geq \frac{2}{3}.$$

2. **Soundness:** For every $n \in \mathbb{N}$, function $f \in \mathcal{F}_n$, string w, and proximity parameter $\varepsilon > 0$, if f is ε-far from Π_n, then the verifier rejects with high probability; that is,

$$\Pr\left[V^f(n, \varepsilon, w) = 0\right] \geq \frac{2}{3}.$$

A MAP is said to have query complexity $q : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{N}$ if for every $n \in \mathbb{N}, \varepsilon > 0$, $f \in \mathcal{F}_n$, and string $w \in \{0,1\}^*$, the verifier reads at most $q(n, \varepsilon)$ bits in its queries to f. We say that a MAP has proof complexity $p : \mathbb{N} \to \mathbb{N}$ if for every $n \in \mathbb{N}$, there always exists a $w \in \{0,1\}^{p(n)}$ satisfying the conditions of Definition 3. We define the complexity of the MAP to be $t(n, \varepsilon) = q(n, \varepsilon) + p(n)$.

Next, we define AM proofs of proximity (AMPs) similarly to MAPs, except that here the proof is also a function of the inner randomness of the verifier (alternatively, the verifier first sends the prover its entire random string).

▶ **Definition 4** (AMP). An Arthur-Merlin proof of proximity (AMP) for a property $\Pi = \bigcup_n \Pi_n$ consists of a probabilistic algorithm V, called the verifier, that is given as explicit inputs an integer $n \in \mathbb{N}$, a proximity parameter $\varepsilon > 0$, and a proof string w that *depends on the verifier's random string* r, as well as oracle access to a function $f \in \mathcal{F}_n$. The verifier must also be deterministic given the random string r. The protocol satisfies the following conditions.

1. **Completeness:** For every $n \in \mathbb{N}$ and $f \in \Pi_n$,

$$\Pr_r\left[\exists w = w(r) \text{ such that } V^f(n, \varepsilon, w; r) = 1\right] \geq \frac{2}{3}.$$

2. **Soundness:** For every $n \in \mathbb{N}$, function $f \in \mathcal{F}_n$, and proximity parameter $\varepsilon > 0$, if f is ε-far from Π_n, then:

$$\Pr_r\left[\exists w \text{ such that } V^f(n, \varepsilon, w; r) = 1\right] \leq \frac{1}{3}.$$

Analogously to MAPs, an AMP is said to have *query complexity* $q : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{N}$ if for every $n \in \mathbb{N}, \varepsilon > 0$, $f \in \mathcal{F}_n$, and string $w \in \{0,1\}^*$, the verifier reads at most $q(n,\varepsilon)$ bits in its queries to f; and *proof complexity* $p : \mathbb{N} \times \mathbb{R}^+ \to \mathbb{N}$ if for every $n \in \mathbb{N}$ and $f \in \mathcal{F}_n$, with probability at least $\frac{2}{3}$ over coin tosses in the first round, there exists a $w \in \{0,1\}^{p(n,\varepsilon)}$ satisfying the completeness condition of Theorem 4. We define the complexity of the AMP to be $t(n,\varepsilon) = q(n,\varepsilon) + p(n,\varepsilon)$.

We note that we do not include the randomness complexity of the verifier in the complexity of the protocol (although the randomness complexity in all the protocols described in this work is not large). This is a similar choice to what is done in similar contexts such as AM query and communication complexities. Moreover, we show in the full version [31] that if a property Π of functions $f : D_n \to R_n$, such that $|R_n|^{|D_n|} = O(\exp(\mathrm{poly}(n)))$, admits an AMP verifier with query complexity q and proof complexity p, then it also admits an AMP verifier with query complexity $O(q)$, proof complexity $O(p)$, and randomness complexity $O(\log n)$.[10] This transformation is similar to known results of Newman[40] in the context of communication complexity, Goldreich and Sheffet [28] in the context of property testing, and Gur and Rothblum [35] for MAPs. Its main disadvantage however is that it does not preserve the *computational* complexity of the verifier.

3 MAP Lower Bound for (Relaxed) k-wise Independence

In this section we show a general MAP lower bound for a large class of properties. More specifically, we show that any MAP for a (non-degenerate) property that is k-wise independent, must have complexity $\Omega(\sqrt{k})$. By a k-wise independent property we mean that if we sample a random element having the property, than its restriction to any k coordinates looks uniform. As mentioned in the introduction, this generalizes a result due to Fischer *et al.* [20].

We would like to apply this lower bound to the permutation property. However, the permutation property is not k-wise independent and so we cannot apply it directly.[11]. Rather, we give a relaxed notion of k-wise independence that does capture the permutation property and for which we can similarly derive an MAP lower bound.

We proceed to define our relaxed notion of k-wise independence. Recall that we use $\mathcal{F}_n$ to denote the set of all functions from D_n to R_n (see Section 2).

▶ **Definition 5** (Relaxed k-wise Independence)**.** Let $\Pi = \bigcup_{n\geq 1} \Pi_n$ be a property, where $\Pi_n \subset \mathcal{F}_n$ for every n. We say that Π is *relaxed k-wise independent*, for $k = k(n)$, if there exists a constant $C \geq 1$ such that for all positive integers n, all pairwise distinct k-tuples $(i_1, i_2, \ldots, i_k) \in (D_n)^k$ and arbitrary $(t_1, t_2, \ldots, t_k) \in (R_n)^k$, we have that

$$\Pr_{f \in \Pi_n}\left[f(i_j) = t_j \text{ for all } j \in [k]\right] \leq \frac{C}{|R_n|^k}. \tag{1}$$

Note that standard definition of a k-wise independence corresponds to the special case of Definition 5 when $C = 1$ (in which case the inequality in Eq. (1) can be replaced with an equality).

At first glance it may seem that the relaxation that we allow in Definition 5 is relatively minor and any lower bound that holds for the full-fledged definition should easily be extendable to our relaxed variant. We argue that it is not the case. For example, in a seminal

[10] For most properties, we have that both the domain and range have size that is polynomial in n. Indeed, the case that $|R_n|^{|D_n|} = \omega(\exp(\mathrm{poly}(n)))$ seems quite pathological.

[11] Indeed, it is not even pairwise independent: the chance of seeing the same element twice is *zero*.

work, Braverman [10] showed that any k-wise independent distribution (for k that is poly-logarithmic) fools AC_0 circuits. Now consider the permutation property (defined formally in the full version [31]) which as noted above is not even pairwise independent but does satisfy our relaxed variant (with $k = \sqrt{n}$). It is not too hard to see that there is a very simple AC_0 circuit for checking whether a function is a permutation: simply by checking whether there exist a pair of entries in the truth table that are identical - thus, our seemingly minor relaxation completely sidesteps Braverman's result. As a matter of fact, a similar situation occurs in the context of AMPs: Rothblum *et al.* [46] showed an AMP lower bound for exact k-wise independent distribution, whereas we show a protocol for the permutation property with logarithmic complexity.

Having defined our notion of relaxed k-wise independence, we proceed to describe a second important condition that we require: namely, that the property is *sparse*, in the sense that a random function is far from the property. Sparsity is essential for our result since there are trivial properties that are k-wise independent but are testable with very few queries (e.g., the property that consists of all functions).

▶ **Definition 6** (Sparse Property)**.** Fix the proximity parameter $\varepsilon = \frac{1}{10}$. We say that a property $\Pi_n = \bigcup_{n\in\mathbb{N}} \Pi_n$ is $t(n)$-*sparse* if:

$$\Pr_{f\in\mathcal{F}_n}[f \text{ is } \varepsilon\text{-far from } \Pi_n] \geq 1 - |R_n|^{-t(n)}.$$

We can now state our main theorem for this section.

▶ **Theorem 7.** *Let Π be a relaxed k-wise independent and k-sparse property. Then, any* MAP *for Π, with respect to proximity parameter $\varepsilon = 1/10$, with proof complexity p and query complexity q must satisfy $p \cdot q = \Omega(k)$.*

The intuition and high level approach for the proof are as follows. First, we use the duality of an MAP as a collection of partial testers [20]. More specifically, the existence of an MAP for a property Π implies that there is some large "sub-property" $\Pi' \subseteq \Pi$ and a *tester* T that distinguishes between inputs in Π' from those that are far from Π.

This simple observation reduces lower bounding MAPs for Π to lower bounding a partial tester for an arbitrary, but large, sub-property. To show such a lower bound, consider the uniform distribution on Π' vs. the uniform distribution over functions that are far from Π. We would like to argue that these two distributions look the same to T, which therefore cannot distinguish between them.

As a matter of fact, we will argue that both these distributions are "close" to being k-wise independent, which suffices as long as k is larger than the tester's query complexity. First, by the sparsity condition we have that the uniform distribution over functions that are far from Π is close to the uniform distribution over *all* functions. Clearly the latter is k-wise independent.

As for the uniform distribution over Π', we would like to argue that since Π' covers a substantial part of Π, which is relaxed k-wise independent, then also Π' is relaxed k-wise independent. The problem with this argument is that Π' only consists of a 2^{-p} fraction of Π, and so it could be quite far from being even relaxed k-wise independent (e.g., it could be that the value of functions in Π' on some fixed elements of R_n is constant over all functions in Π').

This seems like a significant difficulty and was overcome using highly elaborate techniques in [20]. In contrast, we suggest a much simpler argument. The idea is that we first reduce the soundness error of the MAP to $2^{-O(p)}$ by repetition. This increases the query complexity

of the tester to $O(p \cdot q)$ but now that the soundness error is so small, that the fact that Π' covers a 2^{-p} fraction of Π is sufficient to make the argument go through.

We proceed to the actual proof.

3.1 Proof of Theorem 7

Let C be a constant such that Π_n satisfies the constraints of Definition 5.

Let V be an MAP verifier, with respect to proximity parameter ε, for Π_n, and denote its proof complexity by p and query complexity by q. Note that any MAP with standard 2/3 completeness and soundness probability (as in Definition 3) can be amplified, via $O(p)$ repetitions, to have completeness and soundness errors $\frac{1}{10C} \cdot 2^{-p}$ at the cost of increasing the query (but not the proof) complexity by a multiplicative factor of $O(p)$, to $O(p \cdot q)$. For concreteness, let us fix a constant C' such that a $(C' \cdot p)$-fold repetition of V has completeness and soundness errors $\frac{1}{10C} \cdot 2^{-p}$ (while having proof complexity p and query complexity $C' \cdot p \cdot q$). Assume towards a contradiction that $p \cdot q \leq \frac{k}{10C'}$.

Recall that for $\Pi' \subseteq \Pi$, a (Π, Π')-partial tester (a notion due to [20]) is a tester that is required to accept functions in the subset Π' and reject functions that are ε-far from the superset Π. As pointed out by Fischer *et al.* [20] an MAP as we assumed above, implies a covering of the property by partial testers as follows. For every possible proof string $w \in \{0,1\}^p$, let

$$S_w = \left\{ f \in \Pi_n \ : \ \Pr\left[V^f(n, \varepsilon, w) = 1\right] \geq 1 - \frac{1}{10C} \cdot 2^{-p} \right\}.$$

By the completeness requirement of an MAP, these sets cover the property Π_n. That is, $\bigcup_w S_w = \Pi_n$.

Since the number of sets S_w is at most 2^p, there exists a proof w that corresponds to a large S_w. Namely, such that $|S_w| \geq |\Pi_n| \cdot 2^{-p}$. We fix such a proof w and argue that the corresponding (Π_n, S_w)-partial tester must make $\Omega(k)$ queries, which would contradict our assumption, thereby proving Theorem 7. Hence, we have reduced proving an MAP lower bound for Π_n to proving a partial testing lower bound for (Π_n, S_w).

Let $V_w^f(n, \varepsilon) \coloneqq V^f(n, \varepsilon, w)$ be the (S_w, Π_n)-partial tester that is induced by V when we fix the proof string w (and with respect to parameters n and ε). We use the notation $V_w^f(n, \varepsilon; r)$ to denote the *deterministic* output V_w^f when its random string is set to r.

Let $B_\varepsilon = \{f \in \mathcal{F}_n : f \text{ is } \varepsilon\text{-far from } \Pi_n\}$ (i.e., the no-instances). As standard in the property testing literature, we prove a lower bound on the query complexity q' of a tester by presenting a distribution over YES-instances ($f \in S_w$) and a distribution over NO-instances ($f \in B_\varepsilon$) and bounding away from 1 the distinguishing probability for every *deterministic* algorithm making q' queries. Specifically, we give distributions over S_w and B_ε such that any deterministic algorithm making q' queries to f has at most a $1 - \frac{1}{4C} \cdot 2^{-p}$ probability of distinguishing between them, which is sufficient for our purposes. In our case, we simply consider the uniform distributions over S_w and B_ε.

More formally, we first observe that

$$\begin{aligned} &\mathbb{E}_{f \in S_w}\left[\Pr_r\left[V_w^f(n, \varepsilon; r) = 1\right]\right] - \mathbb{E}_{f \in B_\varepsilon}\left[\Pr_r\left[V_w^f(n, \varepsilon; r) = 1\right]\right] \\ &= \mathbb{E}_r\left[\Pr_{f \in S_w}\left[V_w^f(n, \varepsilon; r) = 1\right] - \Pr_{f \in B_\varepsilon}\left[V_w^f(n, \varepsilon; r) = 1\right]\right], \end{aligned} \tag{2}$$

By Eq. (2), it suffices to bound the distinguishing probably for any deterministic verifier. We do this via the following lemma.

▶ **Lemma 8.** *For any deterministic verifier W with query complexity at most $\frac{k}{10}$, we have that*

$$\Pr_{f\in S_w}\left[W^f(n,\varepsilon)=1\right]-\Pr_{f\in B_\varepsilon}\left[W^f(n,\varepsilon)=1\right]\leq 1-\frac{1}{4C}\cdot 2^{-p}.$$

Proof. We first show that

$$\Pr_{f\in B_\varepsilon}\left[W^f(n,\varepsilon)=1\right]\geq\frac{1}{2C}\cdot 2^{-p}\cdot\Pr_{f\in S_w}\left[W^f(n,\varepsilon)=1\right]. \tag{3}$$

We can view the verifier W as a decision tree of depth $q'=k/10$. Each leaf of the decision tree is associated with indices $i_1,i_2,\ldots,i_{q'}\in D_n$ and values $t_1,t_2,\ldots,t_{q'}\in R_n$ such that a function $f\in\mathcal{F}_n$ is accepted at that leaf if and only if $f(i_j)=t_j$ for all $j\in[q']$. We may assume without loss of generality that the sets of indices $i_1,\ldots,i_{q'}$ for all paths in the decision tree are pairwise distinct. Fix such a sequence of indices $i_1,\ldots,i_{q'}\in D_n$ and values $t_1,\ldots,t_{q'}\in R_n$. Then,

$$\begin{aligned}\Pr_{f\in B_\varepsilon}\left[f(i_j)=t_j\text{ for all }j\in[q']\right]&\geq\frac{\left|\left\{f\in\mathcal{F}_n:f(i_j)=t_j\text{ for all }j\in[q']\right\}\right|-\left|\mathcal{F}_n\backslash B_\varepsilon\right|}{|B_\varepsilon|}\\&\geq\frac{1}{|R_n|^{q'}}-\frac{1}{|R_n|^k-1}\\&\geq\frac{1}{2|R_n|^{q'}}.\end{aligned} \tag{4}$$

Here we have used k-sparsity to note that $\frac{|\mathcal{F}_n\backslash B_\varepsilon|}{|B_\varepsilon|}\leq\frac{1}{|R_n|^k-1}$, and we used that $q'=\frac{k}{10}$. On the other hand, we also have that:

$$\Pr_{f\in S_w}\left[f(i_j)=t_j\text{ for all }j\in[q']\right]\leq\frac{\Pr_{f\in\Pi_n}\left[f(i_j)=t_j\text{ for all }j\in[q']\right]}{\Pr_{f\in\Pi_n}[f\in S_w]}\leq\frac{C\cdot 2^p}{|R_n|^{q'}} \tag{5}$$

by relaxed q'-wise independence and the lower bound on the size of S_w.

Dividing Eq. (4) by Eq. (5), we obtain that

$$\Pr_{f\in B_\varepsilon}\left[f(i_j)=t_j\text{ for all }j\in[q']\right]\geq\frac{1}{2C}\cdot 2^{-p}\cdot\Pr_{f\in S_w}\left[f(i_j)=t_j\text{ for all }j\in[q']\right].$$

Now, summing the above equation over all leaves of the decision tree corresponding to W (since these correspond to disjoint events) gives us Eq. (3).

Given Eq. (3), we now consider two cases. First, if

$$\Pr_{f\in S_w}\left[W^f(n,\varepsilon)=1\right]\leq 1-\frac{1}{2C}\cdot 2^{-p}$$

we are obviously done. Otherwise, we can assume that $\Pr_{f\in S_w}\left[W^f(n,\varepsilon)=1\right]>1-\frac{1}{2C}\cdot 2^{-p}$ and so:

$$\begin{aligned}\Pr_{f\in S_w}\left[W^f(n,\varepsilon)=1\right]-\Pr_{f\in B_\varepsilon}\left[W^f(n,\varepsilon)=1\right]&\leq 1-\frac{1}{2C}\cdot 2^{-p}\cdot\Pr_{f\in S_w}\left[W^f(n,\varepsilon)=1\right]\\&\leq 1-\frac{1}{2C}\cdot 2^{-p}\cdot\left(1-\frac{1}{2C}\cdot 2^{-p}\right)\\&\leq 1-\frac{1}{4C}\cdot 2^{-p},\end{aligned}$$

where the first inequality is by Eq. (3). The lemma follows. ◀

Now we are ready to use Lemma 8 to complete our proof of Theorem 7. Because V_w^f has completeness and soundness errors $\frac{1}{10C} \cdot 2^{-p}$, we have that

$$\mathbb{E}_{f \in S_w}\left[\Pr_r\left[V_w^f(n,\varepsilon;r)=1\right]\right] - \mathbb{E}_{f \in B_\varepsilon}\left[\Pr_r\left[V_w^f(n,\varepsilon;r)=1\right]\right] \geq 1 - \frac{1}{10C}\cdot 2^{-p} - \frac{1}{10C}\cdot 2^{-p} = 1 - \frac{1}{5C}\cdot 2^{-p}.$$

On the other hand, by Eq. (2) and Lemma 8, it holds that

$$\mathbb{E}_{f \in S_w}\left[\Pr_r\left[V_w^f(n,\varepsilon;r)=1\right]\right] - \mathbb{E}_{f \in B_\varepsilon}\left[\Pr_r\left[V_w^f(n,\varepsilon;r)=1\right]\right] \leq 1 - \frac{1}{4C}\cdot 2^{-p},$$

which is a contradiction. Therefore, we can conclude that $p \cdot q \geq \frac{k}{10C'}$, as desired.

4 Discussion and Open Problems

The complexity of the permutation property for testers, which do not use a proof, is $\tilde{\Theta}(\sqrt{n})$. In this work we showed a lower bound of $\tilde{\Omega}(n^{\frac{1}{4}})$ for MAPs for Perm. Thus, the MAP complexity of Perm is somewhere between $\tilde{\Omega}(n^{\frac{1}{4}})$ and $\tilde{O}(\sqrt{n})$ - resolving the exact complexity is an interesting open problem:

▶ **Problem 9.** *Does every* MAP *for* Permutation *have complexity* $\tilde{\Omega}(\sqrt{n})$*?*

Second, our work shows that AMPs can be exponentially more efficient than MAPs. It is natural to ask whether the converse also holds - can MAPs be much more efficient than AMPs? A partial answer to this question is known. As mentioned in Footnote 5, every MAP with complexity c can be emulated by an AMP with complexity (roughly) c^2.

Thus, MAPs can be at most *quadratically* more efficient than AMPs. However, we do not know a property for which this gap is tight. In particular, the following problem is open:

▶ **Problem 10.** *Does this exist a property* Π *that has an* MAP *with complexity* $O(\sqrt{n})$ *but every* AMP *for* Π *must have complexity* $\Omega(n)$*?*

References

1. Scott Aaronson. Impossibility of succinct quantum proofs for collision-freeness. *Quantum Information & Computation*, 12(1-2):21–28, 2012. URL: http://www.rintonpress.com/xxqic12/qic-12-12/0021-0028.pdf.
2. Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *TOCT*, 1(1):2:1–2:54, 2009. doi:10.1145/1490270.1490272.
3. Amir Abboud, Aviad Rubinstein, and R. Ryan Williams. Distributed PCP theorems for hardness of approximation in P. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 25–36, 2017. doi:10.1109/FOCS.2017.12.
4. Dorit Aharonov and Tomer Naveh. Quantum np-a survey. *arXiv preprint quant-ph/0210077*, 2002.
5. László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory. In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 337–347, 1986. doi:10.1109/SFCS.1986.15.
6. László Babai and Shlomo Moran. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.*, 36(2):254–276, 1988. doi:10.1016/0022-0000(88)90028-1.

7 Mihir Bellare and Moti Yung. Certifying permutations: Noninteractive zero-knowledge based on any trapdoor permutation. *J. Cryptology*, 9(3):149–166, 1996.

8 Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust pcps of proximity, shorter pcps, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006. doi:10.1137/S0097539705446810.

9 Itay Berman, Ron D. Rothblum, and Vinod Vaikuntanathan. Zero-knowledge proofs of proximity. *IACR Cryptology ePrint Archive*, 2017:114, 2017. URL: http://eprint.iacr.org/2017/114.

10 Mark Braverman. Poly-logarithmic independence fools bounded-depth boolean circuits. *Commun. ACM*, 54(4):108–115, 2011. doi:10.1145/1924421.1924446.

11 Clément L. Canonne. A survey on distribution testing: Your data is big. but is it blue? *Electronic Colloquium on Computational Complexity (ECCC)*, 22:63, 2015. URL: http://eccc.hpi-web.de/report/2015/063.

12 Amit Chakrabarti, Graham Cormode, Andrew McGregor, and Justin Thaler. Annotations in data streams. *ACM Trans. Algorithms*, 11(1):7:1–7:30, 2014. doi:10.1145/2636924.

13 Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. Verifiable stream computation and arthur-merlin communication. In *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*, pages 217–243, 2015. doi:10.4230/LIPIcs.CCC.2015.217.

14 Alessandro Chiesa and Tom Gur. Proofs of proximity for distribution testing. *ECCC*, 24:155, 2017. URL: https://eccc.weizmann.ac.il/report/2017/155.

15 Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. In *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part I*, pages 231–242, 2010. doi:10.1007/978-3-642-15775-2_20.

16 Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *PVLDB*, 5(1):25–36, 2011. URL: http://www.vldb.org/pvldb/vol5/p025_grahamcormode_vldb2012.pdf.

17 Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. *SIAM J. Comput.*, 36(4):975–1024, 2006. doi:10.1137/S0097539705446962.

18 Funda Ergün, Ravi Kumar, and Ronitt Rubinfeld. Fast approximate probabilistically checkable proofs. *Inf. Comput.*, 189(2):135–159, 2004. doi:10.1016/j.ic.2003.09.005.

19 Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *IAM J. Comput.*, 29(1), 1999. Preliminary version in *FOCS'90*. doi:10.1137/S0097539792230010.

20 Eldar Fischer, Yonatan Goldhirsh, and Oded Lachish. Partial tests, universal tests and decomposability. In *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 483–500, 2014. doi:10.1145/2554797.2554841.

21 Eldar Fischer, Oded Lachish, and Yadu Vasudev. Trading query complexity for sample-based testing and multi-testing scalability. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1163–1182, 2015. doi:10.1109/FOCS.2015.75.

22 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.

23 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

24 Oded Goldreich and Tom Gur. Universal locally testable codes. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:42, 2016. URL: http://eccc.hpi-web.de/report/2016/042.

25 Oded Goldreich and Tom Gur. Universal locally verifiable codes and 3-round interactive proofs of proximity for CSP. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:192, 2016. URL: http://eccc.hpi-web.de/report/2016/192.

26 Oded Goldreich, Tom Gur, and Ilan Komargodski. Strong locally testable codes with relaxed local decoders. In *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*, pages 1–41, 2015. doi:10.4230/LIPIcs.CCC.2015.1.

27 Oded Goldreich, Tom Gur, and Ron D. Rothblum. Proofs of proximity for context-free languages and read-once branching programs - (extended abstract). In *International Colloquium on Automata, Languages and Programming ICALP*, 2015. doi:10.1007/978-3-662-47672-7_54.

28 Oded Goldreich and Or Sheffet. On the randomness complexity of property testing. *Computational Complexity*, 19(1):99–133, 2010. doi:10.1007/s00037-009-0282-4.

29 Mika Göös, Toniann Pitassi, and Thomas Watson. Zero-information protocols and unambiguity in arthur-merlin communication. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 113–122, 2015. doi:10.1145/2688073.2688074.

30 Tom Gur. *On Locally Verifiable Proofs of Proximity*. PhD thesis, Weizmann Institute, 2017.

31 Tom Gur, Yang P. Liu, and Ron D. Rothblum. An exponential separation between ma and am proofs of proximity, 2018. URL: https://eccc.weizmann.ac.il/report/2018/083/.

32 Tom Gur and Ran Raz. Arthur-merlin streaming complexity. *Inf. Comput.*, 243:145–165, 2015. doi:10.1016/j.ic.2014.12.011.

33 Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. *Computational Complexity*, June 2016.

34 Tom Gur and Ron D. Rothblum. A hierarchy theorem for interactive proofs of proximity. In *Innovations in Theoretical Computer Science ITCS*, 2017.

35 Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. *Computational Complexity*, 27(1):99–207, 2018.

36 Yael Tauman Kalai and Ron D. Rothblum. Arguments of proximity - [extended abstract]. In *CRYPTO*, 2015. doi:10.1007/978-3-662-48000-7_21.

37 Hartmut Klauck. Rectangle size bounds and threshold covers in communication complexity. In *18th Annual IEEE Conference on Computational Complexity (Complexity 2003), 7-10 July 2003, Aarhus, Denmark*, pages 118–134, 2003. doi:10.1109/CCC.2003.1214415.

38 Hartmut Klauck. On arthur merlin games in communication complexity. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC 2011, San Jose, California, June 8-10, 2011*, pages 189–199, 2011. doi:10.1109/CCC.2011.33.

39 Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992. doi:10.1145/146585.146605.

40 Ilan Newman. Private vs. common random bits in communication complexity. *Inf. Process. Lett.*, 39(2):67–71, 1991. doi:10.1016/0020-0190(91)90157-D.

41 Ran Raz and Amir Shpilka. On the power of quantum proofs. In *19th Annual IEEE Conference on Computational Complexity (CCC 2004), 21-24 June 2004, Amherst, MA, USA*, pages 260–274, 2004. doi:10.1109/CCC.2004.1313849.

42 Ran Raz, Gábor Tardos, Oleg Verbitsky, and Nikolai K. Vereshchagin. Arthur-merlin games in boolean decision trees. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity, Buffalo, New York, USA, June 15-18, 1998*, pages 58–67, 1998. doi:10.1109/CCC.1998.694591.

43 Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th Annual ACM SIGACT Sym-*

posium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016, pages 49–62, 2016. doi:10.1145/2897518.2897652.

44 Dana Ron. Property testing: A learning theory perspective. *Foundations and Trends in Machine Learning*, 1(3):307–402, 2008. doi:10.1561/2200000004.

45 Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2009. doi:10.1561/0400000029.

46 Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In *Symposium on Theory of Computing, STOC*, 2013. doi:10.1145/2488608.2488709.

47 Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996. doi:10.1137/S0097539793255151.

48 Alexander A. Sherstov. The multiparty communication complexity of set disjointness. *SIAM J. Comput.*, 45(4):1450–1489, 2016. doi:10.1137/120891587.

49 Justin Thaler. Semi-streaming algorithms for annotated graph streams. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 59:1–59:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.59.

Isolating a Vertex via Lattices: Polytopes with Totally Unimodular Faces

Rohit Gurjar[1]
California Institute of Technology, USA

Thomas Thierauf[2]
Aalen University, Germany

Nisheeth K. Vishnoi
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Abstract

We present a geometric approach towards derandomizing the Isolation Lemma by Mulmuley, Vazirani, and Vazirani. In particular, our approach produces a quasi-polynomial family of weights, where each weight is an integer and quasi-polynomially bounded, that can isolate a vertex in any 0/1 polytope for which each face lies in an affine space defined by a totally unimodular matrix. This includes the polytopes given by totally unimodular constraints and generalizes the recent derandomization of the Isolation Lemma for bipartite perfect matching and matroid intersection. We prove our result by associating a lattice to each face of the polytope and showing that if there is a totally unimodular kernel matrix for this lattice, then the number of vectors of length within 3/2 of the shortest vector in it is polynomially bounded. The proof of this latter geometric fact is combinatorial and follows from a polynomial bound on the number of circuits of size within 3/2 of the shortest circuit in a regular matroid. This is the technical core of the paper and relies on a variant of Seymour's decomposition theorem for regular matroids. It generalizes an influential result by Karger on the number of minimum cuts in a graph to regular matroids.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization, Mathematics of computing → Matroids and greedoids, Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Derandomization, Isolation Lemma, Total unimodularity, Near-shortest vectors in Lattices, Regular matroids

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.74

Related Version A full version of the paper is available at [11], `https://arxiv.org/abs/1708.02222`.

1 Introduction

The Isolation Lemma by Mulmuley, Vazirani, and Vazirani [14] states that for any given family of subsets of a ground set E, if we assign random weights (bounded in magnitude by poly($|E|$)) to the elements of E then, with high probability, the minimum weight set in the family is unique. Such a weight assignment is called an *isolating weight assignment*. The lemma was introduced in the context of randomized parallel algorithms for the matching problem. Since

[1] The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 257575 and from the Israel Science Foundation (grant number 552/16).
[2] Supported by DFG grant TH 472/4.

then it has found numerous other applications, in both algorithms and complexity: e.g., a reduction from CLIQUE to UNIQUE-CLIQUE [14], NL/poly $\subseteq \oplus$L/poly [29], NL/poly = UL/poly [17], an RNC-algorithm for linear matroid intersection [15], and an RP-algorithm for disjoint paths [3]. In all of these results, the Isolation Lemma is the only place where they need randomness. Thus, if the Isolation Lemma can be derandomized, i.e., if a polynomially bounded isolating weight assignment can be deterministically constructed, then the aforementioned results that rely on it can also be derandomized. In particular, it will give a deterministic parallel algorithm for matching.

A simple counting argument shows that a single weight assignment with polynomially bounded weights cannot be isolating for all possible families of subsets of E. We can relax the question and ask if we can construct a poly-size list of poly-bounded weight assignments such that for each family $\mathcal{B} \subseteq 2^E$, one of the weight assignments in the list is isolating. Unfortunately, even this can be shown to be impossible via arguments involving the polynomial identity testing (PIT) problem. The PIT problem asks if an implicitly given multivariate polynomial is identically zero. Derandomization of PIT is another important consequence of derandomizing the Isolation Lemma. Here, the Isolation Lemma is applied to the family of monomials present in the polynomial. In essence, if we have a small list of weight assignments that works for all families, then we will have a small hitting-set for all small degree polynomials, which is impossible (see [2]). Once we know that a deterministic isolation is not possible for all families, a natural question is to solve the isolation question for families $\mathcal{B}$, that have a succinct representation, for example, the family of perfect matchings of a graph.

For the general setting of families with succinct representations, no deterministic isolation is known, other than the trivial construction with exponentially large weights. In fact, derandomizing the isolation lemma in this setting will imply circuit lower bounds [2]. Efficient deterministic isolation is known only for very special kinds of families, for example, perfect matchings in some special classes of graphs [1, 5, 6, 9], s-t paths in directed graphs [4, 12, 28]. Recently, there has been significant progress on deterministic isolation for perfect matchings in bipartite graphs [7] and subsequently, in general graphs [25], and matroid intersection [10], which implied quasi-NC algorithms for these problems.

Motivated by these recent works, we give a generic approach towards derandomizing the Isolation Lemma. We show that the approach works for a large class of combinatorial polytopes and conjecture that it works for a significantly larger class. For a family of sets $\mathcal{B} \subseteq 2^E$, define the polytope $P(\mathcal{B}) \subseteq \mathbb{R}^E$ to be the convex hull of the indicator vectors of the sets in $\mathcal{B}$. Our main result shows that for $m := |E|$, there exists an $m^{O(\log m)}$-sized family of weight assignments on E with weights bounded by $m^{O(\log m)}$ that is isolating for any family $\mathcal{B}$ whose corresponding polytope $P(\mathcal{B})$ satisfies the following property: *the affine space spanned by any face of $P(\mathcal{B})$ is parallel to the null space of* **some** *totally unimodular (TU) matrix*; see Theorem 2.3. This is a black-box weight construction in the sense that it does not need the description of the family or the polytope.

A large variety of polytopes satisfy this property and, as a consequence, have been extensively studied in combinatorial optimization. The simplest such class is when the polytope $P(\mathcal{B})$ has a description $Ax \leq b$ with A being a TU matrix. Thus, a simple consequence of our main result is a resolution to the problem of derandomizing the isolation lemma for polytopes with TU constraints, as raised in a recent work [25]. This generalizes the isolation result for perfect matchings in a bipartite graph [7], since the perfect matching polytope of a bipartite graph can be described by the incidence matrix of the graph, which is TU. Other examples of families whose polytopes are defined by TU constraints are vertex

covers of a bipartite graph, independent sets of a bipartite graph, and, edge covers of a bipartite graph. Note that these three problems are computationally equivalent to bipartite matching and thus, already have quasi-NC algorithms due to [7]. However, the isolation results for these families are not directly implied by isolation for bipartite matchings.

Our work also generalizes the isolation result for the family of common bases of two matroids [10]. In the matroid intersection problem, the constraints of the common base polytope are a rank bound on every subset of the ground set. These constraints, in general, do not form a TUM. However, for every face of the polytope there exist two laminar families of subsets that form a basis for the tight constraints of the face. The incidence matrix for the union of two laminar families is TU (see [20, Theorem 41.11]).

Since our condition on the polytope $P(\mathcal{B})$ does not require the constraint matrix defining the polytope itself (or any of its faces) to be TU, it is quite weak and is also well studied. Schrijver [19, Theorem 5.35] shows that this condition is sufficient to prove that the polytope is *box-totally dual integral*. The second volume of Schrijver's book [20] gives an excellent overview of polytopes that satisfy the condition required in Theorem 2.3 such as

- $R - S$ bibranching polytope [20, Section 54.6]
- directed cut cover polytope [20, Section 55.2]
- submodular flow polyhedron [20, Theorem 60.1]
- lattice polyhedron [20, Theorem 60.4]
- submodular base polytope [20, Section 44.3]
- many other polytopes defined via submodular and supermodular set functions [20, Sections 46.1, 48.1, 48.23, 46.13, 46.28, 46.29, 49.3, 49.12, 49.33, 49.39, 49.53].

We would like to point out that it is not clear if our isolation results in the above settings lead to any new derandomization of algorithms. Finding such algorithmic applications of our isolation result would be quite interesting.

To derandomize the Isolation Lemma, we abstract out ideas from the bipartite matching and matroid intersection isolation [7, 10], and give a geometric approach in terms of certain *lattices* associated to polytopes. For each face F of $P(\mathcal{B})$, we consider the lattice L_F of all integer vectors parallel to F. We show that, if for each face F of $P(\mathcal{B})$, the number of near-shortest vectors in L_F is polynomially bounded then we can construct an isolating weight assignment for $\mathcal{B}$ with quasi-polynomially bounded weights; see Theorem 2.4. Our main technical contribution is to give a polynomial bound on the number of near-shortest vectors in L_F (whose ℓ_1-norm is less than $3/2$ times the smallest ℓ_1-norm of any vector in L_F), when this lattice is the set of integral vectors in the null space of a TUM; see Theorem 2.5.

The above lattice result is in contrast to general lattices where the number of such near-shortest vectors could be exponential in the dimension.

Our result on lattices can be reformulated using the language of matroid theory: the number of near-shortest circuits in a regular matroid is polynomially bounded; see Theorem 2.6. In fact, we show how Theorem 2.5 can be deduced from Theorem 2.6. One crucial ingredient in the proof of Theorem 2.6 is Seymour's remarkable decomposition theorem for regular matroids [21]. Theorem 2.6 answers a question raised by Subramanian [24] and is a generalization of (and builds on) known results in the case of graphic and cographic matroids, that is, the number of near-minimum length cycles in a graph is polynomially bounded (see [24, 26]) and the result of Karger [13] that states that the number of near-mincuts in a graph is polynomially bounded.

Thus, not only do our results make progress in derandomizing the isolation lemma for combinatorial polytopes, they make interesting connections between lattices (that are geometric objects) and combinatorial polytopes. Our structural results about the number of

near-shortest vectors in lattices and near-shortest circuits in matroids should be of independent interest and raise the question: to what extent are they generalizable?

A natural conjecture would be that for any $(0,1)$-matrix, the lattice formed by its integral null vectors has a small number of near-shortest vectors. In turn, this would give us the isolation result for any polytope which is defined by a $(0,1)$-constraint matrix. Many combinatorial polytopes have this property. One such interesting example is the perfect matchings polytope for general graphs. The recent result of [25], which showed a quasi-NC algorithm for perfect matchings, does not actually go via a bound on the number of near-shortest vectors in the associated lattice. Obtaining a polynomial bound on this number would give a proof for their quasi-NC result in our unified framework and with improved parameters. Another possible generalization is for $(0,1)$-polytopes that have this property that the integers occurring in the description of each supporting hyperplane are bounded by a polynomial in the dimension of the polytope. Such polytopes generalize almost all combinatorial polytopes and yet seem to have enough structure – they have been recently studied in the context of optimization [22, 23].

2 Our Results

2.1 Isolating a vertex in a polytope

For a set E and a weight function $w\colon E \to \mathbb{Z}$, we define the extension of w to any set $S \subseteq E$ by $w(S) := \sum_{e\in S} w(e)$. Let $\mathcal{B} \subseteq 2^E$ be a family of subsets of E. A weight function $w\colon E \to \mathbb{Z}$ is called *isolating for* $\mathcal{B}$ if the minimum weight set in $\mathcal{B}$ is unique. In other words, the set $\arg\min_{S\in\mathcal{B}} w(S)$ is unique. The Isolation Lemma of Mulmuley, Vazirani, and Vazirani [14] asserts that a uniformly random weight function is isolating with a good probability for any $\mathcal{B}$.

▶ **Lemma 2.1** (Isolation Lemma). *Let E be a set, $|E| = m$, and let $w\colon E \to \{1, 2, \ldots, 2m\}$ be a random weight function, where for each $e \in E$, the weight $w(e)$ is chosen uniformly and independently at random. Then for any family $\mathcal{B} \subseteq 2^E$, w is isolating with probability at least $1/2$.*

The task of derandomizing the Isolation Lemma requires the deterministic construction of an isolating weight function with weights polynomially bounded in $m = |E|$. Here, we view the isolation question for $\mathcal{B}$ as an isolation over a corresponding polytope $P(\mathcal{B})$, as follows. For a set $S \subseteq E$, its indicator vector $x^S := (x^S_e)_{e\in E}$ is defined as $x^S_e = 1$ if $e \in S$ and $x^S_e = 0$ otherwise. For any family of sets $\mathcal{B} \subseteq 2^E$, the polytope $P(\mathcal{B}) \subseteq \mathbb{R}^m$ is defined as the convex hull of the indicator vectors of the sets in $\mathcal{B}$, i.e., $P(\mathcal{B}) := \operatorname{conv}\left\{ x^S \mid S \in \mathcal{B} \right\}$. Note that $P(\mathcal{B})$ is contained in the m-dimensional unit hypercube.

The isolation question for a family $\mathcal{B}$ is equivalent to constructing a weight vector $w \in \mathbb{Z}^E$ such that $\langle w, x\rangle$ has a unique minimum over $P(\mathcal{B})$. The property we need for our isolation approach is in terms of total unimodularity of a matrix.

▶ **Definition 2.2** (Totally unimodular matrix). A matrix $A \in \mathbb{R}^{n\times m}$ is said to be *totally unimodular (TU)*, if every square submatrix has determinant 0 or ± 1.

Our main theorem gives an efficient quasi-polynomial isolation for a family $\mathcal{B}$ when each face of the polytope $P(\mathcal{B})$ lies in the affine space defined by a TU matrix.

▶ **Theorem 2.3** (Main Result). *Let E be a set with $|E| = m$. Consider a class $\mathcal{C}$ of families $\mathcal{B} \subseteq 2^E$ that have the following property: for any face F of the polytope $P(\mathcal{B})$, there exists*

a TU matrix $A_F \in \mathbb{R}^{n\times m}$ such that the affine space spanned by F is given by $A_F x = b_F$ for some $b_F \in \mathbb{R}^n$. We can construct a set W of $m^{O(\log m)}$ weight assignments on E with weights bounded by $m^{O(\log m)}$ such that for any family $\mathcal{B}$ in the class $\mathcal{C}$, one of the weight assignments in W is isolating.

2.2 Short vectors in lattices associated to polytopes

Our starting point towards proving Theorem 2.3 is a reformulation of the isolation approach for bipartite perfect matching and matroid intersection [7, 10]. For a set E and a family $\mathcal{B} \subseteq 2^E$, we define a lattice corresponding to each face of the polytope $P(\mathcal{B})$. The isolation approach works when this lattice has a small number of near-shortest vectors. For any face F of $P(\mathcal{B})$, consider the lattice of all integral vectors parallel to F,

$$L_F := \left\{ v \in \mathbb{Z}^E \mid v = \alpha(x_1 - x_2) \text{ for some } x_1, x_2 \in F \text{ and } \alpha \in \mathbb{R} \right\}.$$

Let $\lambda(L) := \min\{ \|v\| \mid 0 \neq v \in L \}$ denote the length of the shortest nonzero vector of a lattice L, where $\|\cdot\|$ denotes the ℓ_1-norm. We prove that if, for all faces F of $P(\mathcal{B})$ the number of near-shortest vectors in L_F is small, then we can efficiently isolate a vertex in $P(\mathcal{B})$.

▶ **Theorem 2.4** (Isolation via Lattices). *Let E be a set with $|E| = m$ and let $\mathcal{B} \subseteq 2^E$ be a family such that there exists a constant $c > 1$, such that for any face F of polytope $P(\mathcal{B})$, we have $|\{ v \in L_F \mid \|v\| < c\,\lambda(L_F) \}| \leq m^{O(1)}$. Then one can construct a set of $m^{O(\log m)}$ weight functions with weights bounded by $m^{O(\log m)}$ such that at least one of them is isolating for $\mathcal{B}$.*

The main ingredient of the proof of Theorem 2.3 is to show that the hypothesis of Theorem 2.4 is true when the lattice L_F is the set of all integral vectors in the nullspace of a TU matrix. For any $n \times m$ matrix A we define a lattice:

$$L(A) := \{ v \in \mathbb{Z}^m \mid Av = 0 \}.$$

▶ **Theorem 2.5** (Near-shortest vectors in TU lattices). *For an $n \times m$ TU matrix A, let $\lambda := \lambda(L(A))$. Then $|\{ v \in L(A) \mid \|v\| < 3/2\,\lambda \}| = O(m^5)$.*

A similar statement can also be shown with any ℓ_p-norm for $p \geq 2$, but with an appropriate multiplicative constant. Theorem 2.5 together with Theorem 2.4 implies Theorem 2.3.

Proof of Theorem 2.3. Let F be a face of the polytope $P(\mathcal{B})$ and let A_F be the TU matrix associated with F. Thus $A_F x = b_F$ defines the affine span of F. In other words, the set of vectors parallel to F is precisely the solution set of $A_F x = 0$ and the lattice L_F is given by $L(A_F)$. Theorem 2.5 implies the hypothesis of Theorem 2.4 for any $L_F = L(A_F)$, when the matrix A_F is TU. ◀

2.3 Near-shortest circuits in regular matroids

The proof of Theorem 2.5 is combinatorial and uses the language and results from matroid theory. We recall a few basic definitions from matroid theory. A matroid is said to be *represented by a matrix A*, if its ground set is the column set of A and its independent sets are the sets of linearly independent columns of A. A matroid represented by a TU matrix is said to be a *regular matroid*. A *circuit* of a matroid is a minimal dependent set. The following is one of our main results which gives a bound on the number of near-shortest circuits in a regular matroid, which, in turn, implies Theorem 2.5. Instead of the circuit size, we consider the weight of a circuit and present a more general result.

▶ **Theorem 2.6** (Near-shortest circuits in regular matroids). *Let $M = (E, \mathcal{I})$ be a regular matroid with $m = |E| \geq 2$ and let $w\colon E \to \mathbb{N}$ be a weight function. Suppose M does not have any circuit C with $w(C) < r$ for some number r. Then*

$$|\{\, C \mid C \text{ circuit in } M \text{ and } w(C) < {}^{3r}/_{2} \,\}| \;\leq\; 240\, m^5.$$

▶ Remark. An extension of this result would be to give a polynomial bound on the number of circuits of weight at most αr for any constant α. Our current proof technique does not extend to this setting.

3 Isolation via the Polytope Lattices: Proof of Theorem 2.4

This section is dedicated to a proof of Theorem 2.4. That is, we give a construction of an isolating weight assignment for a family $\mathcal{B} \subseteq 2^E$ assuming that for each face F of the corresponding polytope $P(\mathcal{B})$, the lattice L_F has small number of near-shortest vectors. First, let us see how the isolation question for a family $\mathcal{B}$ translates in the polytope setting. For any weight function $w\colon E \to \mathbb{Z}$, we view w as a vector in $\mathbb{Z}^E$ and consider the function $\langle w, x\rangle$ over the points in $P(\mathcal{B})$. Note that $\langle w, x^B\rangle = w(B)$, for any $B \subseteq E$. Thus, a weight function $w\colon E \to \mathbb{Z}$ is isolating for a family $\mathcal{B}$ if and only if $\langle w, x\rangle$ has a unique minimum over the polytope $P(\mathcal{B})$.

Observe that for any $w\colon E \to \mathbb{Z}$, the points that minimize $\langle w, x\rangle$ in $P(\mathcal{B})$ will form a face of the polytope $P(\mathcal{B})$. The idea is to build the isolating weight function in rounds. In every round, we slightly modify the current weight function to get a smaller minimizing face. Our goal is to significantly reduce the dimension of the minimizing face in every round. We stop when we reach a zero-dimensional face, i.e., we have a unique minimum weight point in $P(\mathcal{B})$.

In the following, we will denote the size of the set E by m. The following claim asserts that if we modify the current weight function on a small scale, then the new minimizing face will be a subset of the current minimizing face. See the full version [11] for a proof.

▶ **Claim 3.1.** *Let $w\colon E \to \mathbb{Z}$ be a weight function and F be the face of $P(\mathcal{B})$ that minimizes w. Let $w'\colon E \to \{0, 1, \ldots, N-1\}$ be another weight function and let F' be the face that minimizes the combined weight function $mN\, w + w'$. Then $F' \subseteq F$.*

Thus, in each round, we will add a new weight function to the current function using a smaller scale and try to get a sub-face with significantly smaller dimension. Henceforth, N will be a sufficiently large number bounded by $\mathrm{poly}(m)$. The following claim gives a way to go to a smaller face.

▶ **Claim 3.2.** *Let F be the face of $P(\mathcal{B})$ minimizing $\langle w, x\rangle$ and let $v \in L_F$. Then $\langle w, v\rangle = 0$.*

Proof. Since $v \in L_F$, we have $v = \alpha(x_1 - x_2)$, for some $x_1, x_2 \in F$ and $\alpha \in \mathbb{R}$. As $x_1, x_2 \in F$, we have $\langle w, x_1\rangle = \langle w, x_2\rangle$. The claim follows. ◀

Now, let F_0 be the face that minimizes the current weight function w_0. Let v be in L_{F_0}. Choose a new weight function $w' \in \{0, 1, \ldots, N-1\}^E$ such that $\langle w', v\rangle \neq 0$. Let $w_1 := mN\, w_0 + w'$ and let F_1 be the face that minimizes w_1. Clearly, $\langle w_1, v\rangle \neq 0$ and thus, by Claim 3.2, $v \notin L_{F_1}$. This implies that F_1 is strictly contained in F_0. To ensure that F_1 is *significantly* smaller than F_0, we choose many vectors in L_{F_0}, say $v_1, v_2, \ldots, v_k$, and construct a weight vector w' such that for all $i \in [k]$, we have $\langle w', v_i\rangle \neq 0$. The following well-known lemma actually constructs a list of weight vectors such that one of them has the desired property (see [8, Lemma 2]).

▶ **Lemma 3.3.** *Given m, k, t, let $q = mk \log t$. In time $\mathrm{poly}(q)$ one can construct a set of weight vectors $w_1, w_2, \ldots, w_q \in \{0, 1, 2, \ldots, q\}^m$ such that for any set of nonzero vectors $v_1, v_2, \ldots, v_k$ in $\{-(t-1), \ldots, 0, 1, \ldots, t-1\}^m$ there exists a $j \in [q]$ such that for all $i \in [k]$ we have $\langle w_j, v_i \rangle \neq 0$. (see the full version [11] for a proof).*

There are two things to note about this lemma: (i) It is black-box in the sense that we do not need to know the set of vectors $\{v_1, v_2, \ldots, v_k\}$. (ii) We do not know a priori which function will work in the given set of functions. So, one has to try all possibilities.

The lemma tells us that we can ensure that $\langle w', v \rangle \neq 0$ for polynomially many vectors v whose coordinates are polynomially bounded. Below, we formally present the weight construction.

To prove Theorem 2.4, let c be the constant in the assumption of the theorem. Let $N = m^{O(1)}$ be a sufficiently large number and $p = \lfloor \log_c(m+1) \rfloor$. Let $w_0 \colon E \to \mathbb{Z}$ be a weight function such that $\langle w_0, v \rangle \neq 0$ for all nonzero $v \in \mathbb{Z}^E$ with $\|v\| < c$. For $i = 1, 2, \ldots, p$, define

- F_{i-1}: the face of $P(\mathcal{B})$ minimizing w_{i-1}
- w'_i: a weight vector in $\{0, 1, \ldots, N-1\}^E$ such that $\langle w'_i, v \rangle \neq 0$ for all nonzero $v \in L_{F_{i-1}}$ with $\|v\| < c^{i+1}$.
- w_i: $mNw_{i-1} + w'_i$.

Observe that $F_i \subseteq F_{i-1}$, for each i by Claim 3.1. Hence, also for the associated lattices we have $L_{F_i} \subseteq L_{F_{i-1}}$. As we show in the next claim, the choice of w'_i together with Claim 3.2 ensures that there are no vectors in L_{F_i} with norm less than c^{i+1}.

▶ **Claim 3.4.** *For $i = 0, 1, 2, \ldots, p$, we have $\lambda(L_{F_i}) \geq c^{i+1}$.*

Proof. Consider a nonzero vector $v \in L_{F_i}$. By Claim 3.2, we have

$$\langle w_i, v \rangle = mN \langle w_{i-1}, v \rangle + \langle w'_i, v \rangle = 0. \tag{1}$$

Since v is in L_{F_i}, it is also in $L_{F_{i-1}}$ and again by Claim 3.2, we have $\langle w_{i-1}, v \rangle = 0$. Together with (1) we conclude that $\langle w'_i, v \rangle = 0$. By the definition of w'_i, this implies that $\|v\| \geq c^{i+1}$. ◀

Finally we argue that w_p is isolating.

▶ **Claim 3.5.** *The face F_p is a point.*

Proof. Let $y_1, y_2 \in F_p$ be vertices and thus belong to $\{0, 1\}^m$. Then $y_1 - y_2 \in L_{F_p}$ and $\|y_1 - y_2\| \leq m < c^{p+1}$. By Claim 3.4, we have that $y_1 - y_2$ must be zero, i.e., $y_1 = y_2$. ◀

We get a bound of $m^{O(\log m)}$ on both the number of weight vectors we need to try and the weights involved, which finishes the proof of Theorem 2.4 (see the full version [11]).

4 Number of Short Vectors in Lattices: Proof of Theorem 2.5

In this section, we show that Theorem 2.5 follows from Theorem 2.6. We define a circuit of a matrix and show that to prove Theorem 2.5, it is sufficient to upper bound the number of near-shortest circuits of a TU matrix. We argue that this, in turn, is implied by a bound on the number of near-shortest circuits of a regular matroid. Just as a circuit of a matroid is a minimal dependent set, a circuit of matrix is a minimal linear dependency among its columns. Recall that for an $n \times m$ matrix A, the lattice $L(A)$ is defined as the set of integer vectors in its kernel,

$$L(A) := \{\, v \in \mathbb{Z}^m \mid Av = 0 \,\}.$$

▶ **Definition 4.1** (Circuit). For an $n \times m$ matrix A, a vector $u \in L(A)$ is a *circuit of* A if
- there is no nonzero $v \in L(A)$ with $\text{supp}(v) \subsetneq \text{supp}(u)$, and
- $\gcd(u_1, u_2, \ldots, u_m) = 1$.

Note that if u is a circuit of A, then so is $-u$. The following property of the circuits of a TU matrix is well known (see [16, Lemma 3.18]).

▶ **Fact 4.2.** *Let A be a TU matrix. Then every circuit of A has its coordinates in $\{-1, 0, 1\}$.*

Now, we define a notion of conformality among two vectors.

▶ **Definition 4.3** (Conformal [16]). Let $u, v \in \mathbb{R}^m$. We say that u is *conformal to* v, denoted by $u \sqsubseteq v$, if $u_i v_i \geq 0$ and $|u_i| \leq |v_i|$, for each $1 \leq i \leq m$.

▶ **Observation 4.4.** *For vectors u and v with $u \sqsubseteq v$, we have $\|v - u\| = \|v\| - \|u\|$.*

The following lemma follows from [16, Lemma 3.19].

▶ **Lemma 4.5.** *Let A be a TU matrix. Then for any nonzero vector $v \in L(A)$, there is a circuit u of A that is conformal to v.*

We use the lemma to argue that any small enough vector in $L(A)$ must be a circuit.

▶ **Lemma 4.6.** *Let A be a TU matrix and let $\lambda := \lambda(L(A))$. Then any nonzero vector $v \in L(A)$ with $\|v\| < 2\lambda$ is a circuit of A.*

Proof. Suppose $v \in L(A)$ is not a circuit of A. We show that $\|v\| \geq 2\lambda$. By Lemma 4.5, there is a circuit u of A with $u \sqsubseteq v$. Since v is not a circuit, $v - u \neq 0$. Since both u and $v - u$ are nonzero vectors in $L(A)$, we have $\|u\|, \|v - u\| \geq \lambda$. By Observation 4.4, we have $\|v\| = \|v - u\| + \|u\|$ and thus, we get that $\|v\| \geq 2\lambda$. ◀

Recall that a matroid represented by a TU matrix is a regular matroid. The following lemma shows that the two definitions of circuits, 1) for TU matrices and 2) for regular matroids, coincide. See the full version [11] for a proof.

▶ **Lemma 4.7.** *Let $M = (E, \mathcal{I})$ be a regular matroid, represented by a TU matrix A. Then there is a one to one correspondence between the circuits of M and the circuits of A (up to change of sign).*

To prove Theorem 2.5, let A be TU matrix. By Lemma 4.6, it suffices to bound the number of near-shortest circuits of A. By Lemma 4.7, the circuits of A and the circuits of the regular matroid M represented by A, coincide. Moreover, the size of a circuit of M is same as the ℓ_1-norm of the corresponding circuit of A, as a circuit of A has its coordinates in $\{-1, 0, 1\}$ by Fact 4.2. Now Theorem 2.5 follows from Theorem 2.6 when we define the weight of each element being 1.

5 Proof Overview of Theorem 2.6

Here we give a proof overview of Theorem 2.6; see the full version [11] for a complete proof. Theorem 2.6 states that for a regular matroid, the number of near-shortest circuits – circuits whose size is a constant multiple of the shortest circuit size – is polynomially bounded. The starting point of the proof of this theorem is a remarkable result of Seymour [21] which showed that every regular matroid can be decomposed into a set of much simpler matroids. Each of these building blocks for regular matroids either belongs to the classes of graphic and cographic matroids – the simplest and well-known examples of regular matroids, or is a

special 10-element matroid R_{10} (see the full version [11] for the definitions). One important consequence of Seymour's result is a polynomial time algorithm, the only one known, for testing the total unimodularity of a matrix; see [18] (recall that a TU matrix represents a regular matroid). Our strategy is to leverage Seymour's decomposition theorem in order to bound the number of circuits in a regular matroid.

Seymour's Theorem and a simple inductive approach

Seymour's decomposition involves a sequence of binary operations on matroids, each of which is either a 1-sum, a 2-sum or a 3-sum. Formally, it states that for every regular matroid M, we can build a decomposition tree – which is a binary rooted tree – in which the root node is the matroid M, every node is a k-sum of its two children for $k = 1, 2$, or 3, and at the bottom we have graphic, cographic and the R_{10} matroids as the leaf nodes. Note that the tree, in general, is not necessarily balanced and can have large depth (linear in the ground set size).

This suggests that to bound the number of near-shortest circuits in a regular matroid, perhaps one can use the tree structure of its decomposition, starting from the leaf nodes and arguing, inductively, all the way up to the root. It is known that the number of near-shortest circuits in graphic and cographic matroids is polynomially bounded. This follows from the polynomial bounds on the number of near-shortest cycles of a graph [24] and on the number of near min-cuts in a graph [13]. The challenge is to show how to combine the information at an internal node.

The k-sum M of two matroids M_1 and M_2 is defined in a way such that each circuit of M can be built from a combination of two circuits, one from M_1 and another from M_2. Thus, if we have upper bounds for the number of circuits in M_1 and M_2, their product will give a naive upper bound for number of circuits in M. Since there can be many k-sum operations involved, the naive product bound can quickly explode. Hence, to keep a polynomial bound we need to take a closer look at the k-sum operations.

k-sum operations

1-sum. A 1-sum M of two matroids M_1 and M_2 is simply their direct sum. That is, the ground set of M is the disjoint union of the ground sets of M_1 and M_2, and any circuit of M is either a circuit of M_1 or a circuit of M_2.

The 2-sum and 3-sum are a bit more intricate. It is known that the set of circuits of a matroid completely characterizes the matroid. The 2-sum and 3-sum operations are defined by describing the set of circuits of the matroid obtained by the sum. To get an intuition for the 2-sum operation, we first describe it on two graphic matroids. A graphic matroid is defined with respect to a graph, where a circuit is a simple cycle in the graph.

2-sum on graphs. For two graphs G_1 and G_2, their 2-sum $G = G_1 \oplus_2 G_2$ is any graph obtained by identifying an edge (u_1, v_1) in G_1 with an edge (u_2, v_2) in G_2, that is, identifying u_1 with u_2 and v_1 with v_2 and then, deleting the edge $(u_1, v_1) = (u_2, v_2)$. It would be instructive to see how a cycle in G, i.e., a circuit of the associated graphic matroid, looks like. A cycle in G is either a cycle in G_1 or in G_2 that avoids the edge $(u_1, v_1) = (u_2, v_2)$, or it is a union of a path $u_1 \rightsquigarrow v_1$ in G_1 and a path $v_2 \rightsquigarrow u_2$ in G_2. This last possibility is equivalent to taking a symmetric difference $C_1 \triangle C_2$ of two cycles C_1 in G_1 and C_2 in G_2 such that C_1 passes through (u_1, v_1) and C_2 passes through (u_2, v_2).

2-sum on matroids. The 2-sum $M_1 \oplus_2 M_2$ of two matroids M_1 and M_2 is defined analogously. The grounds sets of M_1 and M_2, say E_1 and E_2 respectively, have an element in common, say e (this can be achieved by identifying an element from E_1 with an element from E_2). The sum $M_1 \oplus_2 M_2$ is defined on the ground set $E = E_1 \Delta E_2$, the symmetric difference of the two given ground sets. Any circuit of the sum $M_1 \oplus_2 M_2$ is either a circuit in M_1 or in M_2 that avoids the common element e, or it is the symmetric difference $C_1 \triangle C_2$ of two circuits C_1 and C_2 of M_1 and M_2, respectively, such that both C_1 and C_2 contain the common element e.

3-sum on matroids. A 3-sum is defined similarly. A matroid M is a 3-sum of two matroids M_1 and M_2 if their ground sets E_1 and E_2 have a set S of three elements in common such that S is a circuit in both the matroids and the ground set of M is the symmetric difference $E_1 \triangle E_2$. Moreover, a circuit of M is either a circuit in M_1 or in M_2 that avoids the common elements S, or it is the symmetric difference $C_1 \triangle C_2$ of two circuits C_1 and C_2 of M_1 and M_2, respectively, such that both C_1 and C_2 contain a common element e from S and no other element from S.

The inductive bound on the number of circuits

Our proof is by a strong induction on the ground set size.
Base case: For a graphic or cographic matroid with a ground set of size m, if its shortest circuit has size r then the number of its circuits of size less than αr is at most $m^{4\alpha}$. For the R_{10} matroid, we present a constant upper bound on the number of circuits.
Induction hypothesis: For any regular matroid with a ground set of size $m < m_0$, if its shortest circuit has size r, then the number of its circuits of size less than αr is bounded by $m^{c\alpha}$ for some sufficiently large constant c.
Induction step: We prove the induction hypothesis for a regular matroid M with a ground set of size m_0. Let the minimum size of a circuit in M be r. We want to show a bound of $m_0^{c\alpha}$ on the number of circuits in M of size less than αr. The main strategy here is as follows: by Seymour's Theorem, we can write M as a k-sum of two smaller regular matroids M_1 and M_2, with ground sets of size $m_1 < m_0$ and $m_2 < m_0$ respectively. As the circuits of M can be written as a symmetric differences of circuits of M_1 and M_2, we derive an upper bound on the number circuits of M from the corresponding bounds for M_1 and M_2, which we get from the induction hypothesis.

The 1-sum case. In this case, any circuit of M is either a circuit of M_1 or a circuit of M_2. Hence, the number of circuits in M of size less than αr is simply the sum of the number of circuits in M_1 and M_2 of size less than αr. Using the induction hypothesis, this sum is bounded by $m_1^{c\alpha} + m_2^{c\alpha}$, which is less than $m_0^{c\alpha}$ since $m_0 = m_1 + m_2$.

The 2-sum and 3-sum cases. Let the set of common elements in the ground sets of M_1 and M_2 be S. Note that $m_0 = m_1 + m_2 - |S|$. Recall from the definition of a k-sum that any circuit C of M is of the form $C_1 \triangle C_2$, where C_1 and C_2 are circuits in M_1 and M_2 respectively, such that either **(i)** one of them, say C_1, has no element from S and the other one C_2 is empty or **(ii)** they both contain exactly one common element from S. We will refer to C_1 and C_2 as projections of C. Note that $|C_1|, |C_2| \leq |C|$. In particular, if circuit C is of size less than αr, then so are its projections C_1 and C_2.

An obstacle. The first step would be to bound the number of circuits C_1 of M_1 and C_2 of M_2 using the induction hypothesis. However, we do not have a lower bound on the minimum size of a circuit in M_1 or M_2, which is required to use the induction hypothesis. What we do

know is that any circuit in M_1 or M_2 that does not involve elements from S is also a circuit of M, and thus, must have size at least r. However, a circuit that involves elements from S could be arbitrarily small. We give different solutions for this obstacle in case **(i)** and case **(ii)** mentioned above.

Case (i): deleting elements in S. Let us first consider the circuits C_1 of M_1 that do not involve elements from S. These circuits can be viewed as circuits of a new regular matroid $M_1 \setminus S$ obtained by deleting the elements in S from M_1. Since we know that the minimum size of a circuit in $M_1 \setminus S$ is r, we can apply the induction hypothesis to get a bound of $(m_1 - |S|)^{c\alpha}$ for the number of circuits C_1 of $M_1 \setminus S$ of size less than αr. Summing this with a corresponding bound for $M_2 \setminus S$ gives us a bound less than $m_0^{c\alpha}$ for the number of circuits of M in case **(i)**.

Case (ii): stronger induction hypothesis. The case when circuits C_1 and C_2 contain an element from S turns out to be much harder. For this case, we actually need to strengthen our induction hypothesis. Let us assume that for a regular matroid of ground set size $m < m_0$, if the minimum size of a circuit that avoids a given element $\widetilde{e}$ is r, then the number of circuits containing $\widetilde{e}$ and of size less than αr is bounded by $m^{c\alpha}$. This statement will also be proved by induction, but we will come to its proof later.

Since we know that any circuit in M_1 (or M_2) that avoids elements from S has size at least r, we can use the above stronger inductive hypothesis to get a bound of $m_1^{c\alpha}$ on the number of circuits C_1 in M_1 containing a given element from S and of size less than αr. Similarly, we get an analogous bound of $m_2^{c\alpha}$ for circuits C_2 of M_2. Since C can be a symmetric difference of any C_1 and C_2, the product of these two bounds, that is, $(m_1 m_2)^{c\alpha}$ bounds the number of circuits C of M of size less than αr. Unfortunately, this product can be much larger than $m_0^{c\alpha}$. Note that this product bound on the number of circuits C is not really tight since C_1 and C_2 both cannot have their sizes close to αr simultaneously. This is because $C = C_1 \triangle C_2$ and thus, $|C| = |C_1| + |C_2| - 1$. Hence, a better approach is to consider different cases based on the sizes of C_1 and C_2.

Number of circuits C when one of its projections is small. We first consider the case when the size of C_1 is very small, i.e., close to zero. In this case, the size of C_2 will be close to αr and we have to take the bound of $m_2^{c\alpha}$ on the number of such circuits C_2. Now, if number of circuits C_1 with small size is N then we get a bound of $N m_2^{c\alpha}$ on the number of circuits C of M of this case. Note that $N m_2^{c\alpha}$ is dominated by $m_0^{c\alpha}$ only when $N \leq 1$, as m_2 can be comparable to m. While $N \leq 1$ does not always hold, we show something weaker which is true.

Uniqueness of C_1. We can show that for any element s in the set of common elements S, there is at most one circuit C_1 of size less than $r/2$ that contains s and no other element from S. To see this, assume that there are two such circuits C_1 and C_1'. It is known that the symmetric difference of two circuits of a matroid is a disjoint union of some circuits of the matroid. Thus, $C_1 \triangle C_1'$ will be a disjoint union of circuits of M_1. Since $C_1 \triangle C_1'$ does not contain any element from S, it is also a disjoint union of circuits of M. This would lead us to a contradiction because the size of $C_1 \triangle C_1'$ is less than r and M does not have circuits of size less than r. This proves the uniqueness of C_1. Our problem is still not solved since the set S can have three elements in case of a 3-sum, and thus, there can be three possibilities for C_1 (i.e., N=3).

Assigning weights to the elements. To get around this problem, we use a new idea of considering matroids elements with weights. For each element s in S, consider the unique circuit C_1 of size at most $r/2$ that contains s. In the matroid M_2, we assign a weight of $|C_1| - 1$ to the element s. The elements outside S get weight 1. The weight of element $s \in S$ signifies that if a circuit C_2 of M_2 contains s then it has to be summed up with the unique circuit C_1 containing s, which adds a weight of $|C_1| - 1$. Essentially, the circuits of the weighted matroid M_2 that have weight γ will have a one-to-one correspondence with circuits $C = C_1 \triangle C_2$ of M that have size γ and have $|C_1| < r/2$. Hence, we can assume there are no circuits in the weighted matroid M_2 of weight less than r. Thus, we can apply the induction hypothesis on M_2, but we need to further strengthen the hypothesis to a weighted version. By this new induction hypothesis, we will get a bound of $m_2^{c\alpha}$ on the number of circuits of M_2 with weight less that αr. As mentioned above, this will bound the number of circuits $C = C_1 \triangle C_2$ of M with size less than αr and $|C_1| < r/2$. Note that the bound $m_2^{c\alpha}$ is smaller than the desired bound $m_0^{c\alpha}$.

Number of circuits C when none of its projections is small. It is relatively easier to handle the other case when C_1 has size at least $r/2$ (and less than αr). In this case, C_2 has size less than $(\alpha - 1/2)r$. The bounds we get by the induction hypothesis for the number of circuits C_1 and C_2 are $m_1^{c\alpha}$ and $m_2^{c(\alpha-1/2)}$ respectively. Their product $m_1^{c\alpha} m_2^{c(\alpha-1/2)}$ bounds the number of circuits C in this case. However, this product is not bounded by $m_0^{c\alpha}$.

Stronger version of Seymour's Theorem. To get a better bound we need another key idea. Instead of Seymour's Theorem, we work with a stronger variant given by Truemper [27]. It states that any regular matroid can be written as a k-sum of two smaller regular matroids M_1 and M_2 for $k = 1, 2$ or 3 such that one of them, say M_1, is a graphic, cographic or R_{10} matroid. The advantage of this stronger statement is that we can take a relatively smaller bound on the number of circuits of M_1, which gives us more room for the inductive argument. Formally, we know from above that when M_1 is a graphic or cographic matroid, the number of its circuits of size less than αr is at most $m_1^{4\alpha}$. One can choose the constant c in our induction hypothesis to be sufficiently large so that the product $m_1^{4\alpha} m_2^{c(\alpha-1/2)}$ is bounded by $m_0^{c\alpha}$.

A stronger induction hypothesis

To summarize, we work with an inductive hypothesis as follows: If a regular matroid (with weights) has no circuits of weight less than r that avoid a given set R of elements then the number of circuits of weight less than αr that contain the set R is bounded by $m^{c\alpha}$. As the base case, we first show this statement for the graphic and cographic case.

When we rerun the whole inductive argument with weights and with a fixed set R, we run into another issue. It turns out that in the case when the size of C_1 is very small, our arguments above do not go through if C_1 has some elements from R. To avoid such a situation we use yet another strengthened version of Seymour's Theorem. It says that any regular matroid with a given element $\widetilde{e}$ can be written as a k-sum of two smaller regular matroids M_1 and M_2, such that M_1 is a graphic, cographic or R_{10} matroid and M_2 is a regular matroid containing $\widetilde{e}$. When our R is a single element set, say $\{\widetilde{e}\}$, we use this theorem to ensure that M_1, and thus C_1, has no elements from R. This rectifies the problem when R has size 1. However, as we go deeper inside the induction, the set R can grow in size. Essentially, whenever α decreases by $1/2$ in the induction, the size of R grows by 1. Thus, we take α to be $3/2$, which means that to reach $\alpha = 1$ we need only one step of decrement, and thus, the size of R at most becomes 1. This is the reason our main theorem only deals with circuits of size less than $3/2$ times the smallest size.

References

1. Manindra Agrawal, Thanh Minh Hoang, and Thomas Thierauf. The polynomially bounded perfect matching problem is in NC^2. In *24th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 4393 of *Lecture Notes in Computer Science*, pages 489–499. Springer Berlin Heidelberg, 2007. doi:10.1007/978-3-540-70918-3_42.
2. Vikraman Arvind and Partha Mukhopadhyay. Derandomizing the isolation lemma and lower bounds for circuit size. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, 11th International Workshop, APPROX, and 12th International Workshop, RANDOM*, pages 276–289, 2008. doi:10.1007/978-3-540-85363-3_23.
3. Andreas Björklund and Thore Husfeldt. Shortest two disjoint paths in polynomial time. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 211–222, 2014. doi:10.1007/978-3-662-43948-7_18.
4. C. Bourke, R. Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Trans. Comput. Theory*, 1:4:1–4:17, February 2009. doi:10.1145/1490270.1490274.
5. Elias Dahlhaus and Marek Karpinski. Matching and multidimensional matching in chordal and strongly chordal graphs. *Discrete Applied Mathematics*, 84(1–3):79–91, 1998. doi:10.1016/S0166-218X(98)00006-7.
6. Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory of Computing Systems*, 47:737–757, 2010. doi:10.1007/s00224-009-9204-8.
7. Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-NC. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 754–763, 2016. doi:10.1145/2897518.2897564.
8. Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with O(1) worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.
9. Dima Grigoriev and Marek Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in NC (extended abstract). In *28th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 166–172, 1987. doi:10.1109/SFCS.1987.56.
10. Rohit Gurjar and Thomas Thierauf. Linear matroid intersection is in quasi-NC. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 821–830, New York, NY, USA, 2017. ACM. doi:10.1145/3055399.3055440.
11. Rohit Gurjar, Thomas Thierauf, and Nisheeth K. Vishnoi. Isolating a vertex via lattices: Polytopes with totally unimodular faces. *ArXiv e-prints*, abs/1708.02222, August 2017. arXiv:1708.02222.
12. Vivek Anand T. Kallampally and Raghunath Tewari. Trading determinism for time in space bounded computations. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 10:1–10:13, 2016. doi:10.4230/LIPIcs.MFCS.2016.10.
13. David R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 21–30, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=313559.313605.
14. Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987. doi:10.1007/BF02579206.

15 H. Narayanan, Huzur Saran, and Vijay V. Vazirani. Randomized parallel algorithms for matroid union and intersection, with applications to arboresences and edge-disjoint spanning trees. *SIAM J. Comput.*, 23(2):387–397, 1994. doi:10.1137/S0097539791195245.

16 S. Onn. *Nonlinear Discrete Optimization: An Algorithmic Theory.* Zurich lectures in advanced mathematics. European Mathematical Society Publishing House, 2010. URL: https://books.google.co.il/books?id=wzAGMQAACAAJ.

17 Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM Journal on Computing*, 29(4):1118–1131, 2000. doi:10.1137/S0097539798339041.

18 Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986.

19 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency. Vol. A. , Paths, flows, matchings.* Algorithms and combinatorics. Springer-Verlag, Berlin, Heidelberg, New York, 2003.

20 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency. Vol. B. , Matroids, trees, stable sets.* Algorithms and combinatorics. Springer-Verlag, Berlin, Heidelberg, New York, N.Y., et al., 2003.

21 Paul D. Seymour. Decomposition of regular matroids. *J. Comb. Theory, Ser. B*, 28(3):305–359, 1980. doi:10.1016/0095-8956(80)90075-1.

22 Mohit Singh and Nisheeth K. Vishnoi. Entropy, optimization and counting. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 50–59. ACM, 2014.

23 Damian Straszak and Nisheeth K. Vishnoi. Computing Maximum Entropy Distributions Everywhere. *ArXiv e-prints*, 2017. arXiv:1711.02036.

24 Ashok Subramanian. A polynomial bound on the number of light cycles in an undirected graph. *Information Processing Letters*, 53(4):173–176, 1995. doi:10.1016/0020-0190(94)00202-A.

25 Ola Svensson and Jakub Tarnawski. The matching problem in general graphs is in quasi-NC. In *58th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2017, 15-17 October, 2017, Berkeley, California, USA*, 2017.

26 C. P. Teo and K. M. Koh. The number of shortest cycles and the chromatic uniqueness of a graph. *Journal of Graph Theory*, 16(1):7–15, 1992.

27 Klaus Truemper. *Matroid Decomposition.* Leibniz, Plano, Texas (USA), 1998.

28 Dieter van Melkebeek and Gautam Prakriya. Derandomizing isolation in space-bounded settings. In *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*, pages 5:1–5:32, 2017. doi:10.4230/LIPIcs.CCC.2017.5.

29 Avi Wigderson. NL/poly $\subseteq \oplus$ L/poly. In *Proceedings of the Ninth Annual Structure in Complexity Theory Conference, Amsterdam, The Netherlands, June 28 - July 1, 1994*, pages 59–62, 1994. doi:10.1109/SCT.1994.315817.

Synchronization Strings: Channel Simulations and Interactive Coding for Insertions and Deletions

Bernhard Haeupler[1]
Carnegie Mellon University, Pittsburgh, PA, USA
haeupler@cs.cmu.edu

Amirbehshad Shahrasbi[2]
Carnegie Mellon University, Pittsburgh, PA, USA
shahrasbi@cs.cmu.edu

Ellen Vitercik
Carnegie Mellon University, Pittsburgh, PA, USA
vitercik@cs.cmu.edu

Abstract

We present many new results related to reliable (interactive) communication over insertion-deletion channels. *Synchronization errors*, such as insertions and deletions, strictly generalize the usual *symbol corruption errors* and are much harder to protect against.

We show how to hide the complications of synchronization errors in many applications by introducing very general *channel simulations* which efficiently transform an insertion-deletion channel into a regular symbol corruption channel with an error rate larger by a constant factor and a slightly smaller alphabet. We utilize and generalize synchronization string based methods which were recently introduced as a tool to design essentially optimal error correcting codes for insertion-deletion channels. Our channel simulations depend on the fact that, at the cost of increasing the error rate by a constant factor, synchronization strings can be decoded in a streaming manner that preserves linearity of time. Interestingly, we provide a lower bound showing that this constant factor cannot be improved to $1+\varepsilon$, in contrast to what is achievable for error correcting codes. Our channel simulations drastically and cleanly generalize the applicability of synchronization strings.

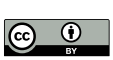
We provide new interactive coding schemes which simulate any interactive two-party protocol over an insertion-deletion channel. Our results improve over the interactive coding schemes of Braverman et al. [TransInf '17] and Sherstov and Wu [FOCS '17] which achieve a small constant rate and require exponential time computations with respect to computational and communication complexities. We provide the first computationally efficient interactive coding schemes for synchronization errors, the first coding scheme with a rate approaching one for small noise rates, and also the first coding scheme that works over arbitrarily small alphabet sizes. We also show tight connections between synchronization strings and edit-distance tree codes which allow us to transfer results from tree codes directly to edit-distance tree codes.

Finally, using on our channel simulations, we provide an explicit low-rate binary insertion-deletion code that improves over the state-of-the-art codes by Guruswami and Wang [TransInf '17] in terms of rate-distance trade-off.

2012 ACM Subject Classification Mathematics of computing → Coding theory, Theory of computation → Interactive computation

Keywords and phrases Synchronization Strings, Channel Simulation, Coding for Interactive Communication

[1] Supported in part by NSF grants CCF-1527110, CCF-1618280 and NSF CAREER award CCF-1750808.
[2] Supported in part by NSF grants CCF-1527110, CCF-1618280 and NSF CAREER award CCF-1750808.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 75; pp. 75:1–75:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.75

Related Version An extended version is available at `https://arxiv.org/pdf/1707.04233.pdf`.

Acknowledgements The authors thank Allison Bishop for valuable discussions in the early stages of this work.

1 Introduction

Communication in the presence of *synchronization errors*, which include both insertions and deletions, is a fundamental problem of practical importance which eluded a strong theoretical foundation for decades. This remained true even while communication in the presence of *half-errors*, which consist of symbol corruptions and erasures, has been the subject of an extensive body of research with many groundbreaking results. Synchronization errors are strictly more general than half-errors, and thus synchronization errors pose additional challenges for robust communication.

In this work, we show that one-way and interactive communication in the presence of synchronization errors can be reduced to the problem of communication in the presence of half-errors. We present a series of efficient channel simulations which allow two parties to communicate over a channel afflicted by synchronization errors as though they were communicating over a half-error channel with only a slightly larger error rate. This allows us to leverage existing coding schemes for robust communication over half-error channels in order to derive strong coding schemes resilient to synchronization errors.

One of the primary tools we use are *synchronization strings*, which were recently introduced by Haeupler and Shahrasbi in order to design essentially optimal error correcting codes (ECCs) robust to synchronization errors [19]. For every $\varepsilon > 0$, synchronization strings allow a sender to index a sequence of messages with an alphabet of size $\varepsilon^{-O(1)}$ in such a way that k synchronization errors are efficiently transformed into $(1+\varepsilon)k$ half-errors for the purpose of designing ECCs. Haeupler and Shahrasbi provide a black-box construction which transforms any ECC into an equally efficient ECC robust to synchronization errors. However, channel simulations and interactive coding in the presence of synchronization errors present a host of additional challenges that cannot be solved by the application of an ECC. Before we describe our results and techniques in detail, we begin with an overview of the well-known interactive communication model.

Interactive communication. Throughout this work, we study a scenario where there are two communicating parties, whom we call Alice and Bob. The two begin with some input symbols and wish to compute a function of their input by having a conversation. Their goal is to succeed with high probability while communicating as few symbols as possible. In particular, if their conversation would consist of n symbols in the noise-free setting, then they would like to converse for at most αn symbols, for some small α, when in the presence of noise. One might hope that Alice and Bob could correspond using error-correcting codes. However, this approach would lead to poor performance because if a party incorrectly decodes a single message, then the remaining communication is rendered useless. Therefore, only a very small amount of noise could be tolerated, namely less than the amount to corrupt a single message.

There are three major aspects of coding schemes for interactive communication that have been extensively studied in the literature. The first is the coding scheme's **maximum**

tolerable error-fraction or, in other words, the largest fraction of errors for which the coding scheme can successfully simulate any given error-free protocol. Another important quality of coding schemes for interactive communication, as with one-way communication, is **communication rate**, i.e., the amount of communication overhead in terms of the error fraction. Finally, the **efficiency** of interactive coding schemes have been of concern in the previous work.

Schulman initiated the study of error-resilient interactive communication, showing how to convert an arbitrary two-party interactive protocol to one that is robust to a $\delta = 1/240$ fraction of adversarial errors with a constant communication overhead [22, 23]. Braverman and Rao increased the bound on the tolerable adversarial error rate to $\delta < 1/4$, also with a constant communication overhead [9]. Brakerski et al. [2] designed the first efficient coding scheme resilient to a constant fraction of adversarial errors with constant communication overhead. The above-mentioned schemes achieve a constant overhead no matter the level of noise. Kol and Raz were the first to study the trade-off between error fraction and communication rate [21]. Haeupler then provided a coding scheme with a communication rate of $1 - O(\sqrt{\delta \log\log(1/\delta)})$ over an adversarial channel [17]. Further prior work has studied coding for interactive communication focusing on communication efficiency and noise resilience [18, 7, 14] as well as computational efficiency [4, 3, 2, 12, 13, 14]. Other works have studied variations of the interactive communication problem [15, 11, 10, 1, 5].

The main challenge that *synchronization errors* pose is that they may cause the parties to become "out of sync." For example, suppose the adversary deletes a message from Alice and inserts a message back to her. Neither party will know that Bob is a message behind, and if this corruption remains undetected, the rest of the communication will be useless. In most state-of-the-art interactive coding schemes for symbol corruptions, the parties communicate normally for a fixed number of rounds and then send back and forth a series of checks to detect any symbol corruptions that may have occurred. One might hope that a synchronization error could be detected during these checks, but the parties may be out of sync while performing the checks, thus rendering them useless as well. Therefore, synchronization errors require us to develop new techniques.

Very little is known regarding coding for interactive communication in the presence of synchronization errors. A 2016 coding scheme by Braverman et al. [8], which can be seen as the equivalent of Schulman's seminal result, achieves a small constant communication rate while being robust against a $1/18 - \varepsilon$ fraction of errors. The coding scheme relies on edit-distance tree codes, which are a careful adaptation of Schulman's original tree codes [23] for edit distance, so the decoding operations are not efficient and require exponential time computations. A recent work by Sherstov and Wu [25] closed the gap for maximum tolerable error fraction by introducing a coding scheme that is robust against $1/6 - \varepsilon$ fraction of errors which is the highest possible fraction of insertions and deletions under which any coding scheme for interactive communication can work. Both Braverman et al. [8] and Sherstov and Wu [25] schemes are of constant communication rate, over large enough constant alphabets, and inefficient. In this work we address the next natural questions which, as arose with ordinary corruption interactive communication, are on finding interactive coding schemes that are computationally efficient or achieve super-constant communication efficiency.

Our results. We present very general channel simulations which allow two parties communicating over an insertion-deletion channel to follow any protocol designed for a regular symbol corruption channel. The fraction of errors on the simulated symbol corruption channel is only slightly larger than that on the insertion-deletion channel. Our channel simulations

are made possible by synchronization strings. Crucially, at the cost of increasing the error rate by a constant factor, synchronization strings can be decoded in a streaming manner which preserves linearity of time. Note that the similar technique is used in Haeupler and Shahrasbi [19] to transform synchronization errors into ordinary symbol corruptions as a stepping-stone to obtain insertion-deletion codes from ordinary error correcting codes in a black-box fashion. However, in the context of error correcting codes, there is no requirement for this transformation to happen in real time. In other words, in the study of insertion-deletion codes by Haeupler and Shahrasbi [19], the entire message transmission is done and then the receiving party uses the entire message to transform the synchronization errors into symbol corruptions. In the channel simulation problem, this transformation is required to happen on the fly. Interestingly, we have found out that in the harder problem of channel simulation, the factor by which the number of synchronization errors increase by being transformed into corruption errors cannot be improved to $1 + o(1)$, in contrast to what is achievable for error correcting codes. This work exhibits the widespread applicability of synchronization strings and opens up several new use cases, such as coding for interactive communication over insertion-deletion channels. Namely, using synchronization strings, we provide techniques to obtain the following simulations of corruption channels over given insertion-deletion channels with binary and large constant alphabet sizes.

▶ **Theorem 1** (Informal Statement)**.**

(a) *Suppose that n rounds of a one-way/interactive insertion-deletion channel over an alphabet Σ with a δ fraction of insertions and deletions are given. Using an ε-synchronization string over an alphabet Σ_{syn}, it is possible to simulate $n\,(1 - O_\varepsilon(\delta))$ rounds of a one-way/interactive corruption channel over Σ_{sim} with at most $O_\varepsilon\,(n\delta)$ symbols corrupted so long as $|\Sigma_{sim}| \times |\Sigma_{syn}| \leq |\Sigma|$.*

(b) *Suppose that n rounds of a binary one-way/interactive insertion-deletion channel with a δ fraction of insertions and deletions are given. It is possible to simulate $n(1 - \Theta(\sqrt{\delta \log(1/\delta)}))$ rounds of a binary one-way/interactive corruption channel with $\Theta(\sqrt{\delta \log(1/\delta)})$ fraction of corruption errors between two parties over the given channel.*

Based on the channel simulations presented above, we present novel interactive coding schemes which simulate any interactive two-party protocol over an insertion-deletion channel.

We use our large alphabet interactive channel simulation along with constant-rate efficient coding scheme of Ghaffari and Haeupler [14] for interactive communication over corruption channels to obtain a coding scheme for insertion-deletion channels that is efficient, has a constant communication rate, and tolerates up to $1/44 - \varepsilon$ fraction of errors. Note that despite the fact that this coding scheme fails to protect against the optimal $1/6 - \varepsilon$ fraction of synchronization errors as the recent work by Sherstov and Wu [25] does, it is an improvement over all previous work in terms of computational efficiency as it is the first efficient coding scheme for interactive communication over insertion-deletion channels.

▶ **Theorem 2.** *For any constant $\varepsilon > 0$ and n-round alternating protocol Π, there is an efficient randomized coding scheme simulating Π in presence of $\delta = 1/44 - \varepsilon$ fraction of edit-corruptions with constant rate (i.e., in $O(n)$ rounds) and in $O(n^5)$ time that works with probability $1 - 2^{\Theta(n)}$. This scheme requires the alphabet size to be a large enough constant $\Omega_\varepsilon(1)$.*

Next, we use our small alphabet channel simulation and the corruption channel interactive coding scheme of Haeupler [17] to introduce an interactive coding scheme for insertion-deletion channels. This scheme is not only computationally efficient, but also the first with super

constant communication rate. In other words, this is the first coding scheme for interactive communication over insertion-deletion channels whose rate approaches one as the error fraction drops to zero. Our computationally efficient interactive coding scheme achieves a near-optimal communication rate of $1 - O(\sqrt{\delta \log(1/\delta)})$ and tolerates a δ fraction of errors. Besides computational efficiency and near-optimal communication rate, this coding scheme improves over all previous work in terms of alphabet size. As opposed to coding schemes provided by the previous work[8, 25], our scheme does not require a large enough constant alphabet and works even for binary alphabets.

▶ **Theorem 3.** *For sufficiently small δ, there is an efficient interactive coding scheme for fully adversarial binary insertion-deletion channels which is robust against δ fraction of edit-corruptions, achieves a communication rate of $1 - \Theta(\sqrt{\delta \log(1/\delta)})$, and works with probability $1 - 2^{-\Theta(n\delta)}$.*

We also utilize the channel simulations in one-way settings to provide efficient binary insertion-deletion codes correcting δ-fraction of synchronization errors–for δ smaller than some constant–with a rate of $1 - \Theta(\sqrt{\delta \log(1/\delta)})$. This is an improvement in terms of rate-distance trade-off over the state-of-the-art low-rate binary insertion-deletion codes by Guruswami and Wang [16]. The codes by Guruswami and Wang [16] achieve a rate of $1 - O(\sqrt{\delta} \log(1/\delta))$.

Finally, we introduce a slightly improved definition of edit-distance tree codes, a generalization of Schulman's original tree codes defined by Braverman et al. [8]. We show that under our revised definition, edit-distance tree codes are closely related to synchronization strings. For example, edit-distance tree codes can be constructed by merging a regular tree code and a synchronization string. This transfers, for example, Braverman's sub-exponential time tree code construction [6] and the candidate construction of Schulman [24] from tree codes to edit-distance tree codes. Lastly, as a side note, we will show that with the improved definition, the coding scheme of Braverman et al. [8] can tolerate $1/10 - \varepsilon$ fraction of synchronization errors rather than $1/18 - \varepsilon$ fraction that the scheme based on their original definition did. This improved definition is independently observed by Sherstov and Wu [25].

1.1 Definitions and preliminaries

In this section, we define the channel models and communication settings considered in this work. We also provide notation and define synchronization strings.

Error model and communication channels. In this work, we study two types of channels, which we call *corruption channels* and *insertion-deletion channels*. In the corruption channel model, two parties communicate with an alphabet Σ, and if one party sends a message $c \in \Sigma$ to the other party, then the other party will receive a message $\tilde{c} \in \Sigma$, but it may not be the case that $c = \tilde{c}$.

In the one-way communication setting over an insertion-deletion channel, messages to the listening party may be inserted, and messages sent by the sending party may be deleted. The two-way channel requires a more careful setup. We emphasize that we cannot hope to protect against arbitrary insertions and deletions in the two-way setting because in the message-driven model, a single deletion could cause the protocol execution to "hang." Therefore, following the standard model of Braverman et al.'s work [8] that is employed in all other previous works on this problem [25], we restrict our attention to *edit corruptions*, which consist of a single deletion followed by a single insertion, which may be aimed at either party. Braverman et al. [8] provide a detailed discussion on their model and show that it is strong

enough to generalize other natural models one might consider, including models that utilize clock time-outs to overcome the stalling issue.

In both the one- and two-way communication settings, we study *adversarial channels* with *error rate* δ. Our coding schemes are robust in both the *fully adversarial* and the *oblivious adversary* models.

Interactive and one-way communication protocols. In an *interactive protocol* Π over a channel with an alphabet Σ, Alice and Bob begin with two inputs from Σ^* and then engage in n *rounds* of communication. In a single round, each party either listens for a message or sends a message, where this choice and the message, if one is generated, depends on the party's state, its input, and the history of the communication thus far. After the n rounds, the parties produce an output. We study *alternating protocols*, where each party sends a message every other round and listens for a message every other round. In this message-driven paradigm, a party "sleeps" until a new message comes, at which point the party performs a computation and sends a message to the other party. In the presence of noise, we say that a protocol Π' *robustly simulates* a deterministic protocol Π over a channel C if given any inputs for Π, the parties can decode the transcript of the execution of Π on those inputs over a noise-free channel from the transcript of the execution of Π' over C.

Finally, we also study *one-way communication*, where one party sends all messages and the other party listens. Coding schemes in this setting are known as *error-correcting codes*.

Synchronization Strings. In short, synchronization strings [19] allow communicating parties to protect against synchronization errors by indexing their messages without blowing up the communication rate. We describe this technique by introducing two intermediaries, C_A and C_B, that conduct the communication over the given insertion-deletion channel. C_A receives all symbols that Alice wishes to send to Bob, C_A sends the symbols to C_B, and C_B communicates the symbols to Bob. C_A and C_B handle the synchronization strings and all the extra work that is involved in keeping Alice and Bob in sync by guessing the actual index of symbols received by C_B. In this way, Alice and Bob communicate via C_A and C_B as though they were communicating over a half-error channel.

Unfortunately, trivially attaching the indices $1, 2, \ldots, n$ to each message will not allow us to maintain a near optimal communication rate. If C_A attaches an index to each of Alice's messages, it would increase the size of Σ by a factor of n and the rate would increase by a factor of $1/\log n$, which is far from optimal. Synchronization strings allow the communicating parties to index their messages using an alphabet size that is independent of the total communication length n.

Suppose that with each of Alice's n messages, C_A sends an encoding of her index using a symbol from Σ. Let S be a "synchronization string" consisting of all n encoded indices sent by C_A. Further, suppose that the adversary injects a total of $n\delta$ insertions and deletions, thus transforming the string S to the string S_τ. Let some element of S like $S[i]$ pass through the channel without being deleted by the adversary and arrive as $S_\tau[j]$. We call $S_\tau[j]$ a *successfully transmitted symbol*.

We assume that C_A and C_B know the string S *a priori*. The intermediary C_B will receive a set of transmitted indices $S_\tau[1], \ldots, S_\tau[n]$. Upon receipt of the jth transmitted index, C_B guesses the actual index of the received symbol when sent by C_A. We call the algorithm that C_B runs to determine this an *(n, δ)-indexing algorithm.* The algorithm can also return a symbol $\top$ which represents an "I don't know" response. Any successfully transmitted symbols that is decoded incorrectly is called a *misdecoding*. The number of misdecodings that

an (n, δ)-indexing algorithm might produced is used as a measure to valuate its quality. An indexing algorithm is *streaming* if its guess for a received symbol only depends on previously arrived symbols.

Haeupler and Shahrasbi [19] defined a family of synchronization strings that admit an (n, δ)-indexing algorithm with strong performance.

▶ **Definition 4** (ε-Synchronization String)**.** A string $S \in \Sigma^n$ is an ε-synchronization string if for every $1 \leq i < j < k \leq n+1$ we have that $ED\left(S[i,j), S[j,k)\right) > (1-\varepsilon)(k-i)$.

Haeupler and Shahrasbi [19, 20] prove the existence and provide several fast constructions for ε-synchronization strings and provide a streaming (n, δ)-indexing algorithm that returns a solution with $\frac{c_i}{1-\varepsilon} + \frac{c_d \varepsilon}{1-\varepsilon}$ misdecodings. The algorithm runs in time $O(n^5)$, spending $O(n^4)$ on each received symbol.

2 Channel Simulations

In this section, we show how ε-synchronization strings can be used as a powerful tool to simulate corruption channels over insertion-deletion channels. In Section 3, we use these simulations to introduce coding schemes resilient to insertion-deletion errors.

2.1 One-way channel simulation over a large alphabet

Assume that Alice and Bob have access to n rounds of communication over a one-way insertion-deletion channel where the adversary is allowed to insert or delete up to $n\delta$ symbols. In this situation, we formally define a corruption channel simulation over the given insertion-deletion channel as follows:

▶ **Definition 5** (Corruption Channel Simulation)**.** Let Alice and Bob have access to n rounds of communication over a one-way insertion-deletion channel with the alphabet Σ. The adversary may insert or delete up to $n\delta$ symbols. Intermediaries C_A and C_B *simulate* n' rounds of a corruption channel with alphabet Σ_{sim} over the given channel as follows. First, the adversary can insert a number of symbols into the insertion-deletion channel between C_A and C_B. Then for n' rounds $i = 1, \ldots, n'$, the following procedure repeats:

1. Alice gives $X_i \in \Sigma_{sim}$ to C_A.
2. Upon receiving X_i from Alice, C_A wakes up and sends a number of symbols (possibly zero) from the alphabet Σ to C_B through the given insertion-deletion channel. The adversary can delete any of these symbols or insert symbols *before*, *among*, or *after* them.
3. Upon receiving symbols from the channel, C_B wakes up and reveals a number of symbols (possibly zero) from the alphabet Σ_{sim} to Bob. We say all such symbols are *triggered* by X_i.

Throughout this procedure, the adversary can insert or delete up to $n\delta$ symbols. However, C_B is required to reveal exactly n' symbols to Bob regardless of the adversary's actions. Let $\tilde{X}_1, \cdots, \tilde{X}_{n'} \in \Sigma_{sim}$ be the symbols revealed to Bob by C_B. This procedure successfully simulates n' rounds of a corruption channel with a δ' fraction of errors if for all but $n'\delta'$ elements i of the set $\{1, \ldots, n'\}$, the following conditions hold: 1) $\tilde{X}_i = X_i$; and 2) $\tilde{X}_i$ is triggered by X_i.

When $\tilde{X}_i = X_i$ and $\tilde{X}_i$ is triggered by X_i, we call $\tilde{X}_i$ an *uncorrupted symbol*. The second condition, that $\tilde{X}_i$ is triggered by X_i, is crucial to preserving linearity of time, which is the fundamental quality that distinguishes channel simulations from channel codings. It forces C_A to communicate each symbol to Alice as soon as it arrives. Studying channel simulations

satisfying this condition is especially important in situations where Bob's messages depends on Alice's, and vice versa.

Conditions (1) and (2) also require that C_B conveys at most one uncorrupted symbol each time he wakes up. As the adversary may delete $n\delta$ symbols from the insertion-deletion channel, C_B will wake up at most $n(1-\delta)$ times. Therefore, we cannot hope for a corruption channel simulation where Bob receives more than $n(1-\delta)$ uncorrupted symbols. In the following theorem, we prove something slightly stronger: no deterministic one-way channel simulation can guarantee that Bob receives more than $n(1-4\delta/3)$ uncorrupted symbols and if the simulation is randomized, the expected number of uncorrupted transmitted symbols is at most $n(1-7\delta/6)$. This puts channel simulation in contrast to channel coding as one can recover $1-\delta-\varepsilon$ fraction of symbols there (as shown in [19]).

▶ **Theorem 6.** *Assume that n uses of a one-way insertion-deletion channel over an arbitrarily large alphabet Σ with a δ fraction of insertions and deletions are given. There is no deterministic simulation of a corruption channel over any alphabet Σ_{sim} where the simulated channel guarantees more than $n\,(1-4\delta/3)$ uncorrupted transmitted symbols. If the simulation is randomized, the expected number of uncorrupted transmitted symbols is at most $n(1-7\delta/6)$.*

We now provide a channel simulation using ε-synchronization strings. Every time C_A receives a symbol from Alice (from an alphabet Σ_{sim}), C_A appends a new symbol from a predetermined ε-synchronization string over an alphabet Σ_{syn} to Alice's symbol and sends it as one message through the channel. On the other side of channel, suppose that C_B has already revealed some number of symbols to Bob. Let $\mathtt{I}_B$ be the index of the next symbol C_B expects to receive. Upon receiving a new symbol from C_A, C_B uses the part of the message coming from the synchronization string to guess the index of the message Alice sent. We will refer to this decoded index as $\tilde{\mathtt{I}}_A$ and its actual index as $\mathtt{I}_A$. If $\widetilde{\mathtt{I}}_A < \mathtt{I}_B$, then C_B reveals nothing to Bob and ignores the message he just received. Meanwhile, if $\widetilde{\mathtt{I}}_A = \mathtt{I}_B$, then C_B reveals Alice's message to Bob. Finally, if $\widetilde{\mathtt{I}}_A > \mathtt{I}_B$, then C_B sends a dummy symbol to Bob and then sends Alice's message.Theorem 7 details the simulation guarantees.

▶ **Theorem 7.** *Assume that n uses of a one-way insertion-deletion channel over an alphabet Σ with a δ fraction of insertions and deletions are given. Using an ε-synchronization string over an alphabet Σ_{syn}, it is possible to simulate $n(1-\delta)$ rounds of a one-way corruption channel over Σ_{sim} with at most $2n\delta(2+(1-\varepsilon)^{-1})$ symbols corrupted so long as $|\Sigma_{sim}| \times |\Sigma_{syn}| \leq |\Sigma|$ and $\delta < 1/7$.*

2.2 Interactive channel simulation over a large alphabet

We now turn to channel simulations for interactive channels. As in Section 2.1, we formally define a corruption interactive channel simulation over a given insertion-deletion interactive channel. We then use synchronization strings to present one such simulation.

▶ **Definition 8** (Corruption Interactive Channel Simulation)**.** Let Alice and Bob have access to n rounds of communication over an interactive insertion-deletion channel with alphabet Σ. The adversary may insert or delete up to $n\delta$ symbols. The intermediaries C_A and C_B *simulate* n' rounds of a corruption interactive channel with alphabet Σ_{sim} over the given channel as follows. The communication starts when Alice gives a symbol from Σ_{sim} to C_A. Then Alice, Bob, C_A, and C_B continue the communication as follows:

1. Whenever C_A receives a symbol from Alice or C_B, he either reveals a symbol from Σ_{sim} to Alice or sends a symbol from Σ through the insertion-deletion channel to C_B.

2. Whenever C_B receives a symbol from Bob or C_A, he either reveals a symbol from Σ_{sim} to Bob or send a symbols from Σ through the insertion-deletion channel to C_A.
3. Whenever C_B reveals a symbol to Bob, Bob responds with a new symbol from Σ_{sim}.
4. Whenever C_A reveals a symbol to Alice, Alice responds with a symbol in Σ_{sim} except for the $\frac{n'}{2}$th time.

Throughout this procedure, the adversary can inject up to $n\delta$ edit corruptions. However, regardless of the adversary's actions, C_A and C_B have to reveal exactly $n'/2$ symbols to Alice and Bob respectively.

Let $X_1, \ldots, X_{n'}$ be the symbols Alice gives to C_A and $\tilde{X}_1, \ldots, \tilde{X}_{n'} \in \Sigma_{sim}$ be the symbols C_B reveals to Bob. Similarly, Let $Y_1, \ldots, Y_{n'}$ be the symbols Bob gives to C_B and $\tilde{Y}_1, \ldots, \tilde{Y}_{n'} \in \Sigma_{sim}$ be the symbols C_A reveals to Alice. We call each pair of tuples $(X_i, \tilde{X}_i)$ and $(Y_i, \tilde{Y}_i)$ a *round* of the simulated communication. We call a round *corrupted* if its elements are not equal. This procedure successfully simulates n' rounds of a corruption interactive channel with a δ' fraction of errors if for all but $n'\delta'$ of the rounds are corrupted.

The protocol and analysis in this large alphabet setting are similar to the harder case where the alphabet is binary. We cover interactive communication for the binary setting in the next section.

▶ **Theorem 9.** *Assume that n uses of an interactive insertion-deletion channel over an alphabet Σ with a δ fraction of insertions and deletions are given. Using an ε-synchronization string over an alphabet Σ_{syn}, it is possible to simulate $n - 2n\delta(1 + (1-\varepsilon)^{-1})$ uses of an interactive corruption channel over Σ_{sim} with at most a $\frac{2\delta(5-3\varepsilon)}{1-\varepsilon+2\varepsilon\delta-4\delta}$ fraction of symbols corrupted so long as $|\Sigma_{sim}| \times |\Sigma_{syn}| \leq |\Sigma|$ and $\delta < 1/14$.*

2.3 Binary interactive channel simulation

We now show that with the help of synchronization strings, a binary interactive insertion-deletion channel can be used to simulate a binary interactive corruption channel, inducing a $\widetilde{O}(\sqrt{\delta})$ fraction of bit-flips. In this way, the two communicating parties may interact as though they are communicating over a corruption channel. They therefore can employ corruption channel coding schemes while using the simulator as a black box means of converting the insertion-deletion channel to a corruption channel.

The key difference between this simulation and the one-way, large alphabet simulation is that Alice and Bob communicate through C_A and C_B for *blocks* of r rounds, between which C_A and C_B check if they are in sync. Due to errors, there may be times when Alice and Bob are in disagreement about which block, and what part of the block, they are in. C_A and C_B ensure that Alice and Bob are in sync most of the time.

When Alice sends C_A a message from a new block of communication, C_A holds that message and alerts C_B that a new block is beginning. C_A does this by sending C_B a header that is a string consisting of a single one followed by $s-1$ zeros (10^{s-1}). Then, C_A indicates which block Alice is about to start by sending a synchronization symbol to C_B. Meanwhile, when C_B receives a 10^{s-1} string, he listens for the synchronization symbol, makes his best guess about which block Alice is in, and then communicates with Bob and C_A accordingly. This might entail sending dummy blocks to Bob or C_A if he believes that they are in different blocks. To describe the guarantee that our simulation provides, we first define *block corruption channels*.

▶ **Definition 10** (Block Corruption Channel)**.** An n-round adversarial corruption channel is called a (δ, r)-*block corruption channel* if the adversary is restricted to corrupt $n\delta$ symbols which are covered by $n\delta/r$ blocks of r consecutively transmitted symbols.

▶ **Theorem 11.** *Suppose that n rounds of a binary interactive insertion-deletion channel with a δ fraction of insertions and deletions are given. For sufficiently small δ, it is possible to deterministically simulate $n(1-\Theta(\sqrt{\delta\log(1/\delta)}))$ rounds of a binary interactive $(\Theta(\sqrt{\delta\log(1/\delta)}), \sqrt{(1/\delta)\log(1/\delta)})$-block corruption channel between two parties, Alice and Bob, assuming that all substrings of form 10^{s-1} where $s = c\log(1/\delta)$ that Alice sends can be covered by $n\delta$ intervals of $\sqrt{(1/\delta)\log(1/\delta)}$ consecutive rounds. The simulation is performed efficiently if the synchronization string is efficient.*

Proof Sketch. Suppose Alice and Bob communicate via intermediaries C_A and C_B who act according to the algorithm described above. In total, Alice and Bob will attempt to communicate n_s bits to one another over the simulated channel, while C_A and C_B communicate a total of n bits to one another. The adversary is allowed to insert or delete up to $n\delta$ symbols and C_A sends $n/2$ bits, so C_B may receive between $n/2 - n\delta$ and $n/2 + n\delta$ symbols. To prevent C_B from stalling indefinitely, C_B only listens to the first $n(1-2\delta)/2$ bits he receives.

For $r = \sqrt{(1/\delta)\log(1/\delta)}$, we define a *chunk* to be $r_c := (s + |\Sigma_{syn}| + r/2)$ consecutive bits that are sent by C_A to C_B. In particular, a chunk corresponds to a section header and synchronization symbol followed by $r/2$ rounds of messages sent from Alice. As C_B cares about the first $n(1-2\delta)/2$ bits it receives, there are $\frac{n(1-2\delta)}{2r_c}$ chunks in total. Hence, $n_s = \frac{n(1-2\delta)}{2r_c} \cdot r$ since C_B and C_A's communication is alternating.

Note that if Alice sends a substring of form 10^{s-1} in the information part of a chunk, then Bob mistakenly detects a new block. With this in mind, we say a chunk is *good* if:

1. No errors are injected in the chunk or affecting C_B's detection of the chunk's header,
2. C_B correctly decodes the index that C_A sends during the chunk, and
3. C_A does not send a 10^{s-1} substring in the information portion of the chunk.

If a chunk is not good, we call it *bad*. If the chunk is bad because C_B does not decode C_A's index correctly even though they were in sync and no errors were injected, then we call it *decoding-bad*. If it is bad because Alice sends a 10^{s-1} substring, we call it *zero-bad* and otherwise, we call it *error-bad*. Throughout the protocol, C_B uses the variable $\mathtt{I}_B$ to denote the next index of the synchronization string C_B expects to receive and we use $\mathtt{I}_A$ to denote the index of the synchronization string C_A most recently sent. Notice that if a chunk is good and $\mathtt{I}_A = \mathtt{I}_B$, then all messages are correctly conveyed.

We now bound the maximum number of bad chunks that occur over the course of the simulation. Suppose the adversary injects errors into the i^{th} chunk, making that chunk bad. The $(i+1)^{th}$ chunk may also be bad, since Bob may not be listening for 10^{s-1} from C_A when C_A sends them, and therefore may miss the block header. However, if the adversary does not inject any errors into the $(i+1)^{th}$ and the $(i+2)^{th}$ chunk, then the $(i+2)^{th}$ chunk will be good. In effect, a single error may render at most two chunks useless. Since the adversary may inject $n\delta$ errors into the insertion-deletion channel, this means that the number of chunks that are error-bad is at most $2n\delta$. Additionally, by assumption, the number of zero-bad chunks is also at most $n\delta$.

We also must consider the fraction of rounds that are decoding-bad. In order to do this, we appeal to Theorem 6.24 from [19], which guarantees that if an ε-synchronization string of length N is sent over an insertion-deletion channel with a δ' fraction of insertions and deletions, then the receiver will decode the index of the received symbol correctly for all but $2N\delta'/(1-\varepsilon)$ symbols. In this context, N is the number of chunks, i.e. $N = n(1-2\delta)/(2r_c)$, and the fraction of chunks corrupted by errors is $\delta' = 4n\delta/N$. Therefore, the total number of bad chunks is at most $4\delta n + 2N\delta'/(1-\varepsilon) = 4\delta n(3-\varepsilon)/(1-\varepsilon)$.

In the rest of the proof, which is available in the extended version of this paper, we show that all but $12\frac{3-\varepsilon}{1-\varepsilon} \cdot \delta n$ chunks are good chunks and have $\mathtt{I}_A = \mathtt{I}_B$ upon their arrival on Bob's side and we conclude that the simulated channel is a $\left(\frac{3-\varepsilon}{1-\varepsilon}\frac{24\delta}{1-2\delta}r_c, r\right)$-block corruption channel. For the asymptotically optimal choice of $r = \sqrt{(1/\delta)\log(1/\delta)}$, we derive the simulation described in the theorem statement. ◀

The simulation stated in Theorem 11 burdens an additional condition on Alice's stream of bits by requiring it to have a limited number of substrings of form 10^{s-1}. We now introduce a high probability technique to modify a general interactive communication protocol in a way that makes all substrings of form 10^{s-1} in Alice's stream of bits fit into $n\delta$ intervals of length $r = \sqrt{(1/\delta)\log(1/\delta)}$.

▶ **Lemma 12.** *Assume that n rounds of a binary interactive insertion-deletion channel with an oblivious adversary who is allowed to inject $n\delta$ errors are given. There is a pre-coding scheme that can be utilized on top of the simulation introduced in Theorem 11. It modifies the stream of bits sent by Alice so that with probability $1 - e^{-\frac{c-3}{2}n\delta\log\frac{1}{\delta}(1+o(1))}$, all substrings of form 10^{s-1} where $s = c\log(1/\delta)$ in the stream of bits Alice sends over the simulated channel can be covered by $n\delta$ intervals of length $r = \sqrt{(1/\delta)\log(1/\delta)}$. This pre-coding scheme comes at the cost of a $\Theta(\sqrt{\delta\log(1/\delta)})$ fraction of the bits Alice sends through the simulated channel.*

Proof sketch. In the simulation process, each $r/2$ consecutive bits Alice sends forms one of the chunks C_A sends to C_B alongside some headers. The idea of this pre-coding scheme is simple. Alice uses the first $s/2$ data bits (and not the header) of each chunk to share $s/2$ randomly generated bits with Bob (instead of running the interactive protocol) and then both of them extract a string S' of $r/2$ $(s/2)$-wise independent random variables. Then, Alice XORs the rest of data bits she passes to C_A with S' and Bob XORs those bits with S' again to retrieve the original data. In the extended version, we show that this pre-coding scheme guarantees the requirements mentioned in the theorem statement. ◀

Applying this pre-coding for $c \geq 3$ on top of the simulation from Theorem 11 implies the following.

▶ **Theorem 13.** *Suppose that n rounds of a binary interactive insertion-deletion channel with a δ fraction of insertions and deletions performed by an oblivious adversary are given. For sufficiently small δ, it is possible to simulate $n(1 - \Theta(\sqrt{\delta\log(1/\delta)}))$ rounds of a binary interactive $(\Theta(\sqrt{\delta\log(1/\delta)}), \sqrt{(1/\delta)\log 1/\delta})$-block corruption channel between two parties over the given channel. The simulation works with probability $1 - \exp(-\Theta(n\delta\log(1/\delta)))$ and is efficient if the synchronization string is efficient.*

▶ **Lemma 14.** *Suppose that n rounds of a binary, interactive, fully adversarial insertion-deletion channel with a δ fraction of insertions and deletions are given. The pre-coding scheme proposed in Lemma 12 ensures that the stream of bits sent by Alice contains fewer than $n\delta$ substrings of form 10^{s-1} for $s = c\log(1/\delta)$ and $c > 5$ with probability $1 - e^{-\Theta(n\delta\log(1/\delta))}$.*

Theorem 11 and Lemma 14 allow us to conclude that one can perform the simulation stated in Theorem 11 over any interactive protocol with high probability. Note that one can trivially extend the results of Theorems 11 and 13 to one-way binary communication by ignoring the bits Bob sends.

3 Applications: New Interactive Coding Schemes

Efficient Coding Scheme Tolerating 1/44 Fraction of Errors. In this section, we will provide an efficient coding scheme for interactive communication over insertion-deletion channels by first making use of large alphabet interactive channel simulation provided in Theorem 9 to effectively transform the given channel into a simple corruption interactive channel and then use the efficient constant-rate coding scheme of Ghaffari and Haeupler [14] on top of the simulated channel. This will give an efficient constant-rate interactive communication over large enough constant alphabets as described in Theorem 2. We review the following theorem of Ghaffari and Haeupler [14] before proving Theorem 2.

▶ **Theorem 15** (Theorem 1.1 from [14]). *For any constant $\varepsilon > 0$ and n-round protocol Π there is a randomized non-adaptive coding scheme that robustly simulates Π against an adversarial error rate of $\rho \leq 1/4 - \varepsilon$ using $N = O(n)$ rounds, a near-linear $n \log^{O(1)} n$ computational complexity, and failure probability $2^{-\Theta(n)}$.*

Proof of Theorem 2. For a given insertion-deletion interactive channel over alphabet Σ suffering from δ fraction of edit-corruption errors, Theorem 9 enables us to simulate $n - 2n\delta(1 + (1-\varepsilon')^{-1})$ rounds of ordinary interactive channel with $\frac{2\delta(5-3\varepsilon')}{1-\varepsilon'+2\varepsilon'\delta-4\delta}$ fraction of symbol by designating $\log|\Sigma_{syn}|$ bits of each symbol to index simulated channel's symbols with an ε'-synchronization string over Σ_{syn}.

One can employ the scheme of Ghaffari and Haeupler [14] over the simulated channel as long as error fraction is smaller than $1/4$. Note that $\frac{2\delta(5-3\varepsilon')}{1-\varepsilon'+2\delta\varepsilon'-4\delta}\Big|_{\varepsilon'=0} = \frac{10\delta}{1-4\delta} < \frac{1}{4} \Leftrightarrow \delta < \frac{1}{44}$. Hence, as long as $\delta = 1/44 - \varepsilon$ for $\varepsilon > 0$, for small enough $\varepsilon' = O_\varepsilon(1)$, the simulated channel has an error fraction that is smaller than $1/4$. Therefore, by running the efficient coding scheme of Theorem 15 over this simulated channel one gets a constant rate coding scheme for interactive communication that is robust against $1/44 - \varepsilon$ fraction of edit-corruptions. Note that this simulation requires the alphabet size to be large enough to contain synchronization symbols (which can come from a polynomially large alphabet in terms of ε') and also meet the alphabet size requirements of Theorem 15. This requires the alphabet size to be $\Omega_\varepsilon(1)$, i.e., a large enough constant merely depending on ε. The success probability and time complexity are direct consequences of Theorem 15 and Theorem 6.24 from [19]. ◀

Efficient Coding Scheme with Near-Optimal Rate over Small Alphabets. In this section we present another insertion-deletion interactive coding scheme that achieves near-optimal communication efficiency as well as computation efficiency by employing a similar idea as in Section 3.

In order to derive a rate-efficient interactive communication coding scheme over small alphabet insertion-deletion channels, simulations described above can be used to simulate a corruption channel and then the rate-efficient interactive coding scheme for corruption channels introduced by Haeupler [17] can be used on top of the simulated channel.

▶ **Theorem 16** (Interactive Coding against Block Corruption). *By choosing an appropriate block length in the Haeupler [17] coding scheme for oblivious adversaries, one obtains a robust efficient interactive coding scheme for (δ_b, r_b)-block corruption channel with communication rate $1 - \Theta(\sqrt{\delta_b \max\{\delta_b, 1/r_b\}})$ that works with probability $1 - 2^{-\Theta(n\delta_b/r_b)}$.*

Applying the coding scheme of Theorem 16 over the simulation from Theorem 13 implies the following.

▶ **Theorem 17.** *For sufficiently small δ, there is an efficient interactive coding scheme over binary insertion-deletion channels which, is robust against δ fraction of edit-corruptions by an oblivious adversary, achieves a communication rate of $1 - \Theta(\sqrt{\delta \log(1/\delta)})$, and works with probability $1 - 2^{-\Theta(n\delta)}$.*

Moreover, in the extended version, we show that this result is extendable for the fully adversarial setup, as summarized in Theorem 3.

This insertion-deletion interactive coding scheme is, to the best of our knowledge, the first to be computationally efficient, to have communication rate approaching one, and to work over arbitrarily small alphabets.

References

1. Shweta Agrawal, Ran Gelles, and Amit Sahai. Adaptive protocols for interactive communication. In *Information Theory (ISIT), 2016 IEEE International Symposium on*, pages 595–599, 2016.
2. Zvika Brakerski, Yael Tauman Kalai, and Moni Naor. Fast interactive coding against adversarial noise. *Journal of the ACM (JACM)*, 61(6):35, 2014.
3. Zvika Brakerski and Moni Naor. Fast algorithms for interactive coding. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 443–456, 2013.
4. Zvika Brakerski and Yael Tauman Kalai. Efficient interactive coding against adversarial noise. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–166, 2012.
5. Gilles Brassard, Ashwin Nayak, Alain Tapp, Dave Touchette, and Falk Unger. Noisy interactive quantum communication. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 296–305, 2014.
6. Mark Braverman. Towards deterministic tree code constructions. In *Proceedings of the ACM Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 161–167, 2012.
7. Mark Braverman and Klim Efremenko. List and unique coding for interactive communication in the presence of adversarial noise. *SIAM Journal on Computing*, 46(1):388–428, 2017.
8. Mark Braverman, Ran Gelles, Jieming Mao, and Rafail Ostrovsky. Coding for interactive communication correcting insertions and deletions. *IEEE Transactions on Information Theory*, 63(10):6256–6270, 2017.
9. Mark Braverman and Anup Rao. Toward coding for maximum errors in interactive communication. *IEEE Transactions on Information Theory*, 60(11):7248–7255, 2014.
10. Klim Efremenko, Gelles Ran, and Haeupler Bernhard. Maximal noise in interactive communication over erasure channels and channels with feedback. *IEEE Transactions on Information Theory*, 62(8):4575–4588, 2016.
11. Matthew Franklin, Ran Gelles, Rafail Ostrovsky, and Leonard J. Schulman. Optimal coding for streaming authentication and interactive communication. *IEEE Transactions on Information Theory*, 61(1):133–145, 2015.
12. Ran Gelles, Ankur Moitra, and Amit Sahai. Efficient and explicit coding for interactive communication. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 768–777, 2011.
13. Ran Gelles, Ankur Moitra, and Amit Sahai. Efficient coding for interactive communication. *IEEE Transactions on Information Theory*, 60(3):1899–1913, 2014.
14. Mohsen Ghaffari and Bernhard Haeupler. Optimal error rates for interactive coding ii: Efficiency and list decoding. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 394–403, 2014.

15 Mohsen Ghaffari, Bernhard Haeupler, and Madhu Sudan. Optimal error rates for interactive coding i: Adaptivity and other settings. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, pages 794–803, 2014.

16 Venkatesan Guruswami and Carol Wang. Deletion codes in the high-noise and high-rate regimes. *IEEE Transactions on Information Theory*, 63(4):1961–1970, 2017.

17 Bernhard Haeupler. Interactive channel capacity revisited. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 226–235, 2014.

18 Bernhard Haeupler and Ran Gelles. Capacity of interactive communication over erasure channels and channels with feedback. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1296–1311, 2015.

19 Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: Codes for insertions and deletions approaching the singleton bound. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, 2017.

20 Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: Explicit constructions, local decoding, and applications. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, 2018.

21 Gillat Kol and Ran Raz. Interactive channel capacity. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, pages 715–724, 2013.

22 Leonard J. Schulman. Communication on noisy channels: A coding theorem for computation. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 724–733, 1992.

23 Leonard J. Schulman. Deterministic coding for interactive communication. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, pages 747–756, 1993.

24 Leonard J. Schulman. Postscript to "Coding for interactive communication". [Online; accessed 17-March-2017], 2003. URL: `http://users.cms.caltech.edu/~schulman/Papers/intercodingpostscript.txt`.

25 Alexander A. Sherstov and Pei Wu. Optimal interactive coding for insertions, deletions, and substitutions. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 240–251, 2017.

Synchronization Strings: List Decoding for Insertions and Deletions

Bernhard Haeupler[1]
Carnegie Mellon University, Pittsburgh, PA, USA
haeupler@cs.cmu.edu

Amirbehshad Shahrasbi[2]
Carnegie Mellon University, Pittsburgh, PA, USA
shahrasbi@cs.cmu.edu

Madhu Sudan[3]
Harvard University, Cambridge, MA, USA
madhu@cs.harvard.edu

Abstract

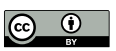
We study codes that are list-decodable under insertions and deletions ("insdel codes"). Specifically, we consider the setting where, given a codeword x of length n over some finite alphabet Σ of size q, $\delta \cdot n$ codeword symbols may be adversarially deleted and $\gamma \cdot n$ symbols may be adversarially inserted to yield a corrupted word w. A code is said to be list-decodable if there is an (efficient) algorithm that, given w, reports a small list of codewords that include the original codeword x. Given δ and γ we study what is the rate R for which there exists a constant q and list size L such that there exist codes of rate R correcting δ-fraction insertions and γ-fraction deletions while reporting lists of size at most L.

Using the concept of *synchronization strings*, introduced by the first two authors [Proc. STOC 2017], we show some surprising results. We show that for every $0 \leq \delta < 1$, every $0 \leq \gamma < \infty$ and every $\varepsilon > 0$ there exist codes of rate $1 - \delta - \varepsilon$ and constant alphabet (so $q = O_{\delta,\gamma,\varepsilon}(1)$) and sub-logarithmic list sizes. Furthermore, our codes are accompanied by efficient (polynomial time) decoding algorithms. We stress that the fraction of insertions can be arbitrarily large (more than 100%), and the rate is independent of this parameter. We also prove several tight bounds on the parameters of list-decodable insdel codes. In particular, we show that the alphabet size of insdel codes needs to be exponentially large in ε^{-1}, where ε is the gap to capacity above. Our result even applies to settings where the unique-decoding capacity equals the list-decoding capacity and when it does so, it shows that the alphabet size needs to be exponentially large in the gap to capacity. This is sharp contrast to the Hamming error model where alphabet size polynomial in ε^{-1} suffices for unique decoding. This lower bound also shows that the exponential dependence on the alphabet size in previous works that constructed insdel codes is actually necessary!

Our result sheds light on the remarkable asymmetry between the impact of insertions and deletions from the point of view of error-correction: Whereas deletions cost in the rate of the code, insertion costs are borne by the adversary and not the code! Our results also highlight the dominance of the model of insertions and deletions over the Hamming model: A Hamming error is equal to one insertion and one deletion (at the same location). Thus the effect of δ-fraction Hamming errors can be simulated by δ-fraction of deletions and δ-fraction of insertions — but insdel codes can deal with much more insertions without loss in rate (though at the price of higher alphabet size).

2012 ACM Subject Classification Mathematics of computing → Coding theory

[1] Supported in part by NSF grants CCF-1527110, CCF-1618280 and NSF CAREER award CCF-1750808.
[2] Supported in part by NSF grants CCF-1527110, CCF-1618280 and NSF CAREER award CCF-1750808.
[3] Supported in part by a Simons Investigator Award and NSF Awards CCF 1565641 and CCF 1715187.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 76; pp. 76:1–76:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Keywords and phrases List Decoding, Insertions and Deletions, Synchronization Strings

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.76

Related Version An extended version of this article is available at `https://arxiv.org/pdf/1802.08663.pdf`.

1 Introduction

We study the complexity of "insdel coding", i.e., codes designed to recover from insertion and deletion of characters, under the model of "list-decoding", i.e., when the decoding algorithm is allowed to report a (short) list of potential codewords that is guaranteed to include the transmitted word if the number of errors is small enough. Recent work by the first two authors and collaborators [12] has shown major progress leading to tight, or nearly tight, bounds on central parameters of codes (with efficient encoding and decoding algorithms as well) under the setting of *unique* decoding. However the list-decoding versions of these questions were not explored previously. Our work complements the previous studies by exploring list-decoding. In the process our results also reveal some striking features of the insdel coding problem that were not exposed by previous works. To explain some of this, we introduce our model and lay out some of the context below.

1.1 Insdel Coding and List Decoding

We use the phrase "insdel coding" to describe the study of codes that are aimed to recover from *insertions* and *deletions*. The principal question we ask is "what is the *rate* of a code that can recover from γ fraction insertions and δ fraction deletions over a sufficiently large alphabet?". Once the answer to this question is determined we ask how small an alphabet suffices to achieve this rate. We define the terms "rate", "alphabet", and "recovery" below.

An insdel *encoder* over alphabet Σ of block length n is an injective function $E : \Sigma^k \to \Sigma^n$. The associated "code" is the image of the function C. The rate of a code is the ratio k/n. We say that an insdel code C is $(\gamma, \delta, L(n))$-list-decodable if there exists a function $D : \Sigma^* \to 2^C$ such that $|D(w)| \leq L(n)$ for every $w \in \Sigma^*$ and for every codeword $x \in C$ and every word w obtained from x be $\delta \cdot n$ deletions of characters in x followed by $\gamma \cdot n$ insertions, it is the case that $x \in D(w)$. In other words the list-decoder D outputs a list of at most $L(n)$ codewords that is guaranteed to include the transmitted word x if the received word w is obtained from x by at most δ-fraction deletions and γ-fraction insertions. Our primary quest in this paper is the largest rate R for which there exists an alphabet of size $q \triangleq |\Sigma|$ and an infinite family of insdel codes of rate at least R, that are $(\gamma, \delta, L(n))$-list-decodable. Of course we are interested in results where $L(n)$ is very slowly growing with n (if at all). In all results below we get $L(n)$ which is polynomially large in terms of n. Furthermore, when a given rate is achievable we seek codes with efficient encoder and decoder (i.e., the functions E and D are polynomial time computable). Finally we also explore the dependence of the rate on the alphabet size (or vice versa).

Previous Work. Insdel coding was first studied by Levenshtein [18] and since then many bounds and constructions for such codes have been given. With respect to unique decoding, Schulman and Zuckerman [21] gave the first construction of efficient insdel codes over a constant alphabet with a (small) constant relative distance and a (small) constant rate in 1999. Guruswami and Wang [9] gave the first efficient codes over fixed alphabets to correct a

deletion fraction approaching 1, as well as efficient binary codes to correct a small constant fraction of deletions with rate approaching 1. A follow-up work gave new and improved codes with similar rate-distance tradeoffs which can be efficiently decoded from insertions and deletions [5]. Finally, [12] gave codes that can correct δ fraction of synchronization errors with a rate approaching $1-\delta-\varepsilon$ for any $\varepsilon > 0$.

A recent work by Wachter-Zeh [23] considers insdel coding with respect to list decoding and provides Johnson-like upper-bounds for insertions and deletions, i.e., bounds on the list size in terms of the minimum edit-distance of a given code. Moreover, for Varshamov-Tenengolts codes, [23] presents lower bounds on the maximum list size as well as a list-decoding algorithm against a constant number of insertions and deletions.

Several other variants of the insdel coding problem have been studied in the previous work and are summarized by the following surveys [22, 20, 19].

1.2 Our Results

We now present our results on the rate and alphabet size of insdel coding under list-decoding. Two points of contrast that we use below are corresponding bounds in (1) the Hamming error setting for list-decoding and (2) the insdel coding setting with unique-decoding.

1.2.1 Rate Under List Decoding

Our main theorem for list-decoding shows that, given $\gamma, \delta, \varepsilon \geq 0$ there is a $q = q_{\varepsilon,\gamma}$ and a slowly growing function $L = L_{\varepsilon,\gamma}(n)$ such that there are q-ary insdel codes that achieve a rate of $1-\delta-\varepsilon$ that are $(\gamma, \delta, L(n))$-list decodable. Furthermore the encoding and decoding are efficient! The formal statement of the main result is as follows.

▶ **Theorem 1.** *For every $0 < \delta, \varepsilon < 1$ and $\gamma > 0$, there exist a family of list-decodable insdel codes that can protect against δ-fraction of deletions and γ-fraction of insertions and achieves a rate of at least $1-\delta-\varepsilon$ or more over an alphabet of size $\left(\frac{\gamma+1}{\varepsilon^2}\right)^{O\left(\frac{\gamma+1}{\varepsilon^3}\right)} = O_{\gamma,\varepsilon}(1)$. These codes are list-decodable with lists of size $L_{\varepsilon,\gamma}(n) = \exp(\exp(\exp(\log^* n)))$, and have polynomial time encoding and decoding complexities.*

The rate in the theorem above is immediately seen to be optimal even for $\gamma = 0$. In particular an adversary that deletes the last $\delta \cdot n$ symbols already guarantees an upper bound on the rate of $1-\delta$.

We now contrast the theorem above with the two contrasting settings listed earlier. Under unique decoding the best possible rate that can be achieved with δ-fraction deletions and γ-fraction insertions is upper bounded by $1-(\gamma+\delta)$. Matching constructions have been achieved, only recently, by Haeupler and Shahrasbi [12]. In contrast our rate has no dependence on γ and thus dominates the above result. The only dependence on γ is in the alphabet size and list-size and we discuss the need for this dependence later below.

We now turn to the standard "Hamming error" setting: Here an adversary may change an arbitrary δ-fraction of the codeword symbols. In this setting it is well-known that given any $\varepsilon > 0$, there are constants $q = q(\varepsilon)$ and $L = L(\varepsilon)$ and an infinite family of q-ary codes of rate at least $1-\delta-\varepsilon$ that are list-decodable from δ fraction errors with list size at most L. In a breakthrough from the last decade, Guruswami and Rudra [6] showed explicit codes that achieve this with efficient algorithms. The state-of-the-art results in this field yield list size $L(n) = o(\log^{(r)} n)$ for any integer r where $\log^{(r)}$ is the rth iterated logarithm and alphabet size $2^{\tilde{O}(\varepsilon^{-2})}$ [11], which are nearly optimal.

The Hamming setting with δ-fraction errors is clearly a weaker setting than the setting with δ-fraction deletions and $\gamma \geq \delta$ fraction of insertions in that an adversary of the latter kind can simulate the former. (A Hamming error is a deletion followed by an insertion at the same location.) The insdel setting is thus stronger in two senses: it allows $\gamma > \delta$ and gives greater flexibility to the adversary in choosing locations of insertions and deletions. Yet our theorem shows that the stronger adversary can still be dealt with, without qualitative changes in the rate. The only difference is in the dependence of q and L on γ, which we discuss next.

We briefly remark at this stage that, while our result simultaneously "dominates" the results of Haeupler and Shahrasbi [12] as well as Guruswami and Rudra [6], this happens because we use their results in our work. We elaborate further on this in Section 3. Indeed our first result (see Theorem 13) shows how we can obtain Theorem 1 by using capacity achieving "list-recoverable codes" in combination with synchronization strings in a modular fashion.

We believe that using the codes from Theorem 1 in the construction of long-distance synchronization strings from [13] will give synchronization strings that allow us to reduce the decoding complexity of insdel codes of Theorem 1 and [12] to near-linear time.

1.2.2 Rate versus Alphabet Size

We now turn to understanding how large the alphabet size needs to be as a function of δ, ε and γ. We consider two extreme cases, first with only deletions (i.e., $\gamma = 0$ and then with only insertions (i.e., with $\delta = 0$).

We start first with the insertion-only setting. We note here that one cannot hope to find a constant rate family of codes that can protect n symbols out of an alphabet of size q against $(q-1)n$ many insertions or more. This is so since, with $(q-1)n$ insertions, one can turn any string $y \in [1..q]^n$ into the fixed sequence $1, 2, \cdots, q, 1, 2, \cdots, q, \cdots, 1, 2, \cdots, q$ by simply inserting $q-1$ many symbols around each symbol of y to construct a $1, \cdots, q$ there. Hence, Theorem 2 only focuses on codes that protect n rounds of communication over an alphabet of size q against γn insertions for $\gamma < q - 1$.

▶ **Theorem 2.** *Any list-decodable family of codes $\mathcal{C}$ that protects against γ fraction of insertions for some $\gamma < q-1$ and guarantee polynomially-large list size in terms of block length cannot achieve a rate R that is strictly larger than* $1 - \log_q(\gamma+1) - \gamma\left(\log_q \frac{\gamma+1}{\gamma} - \log_q \frac{q}{q-1}\right)$.

In particular, the theorem asserts that if the code has rate $R = 1 - \varepsilon$, then its alphabet size must be exponentially large in $1/\varepsilon$, namely, $q \geq (\gamma+1)^{1/\varepsilon}$.

Next, we turn to the deletion-only case. Here again we note that no constant rate q-ary code can protect against $\delta \geq \frac{q-1}{q}$ fraction of deletions since such a large fraction of deletions may remove all but the most frequent symbol of codewords. Therefore, Theorem 3 below only concerns codes that protect against $\delta \leq \frac{q-1}{q}$ fraction of deletions.

▶ **Theorem 3.** *Any list-decodable family of insdel codes that protect against δ-fraction of deletions (and no insertions) for some $0 \leq \delta < \frac{q-1}{q}$ that are list-decodable with polynomially-bounded list size has rate R upper bounded as below:*

- $R \leq f(\delta) \triangleq (1-\delta)\left(1 - \log_q \frac{1}{1-\delta}\right)$ *where $\delta = \frac{d}{q}$ for some integer d.*
- $R \leq (1 - q\delta')f\left(\frac{d}{q}\right) + q\delta' f\left(\frac{d+1}{q}\right)$ *where $\delta = \frac{d}{q} + \delta'$ for some integer d and $0 \leq \delta' < \frac{1}{q}$.*

In particular if $\delta = d/q$ for integer d and rate is $1 - \delta - \varepsilon$ then the theorem above asserts that $q \geq \left(\frac{1}{1-\delta}\right)^{\frac{1-\delta}{\varepsilon}}$, or in other words q must be exponentially large in $1/\varepsilon$. Indeed such a statement is true for all δ as asserted in the corollary below. (A detailed proof is available in the extended version.)

▶ **Corollary 4.** *There exists a function $f : (0,1) \to (0,1)$ such that any family of insdel codes that protects against δ-fraction of deletions with polynomially bounded list sizes and has rate $1 - \delta - \varepsilon$ must have alphabet size $q \geq \exp\left(\frac{f(\delta)}{\varepsilon}\right)$.*

Implications for Unique Decoding. Even though the main thrust of this paper is list-decoding, Corollary 4 also has implications for unique-decoding. (This turns out to be a consequence of the fact that the list-decoding radius for deletions-only equals the unique-decoding radius for the same fraction of deletions.) We start by recalling the main result of Haeupler and Shahrasbi [12]: Given any $\alpha, \varepsilon > 0$ there exists a code of rate $1 - \alpha - \varepsilon$ over an alphabet of size $q = \exp(1/\varepsilon)$ that *uniquely* decodes from any α-fraction synchronization errors, i.e., from γ-fraction insertions and δ-fraction deletions for any pair $0 \leq \gamma, \delta$ satisfying $\gamma + \delta \leq \alpha$. Furthermore, this is the best possible rate one can achieve for α-fraction synchronization error. (See the extended version for a more detailed description with proof.)

Till now this exponential dependence of the alphabet size on ε was unexplained. This is also in sharp contrast to the Hamming error setting, where codes are known to get ε close to unique decoding capacity (half the "Singleton bound" on the distance of code) with alphabets of size polynomial in $1/\varepsilon$. Indeed given this contrast one may be tempted to believe that the exponential growth is a weakness of the "synchronization string" approach of Haeupler and Shahrasbi [12]. But Corollary 4 actually shows that an exponential bound is necessary. We state this result for completeness even though it is immediate from the Corollary above, to stress its importance in understanding the nature of synchronization errors.

▶ **Corollary 5.** *There exists a function $f : (0,1) \to (0,1)$ such that for every $\alpha, \varepsilon > 0$ every family of insdel codes of rate $1 - \alpha - \varepsilon$ that protects against α-fraction of synchronization errors with unique decoding must have alphabet size $q \geq \exp\left(\frac{f(\delta)}{\varepsilon}\right)$.*

Corollary 5 follows immediately from Corollary 4 by setting $\delta = \alpha$ and $\gamma = 0$ (so we get to the zero insertion case) and noticing that a unique-decoding insdel code for α-fraction synchronization error is also a list-decoding insdel code for δ-fractions of deletions (and no insertions). The alphabet size lower bound for the latter is also an alphabet size lower bound for the former.

1.2.3 Analysis of Random Codes

Finally, in Section 5, we provide an analysis of random codes and compute the rates they can achieve while maintaining list-decodability against insertions and deletions. Such rates are essentially lower-bounds for the capacity of insertion and deletion channels and can be compared against the upper-bounds provided in Section 4.

Theorem 6 shows that the family of random codes over an alphabet of size q can, with high probability, protect against δ-fraction of deletions for any $\delta < 1 - 1/q$ up to a rate of $1-(1-\delta)\log_q \frac{1}{1-\delta} - \delta \log_q \frac{1}{\delta} - \delta \log_q(q-1) = 1 - H_q(\delta)$ using list decoding with super-constant list sizes in terms of their block length where H_q represents the q-ary entropy function.

▶ **Theorem 6.** *For any alphabet of size q and any $0 \leq \delta < \frac{q-1}{q}$, the family of random codes with rate $R < 1 - (1-\delta)\log_q \frac{1}{1-\delta} - \delta \log_q \frac{1}{\delta} - \delta \log_q(q-1) - \frac{1-\delta}{l+1}$ is list-decodable with list size of l from any δ fraction of deletions with high probability. Further, the family of random deletion-codes with rate $R > 1 - (1-\delta)\log_q \frac{1}{1-\delta} - \delta \log_q \frac{1}{\delta} - \delta \log_q(q-1)$ is not list-decodable with high probability.*

Further, Theorem 7 shows that the family of random block codes over an alphabet of size q can, with high probability, protect against γ fraction of insertions for any $\gamma < q - 1$ up to a rate of $1 - \log_q(\gamma + 1) - \gamma \log_q \frac{\gamma+1}{\gamma}$ using list decoding with super-constant list sizes in terms of block length.

▶ **Theorem 7.** *For any alphabet of size q and any $\gamma < q - 1$, the family of random codes with rate $R < 1 - \log_q(\gamma + 1) - \gamma \log_q \frac{\gamma+1}{\gamma} - \frac{\gamma+1}{l+1}$ is list-decodable with a list size of l from any γn insertions with high probability.*

2 Definitions and Preliminaries

2.1 Synchronization Strings

In this section, we briefly recapitulate synchronization strings, introduced by Haeupler and Shahrasbi [12] and further studied in [14, 13]. We will review important definitions and techniques from [12] that will be of use throughout this paper.

Synchronization strings are recently introduced mathematical objects that turn out to be useful tools to overcome synchronization errors, i.e., symbol insertion and symbol deletion errors. The general idea employed in [12, 14] to obtain resilience against synchronization errors in various communication setups is *indexing* each symbol of the communication with symbols of a synchronization string and then *guessing* the actual position of received symbols on the other side using indices. [12] provides a variety of different guessing strategies that guarantee a large number of correct guesses and then overcome the incorrect guesses by utilizing classic error correcting codes. As a matter of fact, synchronization strings essentially translate synchronization errors into ordinary Hamming type errors which are strictly easier to handle. We now proceed to review some of the above-mentioned definitions and techniques more formally.

Suppose that two parties are communicating over a channel that suffers from α-fraction of insertions and deletions and one of the parties sends a pre-shared string S of length n to the other one. A distorted version of S will arrive at the receiving end that we denote by S'. A symbol $S[i]$ is called to be a *successfully transmitted* symbol if it is not removed by the adversary. A *decoding algorithm* on the receiving side is an algorithm that, for any received symbol, guesses its actual position in S by either returning a number in $[1..n]$ or $\top$ which means the algorithm is not able to guess the index. For such a decoding algorithm, a successfully transmitted symbol whose index is not guessed correctly by the decoding algorithm is called a *misdecoding*.

Haeupler and Shahrasbi [12] introduce synchronization strings and find several decoding algorithms for them providing strong misdecoding guarantees and then design insertion-deletion codes based on those decoding algorithms. As details of those algorithms are not relevant to this paper we avoid further discussion of those techniques. To conclude this section we introduce ε-synchronization strings and an important property of them.

▶ **Definition 8** (ε-synchronization string)**.** String $S \in \Sigma^n$ is an ε-synchronization string if for every $1 \leq i < j < k \leq n + 1$ we have that $\text{EditDistance}\left(S[i, j), S[j, k)\right) > (1 - \varepsilon)(k - i)$ where $\text{EditDistance}(x, y)$ is the smallest number of insertions and deletions needed to convert x to y.

The key idea in the construction of insdel codes in [12] is to index an error correcting code with a synchronization string. Here we provide a formal definition of indexing operation.

▶ **Definition 9** (Indexing). The operation of indexing code $\mathcal{C}$ with block length n and String S of length n is to simply replace each codeword $w_1, w_2, \cdots, w_n$ with $(w_1, s_1), (w_2, s_2), \cdots, (w_n, s_n)$. Clearly, this operations expands the alphabet of the code.

It is shown in [12, 13] that ε-synchronization strings exist over alphabets of sizes polynomially large in terms of ε^{-1} and can be efficiently constructed. An important property of ε-synchronization strings discussed in [12] is the self matching property defined as follows.

▶ **Definition 10** (ε-self-matching property). String S satisfies ε-self-matching property if for any two sequences of indices $1 \leq a_1 < a_2 < \cdots < a_k \leq |S|$ and $1 \leq b_1 < b_2 < \cdots < b_k \leq |S|$ that satisfy $S[a_i] = S[b_i]$ and $a_i \neq b_i$, k is not larger than $\varepsilon|S|$.

In the end, we review the following theorem from [12] that shows the close connection between synchronization string property and the self-matching property.

▶ **Theorem 11** (Theorem 6.4 from [12]). *If S is an ε-synchronization string, then all substrings of S satisfy ε-self-matching property.*

2.2 List Recoverable Codes

A code $\mathcal{C}$ given by the encoding function $\mathcal{E} : \Sigma^{nr} \to \Sigma^n$ is called to be (α, l, L)-list recoverable if for any collection of n sets $S_1, S_2, \cdots, S_n \subset \Sigma$ of size l or less, there are at most L codewords for which more than αn elements appear in the list that corresponds to their position, i.e.,

$$|\{x \in \mathcal{C} \mid |\{i \in [n] \mid x_i \in S_i\}| \geq \alpha n\}| \leq L.$$

The study of list-recoverable codes was inspired by Guruswami and Sudan's list-decoder for Reed-Solomon codes [7]. Since then, list-recoverable codes have became a very useful tool in coding theory [1, 2, 3, 4] and there have been a variety of constructions provided for them by several works [6, 8, 10, 17, 16, 11, 15]. In this paper, we will make use of the following capacity-approaching polynomial-time list-recoverable codes given by Hemenway, Ron-Zewi, and Wootters [15] that is obtained by altering the approach of Guruswami and Xing [10].

▶ **Theorem 12** (Hemenway et. al. [15, Theorem A.7]). *Let q be an even power of a prime, and choose $l, \epsilon > 0$, so that $q \geq \epsilon^{-2}$. Choose $\rho \in (0, 1)$. There is an $m_{min} = O(l \log_q(l/\epsilon)/\epsilon^2)$ so that the following holds for all $m \geq m_{min}$. For infinitely many n (all n of the form $q^{e/2}(\sqrt{q} - 1)$ for any integer e), there is a deterministic polynomial-time construction of an F_q-linear code $C : \mathbb{F}_{q^m}^{\rho n} \to \mathbb{F}_{q^m}^n$ of rate ρ and relative distance $1 - \rho - O(\epsilon)$ that is $(1 - \rho - \epsilon, l, L)$-list-recoverable in time $\text{poly}(n, L)$, returning a list that is contained in a subspace over $\mathbb{F}_q$ of dimension at most $\left(\frac{l}{\epsilon}\right)^{2^{\log^*(mn)}}$.*

3 List Decoding for Insertions and Deletions

In this section, we prove Theorem 1 by constructing a list-decodable code of rate $1 - \delta - \varepsilon$ that provides resilience against $0 < \delta < 1$ fraction of deletions and γ fraction of insertions over a constant-sized alphabet. Our construction heavily relies on the following theorem that, in the same fashion as [12], uses the technique of indexing an error correcting code with a synchronization string to convert a given list-recoverable code into an insertion-deletion code.

▶ **Theorem 13.** *Let $\mathcal{C} : \Sigma^{nR} \to \Sigma^n$ be a (α, l, L)-list recoverable code with rate R, encoding complexity T_{Enc} and decoding complexity complexity T_{Dec}. For any $\varepsilon > 0$ and $\gamma \leq \frac{l\varepsilon}{2} - 1$, by indexing $\mathcal{C}$ with an $\frac{\varepsilon^2}{4(1+\gamma)}$-synchronization string, one can obtain an L-list decodable*

insertion-deletion code $\mathcal{C}' : \Sigma^{nr} \to [\Sigma \times \Gamma]^n$ *that corrects from* $\delta < 1 - \alpha - \varepsilon$ *fraction of deletions and* γ *fraction of insertions where* $|\Gamma| = \left(\varepsilon^2/(1+\gamma)\right)^{-O(1)}$. $\mathcal{C}'$ *is encodable and decodable in* $O(T_{Enc} + n)$ *and* $O(T_{Dec} + n^2(1+\gamma^2)/\varepsilon)$ *time respectively.*

We take two major steps to prove Theorem 13. In the first step (Theorem 15), we use the synchronization string indexing technique from [12] and show that by indexing the symbols that are conveyed through an insertion-deletion channel with symbols of a synchronization string, the receiver can make *lists* of candidates for any position of the sent string such that $1-\delta-\varepsilon$ fraction of lists are guaranteed to contain the actual symbol sent in the corresponding step and the length of the lists is guaranteed to be smaller than some constant $O_{\gamma,\varepsilon}(1)$.

In the second step, we use list-recoverable codes on top of the indexing scheme to obtain a list decoding using lists of candidates for each position produced by the former step.

We start by the following lemma that directly implies the first step stated in Theorem 15.

▶ **Lemma 14.** *Assume that a sequence of* n *symbols denoted by* $x_1 x_2 \cdots x_n$ *is indexed with an* ε*-synchronization string and is communicated through a channel that suffers from up to* δn *deletions for some* $0 \leq \delta < 1$ *and* γn *insertions. Then, on the receiving end, it is possible to obtain* n *lists* $A_1, \cdots, A_n$ *such that, for any desired integer* K*, for at least* $n \cdot \left(1 - \delta - \frac{1+\gamma}{K} - K \cdot \varepsilon\right)$ *of them,* $x_i \in A_i$*. All lists contain up to* K *elements and the average list size is at most* $1+\gamma$*. These lists can be computed in* $O\left(K(1+\gamma)n^2\right)$ *time.*

Proof. The decoding algorithm we propose to obtain the lists that satisfy the guarantee promised in the statement is the global algorithm introduced in Theorem 6.14 of Haeupler and Shahrasbi [12].

Let S be the ε-synchronization string used for indexing and S' be the index portion of the received string on the other end. Note that S is pre-shared between the sender and the receiver. The decoding algorithm starts by finding a longest common substring M_1 between S and S' and adding the position of any matched element from S' to the list that corresponds to its respective match from side S. Then, it removes every symbol that have been matched from S' and repeats the previous step by finding another longest common subsequence M_2 between S and the remaining elements of S'. This procedure is repeated K times to obtain $M_1, \cdots, M_K$. This way, lists A_i are formed by including every element in S' that is matched to $S[i]$ in any of $M_1, \cdots, M_K$.

A_i contains the actual element that corresponds to $S[i]$, denoted by $S'[j]$, if and only if $S[i]$ is successfully transmitted (i.e., not removed by the adversary), appears in one of M_ks, and matches to $S[i]$ in M_k. Hence, there are three scenarios under which A_i does not contain its corresponding element $S[i]$.

1. $S[i]$ gets deleted by the adversary.
2. $S[i]$ is successfully transmitted but, as $S'[j]$ on the other side, it does not appear on any of M_ks.
3. $S[i]$ is successfully transmitted and, as $S'[j]$ on the other side, it appears in some M_k although it is matched to another element of S.

The first case happens for at most δn elements as adversary is allowed to delete up to δn many elements.

To analyze the second case, note that the sizes of M_ks descend as k grows since we pick the longest common subsequence in each step. If by the end of this procedure p successfully transmitted symbols are still not matched in any of the matchings, they form a common subsequence of size p between S and the remainder of S'. This leads to the fact that

$|M_1| + \cdots + |M_K| \geq K \cdot p$. As $|M_1| + \cdots + |M_K|$ cannot exceed $|S'|$, we have $p \leq |S'|/K$. This bounds above the number of symbols falling into the second category by $|S'|/K$.

Finally, as for the third case, we draw the reader's attention to the fact that each successfully transmitted $S[i]$ which arrives at the other end as $S'[j]$ and mistakenly gets matched to another element of S like $S[k]$ in some M_t, implies that $S[i] = S[k]$. We call the pair (i, k) *a pair of similar elements in S implied by M_t*. Note that there is an actual monotone matching M' from S to S' that corresponds to adversary's actions. As M_t and M' are both monotone, the set of similar pairs in S implied by M_t is a self-matching in S. As stated in Theorem 11, the number of such pairs cannot exceed $n\varepsilon$. Therefore, there can be at most $n\varepsilon$ successfully transmitted symbols that get mistakenly matched in M_t for any t. Hence, the number of elements falling into the third category is at most $nK\varepsilon$.

Summing up all above-mentioned bounds gives that the number of bad lists can be bounded above by $n\delta + \frac{|S'|}{K} + nK\varepsilon \leq n\left(\delta + \frac{1+\gamma}{K} + K\varepsilon\right)$. This proves the list quality guarantee. As proposed decoding algorithm computes longest common substring K many times between two strings of length n and $(1+\gamma)n$ or less, it will run in $O(K(1+\gamma)\cdot n^2)$ time. ◀

▶ **Theorem 15.** *Suppose that n symbols denoted by $x_1, x_2, \cdots, x_n$ are being communicated through a channel suffering from up to δn deletions for some $0 \leq \delta < 1$ and γn insertions for some constant $\gamma \geq 0$. If one indexes these symbols with an $\varepsilon' = \frac{\varepsilon^2}{4(1+\gamma)}$-synchronization string, then, on the receiving end, it is possible to obtain n lists $A_1, \cdots, A_n$ of size $2(1+\gamma)/\varepsilon$ such that, for at least $n \cdot (1-\delta-\varepsilon)$ of them, $x_i \in A_i$. These lists can be computed in $O\left(n^2(1+\gamma)^2/\varepsilon\right)$ time.*

Proof. Using an $\varepsilon' = \frac{\varepsilon^2}{4(1+\gamma)}$-synchronization string in the statement of Lemma 14 and choosing $K = \frac{2(1+\gamma)}{\varepsilon}$ directly gives that the runtime is $O\left(n^2(1+\gamma)^2/\varepsilon\right)$ and list hit ratio is at least $n \cdot \left(1 - \delta - \frac{1+\gamma}{K} - K \cdot \varepsilon'\right) = n \cdot (1 - \delta - \varepsilon/2 - \varepsilon/2) = n \cdot (1 - \delta - \varepsilon)$ ◀

Theorem 15 facilitates the conversion of list-recoverable error correcting codes into list-decodable insertion-deletion codes as stated in Theorem 13.

Proof of Theorem 13. To prove this, we simply index code $\mathcal{C}$, entry by entry, with an $\varepsilon' = \frac{\varepsilon^2}{4(1+\gamma)}$ synchronization string. In the decoding procedure, according to Theorem 15, the receiver can use the index portion of the received symbol to maintain lists of up to $2(1+\gamma)/\varepsilon \leq l$ candidates for each position of the sent codeword of $\mathcal{C}$ so that $1 - \delta - \varepsilon > \alpha$ fraction of those contain the actual corresponding sent message. Having such lists, the receiver can use the decoding function of $\mathcal{C}$ to obtain an L-list-decoding for $\mathcal{C}'$. Finally, the alphabet size and encoding complexity follow from the fact that synchronization strings over alphabets of size ${\varepsilon'}^{-O(1)}$ can be constructed in linear time [12, 13]. ◀

One can use any list-recoverable error correcting code to obtain insertion-deletion codes according to Theorem 13. In particular, using the efficient capacity-approaching list-recoverable code introduced by Hemenway, Ron-Zewi, and Wootters [15], one obtains the insertion-deletion codes as described in Theorem 1.

Proof of Theorem 1. By setting parameters $\rho = 1 - \delta - \frac{\varepsilon}{2}$, $l = \frac{2(\gamma+1)}{\varepsilon}$, and $\epsilon = \frac{\varepsilon}{4}$ in Theorem 12, one can obtain a family of codes $\mathcal{C}$ that achieves rate $\rho = 1 - \delta - \frac{\varepsilon}{2}$ and is (α, l, L)-recoverable in polynomial time for $\alpha = 1 - \delta - \varepsilon/4$ and some $L = \exp\left(\exp\left(\exp\left(\log^* n\right)\right)\right)$ (by treating γ and ε as constants). Such family of codes can be found over an alphabet $\Sigma_{\mathcal{C}}$ of size $q = (l/\epsilon)^{O(l/\epsilon^2)} = \left(\frac{\gamma+1}{\varepsilon^2}\right)^{O\left(\frac{\gamma+1}{\varepsilon^3}\right)} = O_{\gamma,\varepsilon}(1)$ or infinitely many integer numbers larger than q.

Plugging this family of codes into the indexing scheme from Theorem 13 by choosing the parameter $\varepsilon' = \frac{\varepsilon}{4}$, one obtains a family of codes that can recover from $1-\alpha-\varepsilon' = 1-(1-\delta-\varepsilon/4)-\varepsilon/4 = \delta$ fraction of deletions and γ-fraction of insertions and achieves a rate of $\frac{1-\delta-\varepsilon/2}{1+\log|\Sigma_S|/\log|\Sigma_{\mathcal{C}}|}$ which, by taking $|\Sigma_{\mathcal{C}}|$ large enough in terms of ε, is larger than $1-\delta-\varepsilon$. As $\mathcal{C}$ is encodable and decodable in polynomial time, the encoding and decoding complexities of the indexed code will be polynomial as well. ◀

▶ Remark. We remark that by using capacity-approaching near-linear-time list-recoverable code introduced in Theorem 7.1 of Hemenway, Ron-Zewi, and Wootters [15] in the framework of Theorem 13, one can obtain similar list-decodable insertion-deletion codes as in Theorem 1 with a randomized quadratic time decoding. Further, one can use the efficient list-recoverable in the recent work of Guruswami and Xing [11] to obtain same result as in Theorem 1 except with polylogarithmic list sizes.

4 Upper Bounds on the Rate of List-Decodable Synchronization Codes

4.1 Deletion Codes (Theorem 3)

Proof of Theorem 3. To prove this claim, we propose a strategy for the adversary which can reduce the number of strings that may possibly arrive at the receiving side to a number small enough that implies the claimed upper bound for the rate.

We start by proving the theorem for the case where δq is integer. For an arbitrary code $\mathcal{C}$, upon transmission of any codeword, the adversary can remove all occurrences of δq least frequent symbols as the total number of appearances of such symbols does not exceed δn. In case there are more deletions left, adversary may choose to remove arbitrary symbols among the remaining ones. This way, the received string would be a string of $n(1-\delta)$ symbols consisted of only $q-q\delta$ many distinct symbols. Therefore, one can bound above the size of the ensemble of strings that can possibly be received by the $|\mathcal{E}| \leq \binom{q}{q(1-\delta)} [q(1-\delta)]^{n(1-\delta)}$. As the best rate that any $L = \text{poly}(n)$-list decodable code can get is at most $\frac{\log(|\mathcal{E}|\cdot L)}{n\log q} = \frac{\log|\mathcal{E}|}{n\log q} + o(1)$, the following would be an upper bound for the best rate one might hope for.

$$\frac{\log|\mathcal{E}|}{n\log q} + o(1) = \frac{\log\binom{q}{q(1-\delta)} + n(1-\delta)(\log(q(1-\delta)))}{n\log q} + o(1) = (1-\delta)\left(1-\log_q\frac{1}{1-\delta}\right) + o(1)$$

This shows that for the case where $q\delta$ is an integer number, there are no family of codes that achieve a rate that is strictly larger than $(1-\delta)\left(1-\log_q\frac{1}{1-\delta}\right)$.

We now proceed to the general case where $\delta = d/q + \delta'$ for some integer d and $0 \leq \delta' < \frac{1}{q}$. We closely follow the idea that we utilized for the former case. The adversary can partition n sent symbols into two parts of size $nq\delta'$ and $n(1-q\delta')$, and then, similar to the former case, removes the $d+1$ least frequent symbols from the first part by performing $\frac{d+1}{q}\cdot nq\delta'$ deletions and d least frequent symbols from the second one by performing $\frac{d}{q}\cdot n(1-q\delta')$ ones. This is possible because $\frac{d+1}{q}\cdot nq\delta' + \frac{d}{q}\cdot n(1-q\delta') = n\delta$. Doing so, the string received after deletions would contain up to $q-d-1$ distinct symbols in its first $nq\delta'(1-(d+1)/q)$ positions and up to $q-d$ distinct symbols in the other $n(1-q\delta')(1-d/q)$ positions. Therefore, the size of the ensemble of strings that can be received is bounded above as follows.

$$|\mathcal{E}| \leq \binom{q}{q-d-1}[q-d-1]^{nq\delta'\left(1-\frac{d+1}{q}\right)} \cdot \binom{q}{q-d}[q-d]^{n(1-q\delta')\left(1-\frac{d}{q}\right)}$$

This bounds above the rate of any family of list-decodable insdel codes by the following.

$$\begin{aligned}
&\frac{\log|\mathcal{E}|}{n\log q}\\
&= \frac{\log\binom{q}{q-d-1} + nq\delta'\left(1-\frac{d+1}{q}\right)\log(q-d-1) + \log\binom{q}{q-d} + n(1-q\delta')\left(1-\frac{d}{q}\right)\log(q-d)}{n\log q}\\
&= q\delta'\left[\left(1-\frac{d+1}{q}\right)\left(1-\log_q\frac{1}{1-(d+1)/q}\right)\right] + (1-q\delta')\left[\left(1-\frac{d}{q}\right)\left(1-\log_q\frac{1}{1-d/q}\right)\right] \blacktriangleleft
\end{aligned}$$

4.2 Insertion Codes (Theorem 2)

Before providing the proof of Theorem 2, we first point out that any $q-1$ insertions can be essentially used as a single erasure. As a matter of fact, by inserting $q-1$ symbols around the first symbol adversary can make a $1, 2, \cdots, q$ substring around first symbol and therefore, essentially, make the receiver unable to gain any information about it. In fact, with γn insertions, the adversary can repeat this procedure around any $\left\lfloor\frac{\gamma n}{q-1}\right\rfloor$ symbols he wishes. This basically gives that, with γn insertions, adversary can *erase* $\left\lfloor\frac{\gamma n}{q-1}\right\rfloor$ many symbols. Thus, one cannot hope for finding list-decodable codes with rate $1-\frac{\gamma}{q-1}$ or more protecting against γn insertions.

Proof of Theorem 2. To prove this, consider a code $\mathcal{C}$ with rate $R \geq 1-\log_q(\gamma+1) - \gamma\left(\log_q\frac{\gamma+1}{\gamma} - \log_q\frac{q}{q-1}\right)+\varepsilon$ for some $\varepsilon > 0$. We will show that there exist c_0^n many codewords in C that can be turned into one specific string $z \in [1..q]^{n(\gamma+1)}$ with γn insertions for some constant $c_0 > 1$ that merely depends on q and ε.

First, the lower bound assumed for the rate implies that

$$|C| = q^{nR} \geq q^{n\left(1-\log_q(\gamma+1)-\gamma\left(\log_q\frac{\gamma+1}{\gamma}-\log_q\frac{q}{q-1}\right)+\varepsilon\right)}. \tag{1}$$

Let Z be a random string of length $(\gamma+1)n$ over the alphabet $[1..q]$. We compute the expected number of codewords of $\mathcal{C}$ that are subsequences of Z denoted by X.

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{y\in\mathcal{C}}\Pr\{y \text{ is a subsequence of } Z\}\\
&= \sum_{y\in\mathcal{C}}\sum_{1\leq a_1<a_2<\cdots<a_n\leq n(\gamma+1)}\frac{1}{q^n}\left(1-\frac{1}{q}\right)^{a_n-n} &(2)\\
&= |\mathcal{C}|\,(q-1)^{-n}\sum_{l=n}^{n(1+\gamma)}\binom{l}{n}\left(\frac{q-1}{q}\right)^l\\
&\leq |\mathcal{C}|\,(q-1)^{-n}\,n\gamma\binom{n(1+\gamma)}{n}\left(\frac{q-1}{q}\right)^{n(1+\gamma)} &(3)\\
&= n\gamma|\mathcal{C}|(q-1)^{n\gamma}q^{-n(1+\gamma)}2^{n(1+\gamma)H\left(\frac{1}{1+\gamma}\right)+o(n)}\\
&= n\gamma|\mathcal{C}|q^{n\left(\gamma\log_q(q-1)-1-\gamma+\log_q(1+\gamma)+\gamma\log_q\frac{1+\gamma}{\gamma}\right)+o(1)}\\
&= q^{n\varepsilon+o(n)} &(4)
\end{aligned}$$

Step (2) is obtained by conditioning the probability of y being a subsequence of Z over the leftmost occurrence of y in Z indicated by $a_1, a_2, \cdots, a_n$ as indices of Z where the leftmost occurrence of y is located. In that event, Z_{a_i} has to be similar to y_i and y_i cannot appear in $Z[y_{i-1}+1, y_i-1]$. Therefore, the probability of this event is $\left(\frac{1}{q}\right)^n\left(1-\frac{1}{q}\right)^{a_n-n}$. To

verify Step (3), we show that the summation in previous step takes its largest value when $l = n(1+\gamma)$ and bound the summation above by $n\gamma$ times that term. To see that $\binom{l}{n}\left(\frac{q-1}{q}\right)^l$ is maximized for $l = n(1+\gamma)$ in $n \le l \le n(1+\gamma)$ it suffices to show that the ratio of consecutive terms is larger than one for $l \le n(1+\gamma)$:

$$\frac{\binom{l}{n}\left(\frac{q-1}{q}\right)^l}{\binom{l-1}{n}\left(\frac{q-1}{q}\right)^{l-1}} = \frac{l}{l-n}\cdot\frac{q-1}{q} = \frac{1-\frac{1}{q}}{1-\frac{n}{l}} \ge 1$$

The last inequality follows from the fact that $l \le n(\gamma+1) \le nq \Rightarrow \frac{1}{q} < \frac{n}{l}$.

Finally, by (4), there exists some $z \in [1..q]^{(a+1)n}$ to which at least $q^{\varepsilon n + o(n)}$, i.e., exponentially many codewords of $\mathcal{C}$ are subsequences. Therefore, polynomial-sized list decoding for received message z is impossible and proof is complete. ◀

5 Analysis of Random Codes

5.1 Random Insertion Codes (Theorem 7)

Proof of Theorem 7. We prove the claim by considering a random insertion code $\mathcal{C}$ that maps any $x \in [1..q]^{Rn}$ to some uniformly at random chosen member of $[1..q]^n$ denoted by $E_{\mathcal{C}}(x)$ and showing that it is possible to list-decode $\mathcal{C}$ with high probability.

Note that in an insertion channel, the original message sent by Alice is a substring of the message received on Bob's side. Therefore, a random insertion code $\mathcal{C}$ is l-list decodable if for any $z \in [1..q]^{(\gamma+1)n}$, there are at most l codewords of $\mathcal{C}$ that are subsequences of z. For some fixed $z \in [1..q]^{(\gamma+1)n}$, the probability of some uniformly at random chosen $y \in [1..q]^n$ being a substring of z can be bounded above as follows.

$$\begin{aligned}\Pr_y\{y \text{ is a subsequence of } z\} &\le \binom{(\gamma+1)n}{n} q^{-n} \\ &= 2^{n(\gamma+1)H(\frac{1}{\gamma+1})+o(n)} q^{-n} \\ &= q^{n\left(\log_q(\gamma+1)+\gamma\log_q\frac{\gamma+1}{\gamma}-1+o(1)\right)}\end{aligned}$$

Therefore, for a random code $\mathcal{C}$ of rate R and any $m_1, \cdots, m_{l+1} \in [1..q]^{nR}$ and some fixed $z \in [1..q]^{n(\gamma+1)}$:

$$\Pr\{E_{\mathcal{C}}(m_1), \cdots, E_{\mathcal{C}}(m_{l+1}) \text{ are subsequences of } z\} \le q^{n(l+1)\left(\log_q(\gamma+1)+\gamma\log_q\frac{\gamma+1}{\gamma}-1+o(1)\right)}$$

Hence, using the union bound over $z \in [1..q]^{n(\gamma+1)}$, for the random code $\mathcal{C}$:

$$\begin{aligned}&\Pr_{\mathcal{C}}\left\{\exists z \in [1..q]^{n(\gamma+1)}, m_1, \cdots, m_{l+1} \in q^{nR} \text{ s.t. } E_{\mathcal{C}}(m_1), \cdots \text{ are subsequences of } z\right\} \\ \le\ & q^{n(\gamma+1)}\left(q^{Rn}\right)^{l+1} q^{n(l+1)\left(\log_q(\gamma+1)+\gamma\log_q\frac{\gamma+1}{\gamma}-1+o(1)\right)} \\ =\ & q^{n(\gamma+1)+Rn(l+1)+n(l+1)\left(\log_q(\gamma+1)+\gamma\log_q\frac{\gamma+1}{\gamma}-1+o(1)\right)} \qquad (5)\end{aligned}$$

As long as q's exponent in (5) is negative, this probability is less than one and drops exponentially to zero as n grows.

$$\begin{aligned}& n(\gamma+1) + Rn(l+1) + n(l+1)\left(\log_q(\gamma+1) + \gamma\log_q\frac{\gamma+1}{\gamma} - 1 + o(1)\right) < 0 \\ \Leftrightarrow\quad & R < 1 - \log_q(\gamma+1) - \gamma\log_q\frac{\gamma+1}{\gamma} - \frac{\gamma+1}{l+1} + o(1) \qquad (6)\end{aligned}$$

Therefore, the family of random codes with any rate R that satisfies (6) is list-decodable with a list of size l with high probability. ◀

The analysis for random deletion codes (Theorem 6) can be found in the extended version of this article.

References

1. Venkatesan Guruswami and Piotr Indyk. Expander-based constructions of efficiently decodable codes. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 658–667. IEEE, 2001.
2. Venkatesan Guruswami and Piotr Indyk. Near-optimal linear-time codes for unique decoding and new list-decodable codes over smaller alphabets. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 812–821. ACM, 2002.
3. Venkatesan Guruswami and Piotr Indyk. Linear time encodable and list decodable codes. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 126–135. ACM, 2003.
4. Venkatesan Guruswami and Piotr Indyk. Linear-time list decoding in error-free settings. In *International Colloquium on Automata, Languages, and Programming*, pages 695–707. Springer, 2004.
5. Venkatesan Guruswami and Ray Li. Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes. In *Information Theory (ISIT), 2016 IEEE International Symposium on*, pages 620–624. IEEE, 2016.
6. Venkatesan Guruswami and Atri Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions on Information Theory*, 54(1):135–150, 2008.
7. Venkatesan Guruswami and Madhu Sudan. Improved decoding of reed-solomon and algebraic-geometric codes. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 28–37. IEEE, 1998.
8. Venkatesan Guruswami and Carol Wang. Optimal rate list decoding via derivative codes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 593–604. Springer, 2011.
9. Venkatesan Guruswami and Carol Wang. Deletion codes in the high-noise and high-rate regimes. *IEEE Transactions on Information Theory*, 63(4):1961–1970, 2017.
10. Venkatesan Guruswami and Chaoping Xing. List decoding reed-solomon, algebraic-geometric, and gabidulin subcodes up to the singleton bound. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 843–852. ACM, 2013.
11. Venkatesan Guruswami and Chaoping Xing. Optimal rate list decoding over bounded alphabets using algebraic-geometric codes. *arXiv preprint arXiv:1708.01070*, 2017.
12. Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: Codes for insertions and deletions approaching the singleton bound. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, 2017.
13. Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: Explicit constructions, local decoding, and applications. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, 2018.
14. Bernhard Haeupler, Amirbehshad Shahrasbi, and Ellen Vitercik. Synchronization strings: Channel simulations and interactive coding for insertions and deletions. *Proceedings of the International Conference on Automata, Languages, and Programming (ICALP)*, 2017.
15. Brett Hemenway, Noga Ron-Zewi, and Mary Wootters. Local list recovery of high-rate tensor codes and applications. *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2017.

16 Brett Hemenway and Mary Wootters. Linear-time list recovery of high-rate expander codes. In *International Colloquium on Automata, Languages, and Programming*, pages 701–712. Springer, 2015.

17 Swastik Kopparty. List-decoding multiplicity codes. *Theory of Computing*, 11(5):149–182, 2015.

18 Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR 163*, 4:845–848, 1965.

19 Hugues Mercier, Vijay K Bhargava, and Vahid Tarokh. A survey of error-correcting codes for channels with symbol synchronization errors. *IEEE Communications Surveys & Tutorials*, 12(1), 2010.

20 Michael Mitzenmacher. A survey of results for deletion channels and related synchronization channels. *Probability Surveys*, 6:1–33, 2009.

21 Leonard J. Schulman and David Zuckerman. Asymptotically good codes correcting insertions, deletions, and transpositions. *IEEE transactions on information theory*, 45(7):2552–2557, 1999.

22 Neil JA Sloane. On single-deletion-correcting codes. *Codes and designs*, 10:273–291, 2002.

23 Antonia Wachter-Zeh. List decoding of insertions and deletions. *IEEE Transactions on Information Theory*, 2017.

Approximate Sparse Linear Regression

Sariel Har-Peled[1]
Department of Computer Science, University of Illinois, Urbana, IL, USA
sariel@illinois.edu

Piotr Indyk
Department of Computer Science, MIT, Cambridge, MA, USA
indyk@mit.edu

Sepideh Mahabadi[2]
Data Science Institute, Columbia University, New York, NY, USA
mahabadi@mit.edu

Abstract

In the *Sparse Linear Regression* (SLR) problem, given a $d \times n$ matrix M and a d-dimensional query q, the goal is to compute a k-sparse n-dimensional vector τ such that the error $\|M\tau - q\|$ is minimized. This problem is equivalent to the following geometric problem: given a set P of n points and a query point q in d dimensions, find the closest k-dimensional subspace to q, that is spanned by a subset of k points in P. In this paper, we present data-structures/algorithms and conditional lower bounds for several variants of this problem (such as finding the closest induced k dimensional flat/simplex instead of a subspace).

In particular, we present *approximation* algorithms for the online variants of the above problems with query time $\widetilde{O}(n^{k-1})$, which are of interest in the "low sparsity regime" where k is small, e.g., 2 or 3. For $k = d$, this matches, up to polylogarithmic factors, the lower bound that relies on the *affinely degenerate conjecture* (i.e., deciding if n points in $\mathbb{R}^d$ contains $d+1$ points contained in a hyperplane takes $\Omega(n^d)$ time). Moreover, our algorithms involve formulating and solving several geometric subproblems, which we believe to be of independent interest.

2012 ACM Subject Classification Theory of computation → Computational geometry, Theory of computation → Data structures design and analysis

Keywords and phrases Sparse Linear Regression, Approximate Nearest Neighbor, Sparse Recovery, Nearest Induced Flat, Nearest Subspace Search

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.77

Related Version A full version of the paper is available at `https://arxiv.org/abs/1609.08739`.

Funding This research was supported by NSF and Simons Foundation.

1 Introduction

The goal of the *Sparse Linear Regression* (SLR) problem is to find a sparse linear model explaining a given set of observations. Formally, we are given a matrix $M \in \mathbb{R}^{d \times n}$, and a vector $q \in \mathbb{R}^d$, and the goal is to find a vector τ that is k-sparse (has at most k non-zero entries) and that minimizes $\|q - M\tau\|_2$. The problem also has a natural query/online variant

[1] [Work on this paper was partially supported by NSF AF awards CCF-1421231, and CCF-1217462.]
[2] [This work was done while this author was at MIT.]

where the matrix M is given in advance (so that it can be preprocessed) and the goal is to quickly find τ given q.

Various variants of SLR have been extensively studied, in a wide range of fields including
(i) statistics and machine learning [15, 16],
(ii) compressed sensing [6], and
(iii) computer vision [17].
The query/online variant is of particular interest in the application described by Wright *et al.* [17], where the matrix M describes a set of image examples with known labels and q is a new image that the algorithm wants to label.

If the matrix M is generated at random or satisfies certain assumptions, it is known that a natural convex relaxation of the problem finds the optimum solution in polynomial time [3, 4]. However, in general the problem is known to be NP-Hard [13, 5], and even hard to approximate up to a polynomial factor [8] (see below for a more detailed discussion). Thus, it is likely that any algorithm for this problem that guarantees "low" approximation factor must run in exponential time. A simple upper bound for the offline problem is obtained by enumerating $\binom{n}{k}$ possible supports of τ and then solving an instance of the $d \times k$ least squares problem. This results in $n^k(d+k)^{O(1)}$ running time, which (to the best of our knowledge) constitutes the fastest known algorithm for this problem. At the same time, one can test whether a given set of n points in a d-dimensional space is degenerate by reducing it to n instances of SLR with sparsity d. The former problem is conjectured to require $\Omega(n^d)$ time [7] – this is the *affinely degenerate conjecture*. This provides a natural barrier for running time improvements (we elaborate on this below in Section 1.1.1).

In this paper, we study the complexity of the problem in the case where the sparsity parameter k is constant. In addition to the formulation above, we also consider two more constrained variants of the problem. First, we consider the *Affine SLR* where the vector τ is required to satisfy $\|\tau\|_1 = 1$, and second, we consider the *Convex SLR* where additionally τ should be non-negative. We focus on the approximate version of these problems, where the algorithm is allowed to output a k-sparse vector τ' such that $\|M\tau' - q\|_2$ is within a factor of $1+\varepsilon$ of the optimum.

Geometric interpretation. The SLR problem is equivalent to the *Nearest Linear Induced Flat* problem defined as follows. Given a set P of n points in d dimensions and a d-dimensional vector q, the task is to find a k-dimensional flat spanning a subset B of k points in P and the origin, such that the (Euclidean) distance from q to the flat is minimized. The Affine and Convex variants of SLR respectively correspond to finding the *Nearest Induced Flat* and the *Nearest Induced Simplex* problems, where the goal is to find the closest $(k-1)$-dimensional flat/simplex spanned by a subset of k points in P to the query.

Motivation for the problems studied. Given a large[3] set of items (e.g., images), one would like to store them efficiently for various purposes. One option is to pick a relatively smaller subset of representative items (i.e., support vectors), and represent all items as a combination of this *supporting set*. Note, that if our data-set is diverse and is made out of several distinct groups (say, images of the sky, and images of children), then naturally, the data items would use only some of the supporting set for representation (i.e., the representation over the

[3] *Bigger than the biggest thing ever and then some. Much bigger than that in fact, really amazingly immense, a totally stunning size, "wow, that's big", time.* – The Restaurant at the End of the Universe, Douglas Adams.

Table 1 Summary of results. Here, $S(n,d,\varepsilon)$ denotes the preprocessing time and space used by a $(1+\varepsilon)$-ANN (approximate nearest-neighbor) data-structure, and $T_Q(n,d,\varepsilon)$ denotes the query time (we assume all these bounds are at least linear in the dimension d). All the data-structures, except the last one, provide $(1+\varepsilon)$-approximation. In the nearest induced segment case (i.e., this is the offline convex SLR case) the algorithm answers a single query.

	Comment	Space	Query	See
SLR		$n^{k-1}S(n,d,\varepsilon)$	$n^{k-1}T_Q(n,d,\varepsilon)$	Theorem 12
Affine SLR		$n^{k-1}S(n,d,\varepsilon)$	$n^{k-1}T_Q(n,d,\varepsilon)$	Theorem 11
Convex SLR		$n^{k-1}S(n,d,\varepsilon)\log^k n$	$n^{k-1}T_Q(n,d,\varepsilon)\log^k n$	Lemma 25
	$k=2$ & $\varepsilon \le 1$	$nS(n,d,\varepsilon)\log n$	$nT_Q(n,d,\varepsilon)\varepsilon^{-2}\log n$	Full version
Approximate nearest induced segment	$k=2$ $2(1+\varepsilon)$ Approx	$n^{1+O(\frac{1}{(1+\varepsilon)^2})}$		Full version
	$d=O(1)$	$O(n\log n + n/\varepsilon^d)$		Full version

supporting set is naturally sparse). As such, it is natural to ask for a sparse representation of each item over the (sparse but still relatively large) supporting set. (As a side note, surprisingly little is known about how to choose such a supporting set in theory, and the problem seems to be surprisingly hard even for points in the plane.)

Now, when a new item arrives to the system, the task is to compute its best sparse representation using the supporting set, and we would like to do this as fast as possible (which admittedly is not going to be that fast, see below for details).

1.1 Our results

Data-structures. We present data-structures to solve the online variants of the SLR, Affine SLR and Convex SLR problems, for general value of k. Our algorithms use a provided approximate nearest-neighbor (ANN) data-structure as a black box. The new results are summarized in Table 1.

For small values of k, our algorithms offer notable improvements of the query time over the aforementioned naive algorithm, albeit at a cost of preprocessing. Below in Section 1.1.1, we show how our result matches the lower bound that relies on the affinely degenerate conjecture. Moreover, our algorithms involve formulating and solving several interesting geometric subproblems, which we believe to be of independent interest.

Conditional lower bound. We show a conditional lower bound of $\Omega(n^{k/2}/(e^k \log^{\Theta(1)} n))$, for the offline variants of all three problems. Improving this lower bound further, for the case of $k=4$, would imply a nontrivial lower bound for famous Hopcroft's problem. See full version of the paper for the description. Our conditional lower bound result presented in the full version of the paper follows by a reduction from the k-sum problem which is conjectured to require $\Omega(n^{\lceil k/2\rceil}/\log^{\Theta(1)} n)$ time (see e.g., [14], Section 5). This provides further evidence that the off-line variants of the problem require $n^{\Omega(k)}$ time.

1.1.1 Detecting affine degeneracy

Given a point set P in $\mathbb{R}^d$ (here d is conceptually small), it is natural to ask if the points are in general position – that is, all subsets of $d+1$ points are affinely independent. The *affinely degenerate conjecture* states that this problem requires $\Omega(n^d)$ time to solve [7]. This can be achieved by building the arrangement of hyperplanes in the dual, and detecting

any vertex that has $d+1$ hyperplanes passing through it. This problem is also solvable using our data-structure. (We note that since the approximation of our data structure is multiplicative, and in the reduction we only need to detect distance of 0 from larger than 0, we are able to solve the exact degeneracy problem as described next). Indeed, we instantiate Theorem 11, for $k=d$, and using a low-dimensional $(1+\varepsilon)$-ANN data-structure of Arya *et al.* [1]. Such an ANN data-structure uses $S(n,d,\varepsilon)=O(n)$ space, $O(n\log n)$ preprocessing time, and $T_Q(n,d,\varepsilon)=O(\log n+1/\varepsilon^d)=O(\log n)$ query time (for a fixed constant $\varepsilon<1$). Thus, by Theorem 11, our data structure has total space usage and preprocessing time of $\tilde{O}(n^k)$ and a query time of $\tilde{O}(n^{k-1})$. Detecting affine degeneracy then reduces to solving for each point of $q\in P$, the problem of finding the closest $(d-1)$-dimensional induced flat (i.e., passing through d points) of $P\setminus\{q\}$ to q. It is easy to show that this can be solved using our data-structure with an extra log factor[4]. This means that the total runtime (including the preprocessing and the n queries) will be $\tilde{O}(n^k)=\tilde{O}(n^d)$. Thus, up to polylogarithmic factor, the data-structure of Theorem 11 provides an optimal trade-off under the affinely degenerate conjecture. We emphasize that this reduction only rules out the existence of algorithms for online variants of our problems that improve both the preprocessing time from $O(n^k)$, and query time from $O(n^{k-1})$ by much; it does not rule out for example the algorithms with large preprocessing time (in fact much larger than n^k) but small query time.

1.2 Related work

The computational complexity of the approximate sparse linear regression problem has been studied, e.g., in [13, 5, 8]. In particular, the last paper proved a strong hardness result, showing that the problem is hard even if the algorithm is allowed to output a solution with sparsity $k'=k2^{\log^{1-\delta} n}$ whose error is within a factor of $n^c m^{1-\alpha}$ from the optimum, for any constants $\delta,\alpha>0$ and $c>1$.

The query/online version of the Affine SLR problem can be reduced to the *Nearest k-flat Search Problem* studied in [11, 2, 12], where the database consists of a set of k-flats (affine subspaces) of size N and the goal is to find the closest k-flat to a given query point q. Let P be a set of n points in $\mathbb{R}^d$ that correspond to the columns of M. The reduction proceeds by creating a database of all $N=\binom{n}{k}$ possible k-flats that pass through k points of P. However, the result of [2] does not provide multiplicative approximation guarantees, although it does provide some alternative guarantees and has been validated by several experiments. The result of [11], provides provable guarantees and fast query time of $(d+\log N+1/\varepsilon)^{O(1)}$, but the space requirement is quasi-polynomial of the form $2^{(\log N)^{O(1)}}=2^{(k\log n)^{O(1)}}$. Finally the result of [12] only works for the special case of $k=2$, and yields an algorithm with space usage $O\big(n^{14}\varepsilon^{-3}S\big(n^2,d,\varepsilon\big)\big)$ and query time $O\big(T_Q\big(n^2\varepsilon^{-4},d,\varepsilon\big)\log^2 n\big)$[5]. Similar results can be achieved for the other variants.

The SLR problem has a close relationship with the *Approximate Nearest Neighbor (ANN)* problem. In this problem, we are given a collection of N points, and the goal is to build a data structure which, given any query point q, reports the data point whose distance to the query is within a $(1+\varepsilon)$ factor of the distance of the closest point to the query. There are

[4] The details are somewhat tedious – one generates $O(\log n)$ random samples of P where each point is picked with probability half. Now, we build the data-structure for each of the random samples. With high probability, for each of the query point $q\in P$, one of the samples contains, with high probability, the d points defining the closest flat, while not containing q.

[5] The exact exponent is not specified in the main theorem of [12] and it was obtained by an inspection of the proofs in that paper.

many efficient algorithms known for the latter problem. One of the state of the art results for ANN in Euclidean space answers queries in time $(d \log(N)/\varepsilon^2)^{O(1)}$ using $(dN)^{O(1/\varepsilon^2)}$ space [10, 9].

1.3 Our techniques and sketch of the algorithms

Affine SLR (nearest flat). To solve this problem, we first fix a subset $B \subseteq P$ of $k-1$ points, and search for the closest $(k-1)$-flat among those that contain B. Note, that there are at most $n-k+1$ such flats. Each such flat f, as well as the query flat Q_{flat} (containing B and the query q), has only one additional degree of freedom, which is represented by a vector v_H (v_Q, resp.) in a $d-k+1$ space. The vector v_H that is closest to v_Q corresponds to the flat that is closest to q. This can be found approximately using standard ANN data structure, resulting in an algorithm with running time $O(n^{k-1} \cdot T_Q(n,d,\varepsilon))$. Similarly, by adding the origin to the set B, we could solve the SLR problem in a similar way.

Convex SLR (nearest simplex). This case requires an intricate combination of low and high dimensional data structures, and is the most challenging part of this work. To find the closest $(k-1)$-dimensional induced *simplex*, one approach would be to fix B as before, and find the closest corresponding flat. This will work only if the projection of the query onto the closest flat falls inside of its corresponding simplex. Because of that, we need to restrict our search to the flats of *feasible simplices*, i.e., the simplices S such that the projection of the query point onto the corresponding flat falls inside S. If we manage to find this set, we can use the algorithm for affine SLR to find the closest one. Note that finding the distance of the query to the closest non-feasible simplex can easily be computed in time n^{k-1} as the closest point of such a simplex to the query lies on its boundary which is a lower dimensional object.

Let S be the unique simplex obtained from B and an additional point p. Then we can determine whether S is feasible or not only by looking at (i) the relative positioning of p with respect to B, that is, how the simplex looks like in the flat going through S, (ii) the relative positioning of q with respect to B, and (iii) the distance between the query and the flat of the simplex. Thus, if we were given a set of simplices through B such that all their flats were at a distance r from the query, we could build a single data structure for retrieving all the feasible flats. This can be done by mapping all of them in advance onto a unified $(k-1)$ dimensional space (the "parameterized space"), and then using $k-1$ dimensional orthogonal range-searching trees in that space.

However, the minimum distance r is not known in general. Fortunately, as we show the feasibility property is monotone in the distance: the farther the flat of the simplex is from the query point, the weaker constraints it needs to satisfy. Thus, given a threshold value r, our algorithm retrieves the simplices satisfying the restrictions they need to satisfy if they were at a distance r from the query. This allows us to use binary search for finding the right value of r by random sampling. The final challenge is that, since our access is to an approximate NN data structure (and not an exact one), the above procedure yields a superset of feasible simplices. The algorithm then finds the closest flat corresponding to the simplices in this superset. We show that although the reported simplex may not be feasible, its distance to the query is still approximately at most r.

For overview of the offline nearest segment, and conditional lower bound, see the full version.

2 Preliminaries

2.1 Notations

Throughout the paper, we assume $P \subseteq \mathbb{R}^d$ is the set of input points which is of size n. In this paper, for simplicity, we assume that the point-sets are non-degenerate, however this assumption is not necessary for the algorithms. We use the notation $X \subset_i B$ to denote that X is a subset of B of size i, and use $\mathbf{0}$ to denote the origin. For two points $y, u \in \mathbb{R}^d$, the segment the form is denoted by yu, and the line formed by them by $\mathrm{line}(y, u)$.

▶ **Definition 1.** For a set of points S, let $\mathsf{f}_S = \mathrm{aff}(S) = \left\{ \sum_{i=1}^{|S|} \alpha_i p_i \;\middle|\; p_i \in S, \text{ and } \sum_{i=1}^{|S|} \alpha_i = 1 \right\}$ be the $(|S|-1)$-dimensional flat (or $(|S|-1)$-*flat* for short) passing through the points in the set S (aka the *affine hull* of S). The $(|S|-1)$-dimensional simplex ($(|S|-1)$-*simplex* for short) that is formed by the convex-hull of the points of S is denoted by $\triangle_S$. We denote the *interior* of a simplex $\triangle_S$ by $\mathrm{int}(\triangle_S)$.

▶ **Definition 2** (distance and nearest-neighbor)**.** For a point $q \in \mathbb{R}^d$, and a point $p \in \mathbb{R}^d$, we use $\mathsf{d}(q, p) = \|q - p\|_2$ to denote the *distance* between q and p. For a closed set $X \subseteq \mathbb{R}^d$, we denote by $\mathsf{d}(q, X) = \min_{p \in X} \|q - p\|_2$ the *distance* between q and X. The point of X realizing the distance between q and X is the *nearest neighbor* to q in X, denoted by $\mathsf{nn}(q, X)$. We sometimes refer to $\mathsf{nn}(q, X)$ as the *projection* of q onto X.

More generally, given a finite family of such sets $\mathcal{G} = \left\{ X_i \subseteq \mathbb{R}^d \;\middle|\; i = 1, \ldots, m \right\}$, the *distance* of q from $\mathcal{G}$ is $\mathsf{d}(q, \mathcal{G}) = \min_{X \in \mathcal{G}} \mathsf{d}(q, X)$. The *nearest-neighbor* $\mathsf{nn}(q, \mathcal{G})$ is defined analogously to the above.

▶ Assumption 3. Throughout the paper, we assume we have access to a data structure that can answer $(1+\varepsilon)$-ANN queries on a set of n points in $\mathbb{R}^d$. We use $S(n, d, \varepsilon)$ to denote the space requirement of this data structure, and by $T_Q(n, d, \varepsilon)$ to denote the query time.

2.1.1 Induced stars, bouquets, books, simplices and flats

▶ **Definition 4.** Given a point b and a set P of points in $\mathbb{R}^d$, the *star* of P, with the base b, is the set of segments $\mathrm{star}(b, P) = \{ bp \mid p \in P \setminus \{b\} \}$. Similarly, given a set B of points in $\mathbb{R}^d$, with $|B| = k - 1 \leq d$, the *book* of P, with the base B, is the set of simplices $\boldsymbol{\Delta}(B, P) = \left\{ \triangle_{B \cup \{p\}} \;\middle|\; p \in P \setminus B \right\}$. Finally, the set of flats induced by these simplices, is the *bouquet* of P, denoted by $\mathrm{bqt}(B, P) = \left\{ \mathsf{f}_{B \cup \{p\}} \;\middle|\; p \in P \setminus B \right\}$.

If B is a single point, then the corresponding book is a star, and the corresponding bouquet is a set of lines all passing through the single point in B.

▶ **Definition 5.** For a set $P \subseteq \mathbb{R}^d$, let $\mathcal{L}_k(P) = \left\{ \mathsf{f}_{S \cup \{\mathbf{0}\}} \;\middle|\; S \subset_k P \right\}$ be the set of all linear k-dimensional subspaces induced by P, and $\mathcal{F}_k(P) = \{ \mathsf{f}_S \mid S \subset_k P \}$ be the set of all $(k-1)$-flats induced by P. Similarly, let $\boldsymbol{\Delta}_k(P) = \{ \triangle_S \mid S \subset_k P \}$ be the set of all $(k-1)$-simplices induced by P.

2.2 Problems

In the following, we are given a set P of n points in $\mathbb{R}^d$, a query point q and parameters k and $\varepsilon > 0$. We are interested in the following problems:

I. *SLR* (nearest induced linear subspace): Compute $\mathsf{nn}\big(q, \mathcal{L}_k(P)\big)$.
II. *ANLIF* (approximate nearest linear induced flat): Compute a k-flat $\mathsf{f} \in \mathcal{L}_k(P)$, such that $\mathsf{d}(q, \mathsf{f}) \leq (1+\varepsilon)\mathsf{d}(q, \mathcal{L}_k(P))$.

III. *Affine SLR* (nearest induced flat): Compute $\mathsf{nn}\big(q, \mathcal{F}_k(P)\big)$.
IV. *ANIF* (Approximate Nearest Induced Flat): Compute a $(k-1)$-flat $\mathsf{f} \in \mathcal{F}_k(P)$, such that $\mathsf{d}(q, \mathsf{f}) \leq (1+\varepsilon)\mathsf{d}(q, \mathcal{F}_k(P))$.
V. *Convex SLR* (Nearest Induced Simplex): Compute $\mathsf{nn}\big(q, \mathbf{\Delta}_k(P)\big)$.
VI. *ANIS* (Approximate Nearest Induced Simplex): Compute a $(k-1)$-simplex $\triangle \in \mathbf{\Delta}_k(P)$, such that $\mathsf{d}(q, \triangle) \leq (1+\varepsilon)\mathsf{d}(q, \mathbf{\Delta}_k(P))$.

Here, the parameter k corresponds to the *sparsity* of the solution.

3 Approximating the nearest induced flats and subspaces

Here, we show how to solve approximately the online variants of SLR and affine SLR problems. These are later used in Section 4. We start with the simplified case of the uniform star.

3.1 Approximating the nearest neighbor in a uniform star

Input & task. We are given a base point b, a set P of n points in $\mathbb{R}^d$, and a parameter $\varepsilon > 0$. We assume that $\|b - p\| = 1$, for all $p \in P$. The task is to build a data structure that can report quickly, for a query point q that is also at distance one from b, the $(1+\varepsilon)$-ANN segment to q in $\mathrm{star}(b, P)$.

Preprocessing. The algorithm computes the set $V = \{p - b \mid p \in P \setminus \{b\}\}$, which lies on a unit sphere in $\mathbb{R}^d$. Next, the algorithm builds a data structure $\mathcal{D}_V$ for answering $(1+\varepsilon)$-ANN queries on V.

Answering a query. For a query point q, the algorithm does the following:
(A) Compute $\tau = q - b$.
(B) Compute $(1+\varepsilon)$-ANN to τ in V, denoted by u using $\mathcal{D}_V$.
(C) Let y be the point in P corresponding to u.
(D) Return $\min\big(\mathsf{d}(q, by),\, 1\big)$.

▶ **Lemma 6.** *Consider a base point b, and a set P of n points in $\mathbb{R}^d$ all on $\mathbb{S}(b,1)$, where $\mathbb{S} = \mathbb{S}(b,1)$ is the sphere of radius 1 centered at b. Given a query point $q \in \mathbb{S}$, the above algorithm reports correctly a $(1+\varepsilon)$-ANN in $\mathrm{star}(b, P)$. The query time is dominated by the time to perform a single $(1+\varepsilon)$-ANN query. (Proof in the full version)*

3.2 Approximating the nearest flat in a bouquet

▶ **Definition 7.** For a set X and a point p in $\mathbb{R}^d$, let $p' = \mathsf{nn}(p, X)$. We use $\mathsf{dir}(X, p)$ to denote the unit vector $(p - p')/\|p - p'\|$, which is the *direction* of p in relation to X.

Input & task. We are given sets B and P of $k-1$ and n points, respectively, in $\mathbb{R}^d$, and a parameter $\varepsilon > 0$. The task is to build a data structure that can report quickly, for a query point q, a $(1+\varepsilon)$-ANN flat to q in $\mathrm{bqt}(B, P)$, see Definition 4.

Preprocessing. Let $F = \mathsf{f}_B$. The algorithm computes the set

$$V = \{\mathsf{dir}(F, p), -\mathsf{dir}(F, p) \mid p \in P \setminus B\},$$

which lies on a $d-k+2$ dimensional unit sphere in $\mathbb{R}^{d-k+1}$, and then builds a data structure $\mathcal{D}_V$ for answering (standard) ANN queries on V.

Answering a query. For a query point q, the algorithm does the following:
(A) Compute $\tau = \mathsf{dir}(F, q)$.
(B) Compute ANN to τ in V, denoted by u using the data structure $\mathcal{D}_V$.
(C) Let p be the point in P corresponding to u.
(D) Return the distance $\mathsf{d}(q, \mathsf{f}_{B\cup\{p\}})$.

▶ **Definition 8.** For sets $X, Y \subseteq \mathbb{R}^d$, let $\mathsf{proj}_X(Y) = \{\mathsf{nn}(q, X) \mid q \in Y\}$ be the *projection* of Y on X.

▶ **Lemma 9.** *Consider two affine subspaces $F \subseteq H$ with a base point $b \in F$, and the orthogonal complement affine subspace $F^\perp = \left\{b + \tau \;\middle|\; \langle \tau, u - v\rangle = 0 \text{ for all } u, v \in F, \tau \in \mathbb{R}^d\right\}$. For an arbitrary point $q \in \mathbb{R}^d$, let $q^\perp = \mathsf{proj}_{F^\perp}(q)$. We have that $\mathsf{d}(q, H) = \mathsf{d}(q^\perp, \mathsf{proj}_{F^\perp}(H))$. (Proof in the full version)*

Using the notation of Assumption 3 and Definition 4, we have the following:

▶ **Lemma 10** (ANN flat in a bouquet)**.** *Given sets B and P of $k-1$ and n points, respectively, in $\mathbb{R}^d$, and a parameter $\varepsilon > 0$, one can preprocess them, using a single ANN data structure, such that given a query point, the algorithm can compute a $(1+\varepsilon)$-ANN to the closest $(k-1)$-flat in $\mathrm{bqt}(B, P)$. The algorithm space and preprocessing time is $O(S(n, d, \varepsilon))$, and the query time is $O(T_Q(n, d, \varepsilon))$. (Proof in the full version)*

3.3 The result

Here, we show simple algorithms for the ANIF and the ANLIF problems by employing Lemma 10. We assume $\varepsilon > 0$ is a prespecified approximation parameter.

Approximating the affine SLR. As discussed earlier, the goal is to find an approximately closest $(k-1)$-dimensional flat that passes through k points of P, to the query. To this end, we enumerate all possible $k-1$ subsets of points of $B \subset_{k-1} P$, and build for each such base set B, the data structure of Lemma 10. Given a query, we compute the ANN flat in each one of these data structures, and return the closest one found.

▶ **Theorem 11.** *The aforementioned algorithm computes a $(1+\varepsilon)$-ANN to the closest $(k-1)$-flat in $\mathcal{F}_k(P)$, see Definition 5. The space and preprocessing time is $O(n^{k-1}S(n, d, \varepsilon))$, and the query time is $O(n^{k-1}T_Q(n, d, \varepsilon))$.*

Approximating the SLR. The goal here is to find an approximately closest k-dimensional flat that passes through k points of P and the origin $\mathbf{0}$, to the query. We enumerate all possible $k-1$ subsets of points of $B' \subset_{k-1} P$, and build for each base set $B = B' \cup \{\mathbf{0}\}$, the data structure of Lemma 10. Given a query, we compute the ANN flat in each one of these data structures, and return the closest one found.

▶ **Theorem 12.** *The aforementioned algorithm computes a $(1+\varepsilon)$-ANN to the closest k-flat in $\mathcal{L}_k(P)$, see Definition 5, with space and preprocessing time of $O(n^{k-1}S(n, d, \varepsilon))$, and the query time of $O(n^{k-1}T_Q(n, d, \varepsilon))$.*

4 Approximating the nearest induced simplex

In this section we consider the online variant of the ANIS problem. Here, we are given the parameter k, and the goal is to build a data structure, such that given a query point q, it can find a $(1+\varepsilon)$-ANN induced $(k-1)$-simplex.

As before, we would like to fix a set B of $k-1$ points and look for the closest simplex that contains B and an additional point from P. The plan is to filter out the simplices for which the projection of the query on to them falls outside of the interior of the simplex. Then we can use the algorithm of the previous section to find the closest flat corresponding to the feasible simplices (the ones that are not filtered out). First we define a canonical space and map all these simplices and the query point to a unique $(k+1)$-dimensional space. As it will become clear shortly, the goal of this conversion is to have a common lower dimensional space through which we can find all feasible simplices using range searching queries.

4.1 Simplices and distances

4.1.1 Canonical realization

In the following, we fix a sequence $B = (p_1, \ldots, p_{k-1})$ of $k-1$ points in $\mathbb{R}^d$. We are interested in arguing about simplices induced by $k+1$ points, i.e., B, an additional input point p_k, and a query point q. Since the ambient dimension is much higher (i.e., d), it would be useful to have a common canonical space, where we can argue about all entities.

▶ **Definition 13.** For a given set of points B, let $F = \mathsf{f}_B$. Let $p \notin F$ be a given point in $\mathbb{R}^d$, and consider the two connected components of $\mathsf{f}_{B\cup\{p\}} \setminus F$, which are *halfflats*. The halfflat containing p is the *positive halfflat*, and it is denoted by $\mathsf{f}^+(B, p)$.

Fix some arbitrary point $\boldsymbol{s}^* \in \mathbb{R}^d \setminus F$, and let $G = \mathsf{f}^+(B, \boldsymbol{s}^*)$ be a *canonical* such halfflat. Similarly, for a fixed point $\boldsymbol{s}^{**} \in \mathbb{R}^d \setminus \mathsf{f}_{B\cup\{\boldsymbol{s}^*\}}$, let $H = \mathsf{f}^+(B \cup \boldsymbol{s}^*, \boldsymbol{s}^{**})$. Conceptually, it is convenient to consider $H = \mathbb{R}^{k-2} \times \mathbb{R} \times \mathbb{R}^+$, where the first $k-2$ coordinates correspond to F, and the first $k-1$ coordinates correspond to G (this can be done by applying a translation and a rotation that maps H into this desired coordinates system). This is the *canonical parameterization* of H.

The following observation formalizes the following: Given a $(k-1)$ dimensional halfflat G passing through B, a point on G is uniquely identified by its distances from the points in B.

▶ **Observation 14.** *Given a sequence of distances $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_{k-1})$, there might be only one unique point $p = p_G(\boldsymbol{\ell}) \in G$, such that $\|p - p_i\| = \ell_i$, for $i = 1, \ldots, k-1$. Such a point might not exist at all*[6].

Next, given G and H, a point q and a value $\ell < \mathsf{d}(q, F)$, we aim to define the points $q_G(\ell)$ and $q_H(\ell)$. Consider a point $q \in \mathbb{R}^d \setminus F$ (not necessarily the query point), and consider any positive $(k-1)$-halfflat g that contains B, and is in distance ℓ from q. Furthermore assume that $\ell = \mathsf{d}(q, \mathsf{g}) < \mathsf{d}(q, F)$. Let q_{g} be the projection of q to g. Observe that, by the Pythagorean theorem, we have that $d_i = \|q_{\mathsf{g}} - p_i\| = \sqrt{\|q - p_i\|^2 - \ell^2}$, for $i = 1, \ldots, k-1$. Thus, the above observation implies, that the canonical point $q_G(\ell) = p_G(d_1, \ldots, d_{k-1})$ (see Observation 14) is uniquely defined. Note that this is somewhat counterintuitive as the flat g and thus the point q_{g} are not uniquely defined. Similarly, there is a unique point $q_H(\ell) \in H$, such that:

- **(i)** the projection of $q_H(\ell)$ to G is the point $q_G(\ell)$,
- **(ii)** $\|q_H(\ell) - q_G(\ell)\| = \ell$, and these two also imply that
- **(iii)** $\|q_H(\ell) - p_i\| = \|q - p_i\|$, for $i = 1, \ldots, k-1$.

[6] *Trilateration* is the process of determining the location of $p \in G$ given $\boldsymbol{\ell}$. *Triangulation* is the process of determining the location when one knows the angles (not the distances).

Therefore, given G and H, a point q and a value $\ell < \mathsf{d}(q, F)$, the points $q_G(\ell)$ and $q_H(\ell)$ are uniquely defined. Intuitively, for a halfflat that passes through B and is at distance ℓ from the query, $q_G(\ell)$ models the position of the projection of the query onto the halfflat, and $q_H(\ell)$ models the position of the query point itself with respect to this halfflat. Next, we prove certain properties of these points.

4.1.2 Orbits

▶ **Definition 15.** For a set of points B in $\mathbb{R}^d$, define Φ_B to be the open set of all points in $\mathbb{R}^d$, such that their projection into F lies in the interior of the simplex $\triangle_B = \text{ConvexHull}(B)$. The set Φ_B is a *prism*.

Consider a query point $q \in \Phi_B$, and its projection $q_B = \mathsf{nn}(q, F)$. Let $r = \mathrm{r}_B(q) = \|q - q_B\|$ be the *radius* of q in relation to B. Using the above canonical parameterization, we have that $q_G(0) = (q_B, r)$, and $q_H(0) = (q_G(0), 0) = (q_B, r, 0)$. More generally, for $\ell \in [0, r]$, we have

$$q_G(\ell) = \left(q_B, \sqrt{r^2 - \ell^2}\right) \qquad \text{and} \qquad q_H(\ell) = \left(q_B, \sqrt{r^2 - \ell^2}, \ell\right). \tag{1}$$

The curve traced by $q_H(\ell)$, as ℓ varies from 0 to r, is the *orbit* of q – it is a quarter circle with radius r. The following lemma states a monotonicity property that is the basis for the binary search over the value of ℓ.

▶ **Lemma 16.** *(i) Define $\widehat{q}(\ell) = \left(\sqrt{r^2 - \ell^2}, \ell\right)$, and consider any point $p = (x, 0)$, where $x \geq 0$. Then, the function $d(\ell) = \|\widehat{q}(\ell) - p\|$ is monotonically increasing for $\ell \in [0, r]$.*

(ii) For any point p in the halfflat G, the function $\|q_H(\ell) - p\|$ is monotonically increasing. (Proof in the full version).

4.1.3 Distance to a simplex via distance to the flat

▶ **Definition 17.** Given a point q, and a distance ℓ, let $\triangle_G(q, \ell)$ be the unique simplex in G, having the points of B and the point $q_G(\ell)$ as its vertices. Similarly, let $\triangle_G(q) = \triangle_G(q, 0)$.

Next, we provide the necessary and sufficient conditions for a simplex to be feasible. This lemma lies at the heart of our data structure.

▶ **Lemma 18.** *(Proof in the full version) Given a query point $q \in \Phi_B$, and a point $p_k \in P \setminus B$, for a number $0 < x \leq \mathsf{d}(q, F)$ we have*

(A) $q_G(x) \in \triangle_G(p_k)$ *and* $\mathsf{d}(q, \mathsf{f}^+(B, p_k)) \leq x \implies \mathsf{d}(q, \triangle_{B \cup \{p_k\}}) \leq x$.

(B) $\mathsf{d}(q, \triangle_{B \cup \{p_k\}}) \leq x$ *and* $q \in \Phi_{B \cup \{p_k\}} \implies q_G(x) \in \triangle_G(p_k)$ *and* $\mathsf{d}(q, \mathsf{f}^+(B, p_k)) \leq x$.

4.2 Approximating the nearest page in a book

▶ **Definition 19.** Let P be a set of n points in $\mathbb{R}^d$, and let B be a sequence of $k - 1$ points. Consider the set of simplices having B and one additional point from P; that is, $\mathbf{\Delta} = \mathbf{\Delta}(B, P) = \left\{\triangle_{B \cup \{p\}} \;\middle|\; p \in P \setminus B\right\}$. The set $\mathbf{\Delta}$ is the *book* induced by (B, P), and to a single simplex in this book is (naturally) a *page*.

The task at hand, is to preprocess $\mathbf{\Delta}$ for ANN queries, as long as (i) the nearest point lies in the interior of one of these simplices and (ii) $q \in \Phi_B$. To this end, we consider the canonical representation of this set of simplices $\mathbf{\Delta}_G = \{\triangle_G(p) \mid p \in P \setminus B\}$.

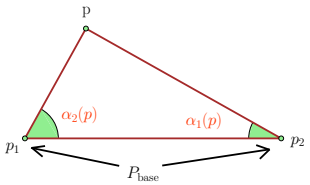


Figure 1 Example base angles when $k = 3$

Idea. The algorithm follows Lemma 18 (A). Given a query point, using standard range-searching techniques, we extract a small number of canonical sets of the points, that in the parametric space, their simplex contains the parameterized query point. This is described in Section 4.2.1. For each of these canonical sets, we use the data structure of Lemma 10 to quickly query each one of these canonical sets for their nearest positive flat (see Remark 4.2.2 below). This would give us the desired ANN.

4.2.1 Reporting all simplices containing a point

▶ **Definition 20.** Let $B = (p_1, \ldots, p_{k-1})$ be a sequence of $k-1$ points in $\mathbb{R}^d$. For a point $p \in \mathbb{R}^d$, consider the $(k-1)$-simplex $\triangle_{B\cup\{p\}}$, which is a full dimensional simplex in the flat $\mathsf{f}_{B\cup\{p\}}$ (see Definition 1). The *base angles* of p (with respect to B), is the $(k-1)$-tuple $\boldsymbol{\alpha}_B(p) = \big(\alpha_1(p), \ldots, \alpha_{k-1}(p)\big)$, where $\alpha_i(p)$ is the dihedral angle between the facet $\triangle_{B\cup\{p\}\setminus\{p_i\}}$ and the base facet $\triangle_B$. See Figure 1, where $k = 3$.

▶ **Observation 21** (Inclusion and base angles). *Let B be a set of $k-1$ points in $\mathbb{R}^{k-1}$ all with their $(k-1)$th coordinate being zero, and let p be an additional point with its $(k-1)$th coordinate being a positive number. Then, for a point $q \in \mathbb{R}^{k-1}$, we have that $q \in \triangle_{B\cup\{p\}} \iff \boldsymbol{\alpha}_B(q) \le \boldsymbol{\alpha}_B(p)$ (i.e., $(\forall i : \alpha_i(q) \le \alpha_i(p))$.*

▶ **Lemma 22.** *Given a set n of $(k-1)$-simplices $\boldsymbol{\Delta}_G$ in $\mathbb{R}^{k-1}$, that all share common $k-1$ vertices, one can build a data structure of size $O(n \log^{k-1} n)$, such that given a query point $q \in \mathbb{R}^{k-1}$, one can compute $O(\log^{k-1} n)$ disjoint canonical sets, such that the union of these sets, is the set of all simplices in $\boldsymbol{\Delta}_G$ that contain q. The query time is $O(\log^{k-1} n)$. (Proof in the full version)*

▶ **Lemma 23.** *The data structure of Lemma 22 can be used to report all simplices that contain a specific point p, and do not contain another point p', which is vertically above p (i.e., the same point with larger $(k-1)$th coordinate). This corresponds to k (possibly unbounded) box queries instead of quadrant query in the orthogonal data structure. The query time and number of canonical sets will be multiplied by at most k. The space bound remains the same. Moreover, we ensure these set of k boxes are disjoint. (Proof in the full version)*

4.2.2 Data structure and correctness

▶ Remark. For a set of points P and a base set B, consider the set of positive halfspaces (the *positive bouquet*) $\mathrm{bqt}^+(B, P) = \{\mathsf{f}^+(B, p) \mid p \in P \setminus B\}$. We can preprocess such a set for ANN queries readily, by using the data structure of Lemma 10. The only modification is that for every positive flat we assign one vector (in the positive direction), instead of two vectors in both directions which we put in the data structure of Section 3.2.

Preprocessing. The algorithm computes the set of canonical simplices $\mathbf{\Delta}_G$, see Eq. (4.2). Next, the algorithm builds the data structure of Lemma 22 for this set of simplices. For each canonical set V in this data structure, for the corresponding set of original points, we build the data structure of Remark 4.2.2 to answer ANN queries on the positive bouquet $\mathrm{bqt}^+(B, V)$. (Observe that the total size of these canonical sets is $O(n \log^{k-1} n)$.)

Answering a query. Given a query point $q \in \Phi_B$, the algorithm computes its projection $q_B = \mathsf{nn}(q, F)$, where $F = \mathsf{f}_B$. Let $r = \|q - q_B\|$ be the radius of q. The desired ANN distance is somewhere in the interval $[0, r]$, and the algorithm maintains an interval $[\alpha, \beta]$ where this distance lies, and uses binary search to keep pruning away on this interval, till reaching the desired approximation.

Observe that for every point $p \in P$, there is a critical value $\gamma(p)$, such that for $x \geq \gamma(p)$, the parameterized point $q_G(x)$ is inside the simplex $\triangle_G(p)$, and is outside if $x < \gamma(p)$. Note that this statement only holds for queries in Φ_B (otherwise it could have been false on simplices $\triangle_{B \cup \{p\}}$ with obtuse angles, see Remark 4.3 for handling the case of $q \notin \Phi_B$).

Now, by Lemma 23, we can compute a polylogarithmic number of canonical sets, such that the union of these sets, are (exactly) all the points with critical values in the range $[\alpha, \beta)$. As long as the number of critical values is at least one, we randomly pick one of these values (by sampling from the canonical sets – one can assume each canonical set is stored in an array), and let γ be this value. We have to decide if the desired ANN is smaller or larger than γ. To this end, we compute a representation, by polylogarithmic number of canonical sets, of all the points of P such that their simplex contains the parameterized point $q_G(\gamma)$, using Lemma 22. For each such canonical set, the algorithm computes the approximate closest positive halfflat, see Remark 4.2.2. Let τ be the minimum distance of such a halfflat computed. If this distance is smaller than γ, then the desired ANN is smaller than γ, and the algorithm continues the search in the interval $[\alpha, \gamma)$, otherwise, the algorithm continues the search in the interval $[\gamma, \beta)$.

After logarithmic number of steps, in expectation, we have an interval $[\alpha', \beta')$, that contains no critical value in it, and the desired ANN distance lies in this interval. We compute the ANN positive flats for all the points that their parameterized simplex contains $q_G(\beta')$, and we return this as the desired ANN distance.

For proof of correctness and query time analysis see the full version.

▶ **Lemma 24** (Approximate nearest induced page)**.** *Given a set P of n points in $\mathbb{R}^d$, a set B of $k-1$ points, and a parameter $\varepsilon > 0$, one can preprocess them, such that given a query point, the algorithm computes an $(1+\varepsilon)$-ANN to the closest page in $\mathbf{\Delta}(B, P)$, see Definition 19. This assumes that (i) the nearest point to the query lies in the interior of the nearest page, and (ii) $q \in \Phi_B$. The algorithm space and preprocessing time is $O(S(n, d, \varepsilon) \log^k n)$, and the query time is $O(T_Q(n, d, \varepsilon) \log^k n)$.*

4.3 Result: nearest induced simplex

The idea is to use brute-force to handle the distance of the query to the $\leq (k-2)$-simplices induced by the given point set which takes $O(n^{k-1})$ time. As such, the remaining task is to handle the $(k-1)$-simplices, and thus we can assume that the nearest point to the query lies in the interior of the nearest simplex, as desired by Lemma 24. To this end, we generate the $\binom{n}{k-1} = O(n^{k-1})$ choices for $B \subseteq P$, and for each one of them we build the data structure of Lemma 24, and query each one of them, returning the closet one found.

▶ Remark. Note that for a set of k points $X \subset_k P$, if the projection of the query onto the simplex $\triangle_A$ falls inside the simplex, i.e. $q \in \Phi_A$, then there exists a subset of $k-1$ points $B \subset_{k-1} X$ such that the projection of the query onto the simplex $\triangle_B$ falls inside the simplex, i.e., $q \in \Phi_B$. Therefore, either the brute-force component of the algorithm finds an ANN, or there exists a set B for which the corresponding data structure reports the correct ANN.

We thus get the following result.

▶ **Theorem 25** (Convex SLR). *Given a set P of n points in $\mathbb{R}^d$, and parameters k and $\varepsilon > 0$, one can preprocess them, such that given a query point, the algorithm can compute a $(1+\varepsilon)$-ANN to the closest $(k-1)$-simplex in $\mathbf{\Delta}_k(P)$, see Definition 5. The algorithm space and preprocessing time is $O(n^{k-1} S(n,d,\varepsilon) \log^k n)$, and the query time is $O(n^{k-1} T_Q(n,d,\varepsilon) \log^k n)$.*

References

1. S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. Assoc. Comput. Mach.*, 45(6):891–923, 1998. doi:10.1145/293347.293348.
2. Ronen Basri, Tal Hassner, and Lihi Zelnik-Manor. Approximate nearest subspace search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(2):266–278, 2011.
3. E. J. Candes, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Inf. Theor.*, 52(2):489–509, February 2006. doi:10.1109/TIT.2005.862083.
4. Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. Atomic decomposition by basis pursuit. *SIAM J. Sci. Comput.*, 20(1):33–61, 1998. doi:10.1137/S1064827596304010.
5. G. Davis, S. Mallat, and M. Avellaneda. Adaptive greedy approximations. *Constructive Approx.*, 13(1):57–98, 1997. doi:10.1007/BF02678430.
6. David L. Donoho. Compressed sensing. *IEEE Trans. Inf. Theor.*, 52(4):1289–1306, 2006. doi:10.1109/TIT.2006.871582.
7. J. Erickson and R. Seidel. Better lower bounds on detecting affine and spherical degeneracies. *Discrete Comput. Geom.*, 13:41–57, 1995. doi:10.1007/BF02574027.
8. Dean P. Foster, Howard J. Karloff, and Justin Thaler. Variable selection is hard. In Peter Grünwald, Elad Hazan, and Satyen Kale, editors, *Proc. 28th Annu. Conf. Comp. Learn. Theo.* (COLT), volume 40 of *JMLR Proceedings*, pages 696–709. JMLR.org, 2015. URL: http://jmlr.org/proceedings/papers/v40/Foster15.html.
9. P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annu. ACM Sympos. Theory Comput.* (STOC), pages 604–613, 1998. doi:10.1145/276698.276876.
10. E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM J. Comput.*, 2(30):457–474, 2000. doi:10.1137/S0097539798347177.
11. Avner Magen. Dimensionality reductions that preserve volumes and distance to affine spaces, and their algorithmic applications. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 239–253. Springer, 2002.
12. Sepideh Mahabadi. Approximate nearest line search in high dimensions. In *Proc. 26th ACM-SIAM Sympos. Discrete Algs.* (SODA), SODA '15, pages 337–354. SIAM, 2015. URL: http://dl.acm.org/citation.cfm?id=2722129.2722154.
13. Balas Kausik Natarajan. Sparse approximate solutions to linear systems. *SIAM J. Comput.*, 24(2):227–234, 1995. doi:10.1137/S0097539792240406.

14 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In Moses Charikar, editor, *Proc. 21st ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 1065–1075. SIAM, 2010. doi:10.1137/1.9781611973075.86.

15 R. Tibshirani. Regression shrinkage and selection via the lasso. *J. Royal Stat. Soc. Series B*, 58(1):267–288, 1996. URL: http://statweb.stanford.edu/~tibs/lasso/lasso.pdf.

16 Robert Tibshirani. Regression shrinkage and selection via the lasso: a retrospective. *J. Royal Stat. Soc. Series B*, 73(3):273–282, 2011. doi:10.1111/j.1467-9868.2011.00771.x.

17 John Wright, Allen Y Yang, Arvind Ganesh, Shankar S Sastry, and Yi Ma. Robust face recognition via sparse representation. *IEEE Trans. Pattern Anal. Machine Intel.*, 31(2):210–227, 2009. doi:10.1109/TPAMI.2008.79.

A Polynomial Time Algorithm to Compute Geodesics in CAT(0) Cubical Complexes

Koyo Hayashi
Department of Mathematical Informatics, University of Tokyo, Tokyo 113-8656, Japan
koyo_hayashi@mist.i.u-tokyo.ac.jp

Abstract

This paper presents the first polynomial time algorithm to compute geodesics in a CAT(0) cubical complex in general dimension. The algorithm is a simple iterative method to update breakpoints of a path joining two points using Miller, Owen and Provan's algorithm (Adv. in Appl. Math, 2015) as a subroutine. Our algorithm is applicable to any CAT(0) space in which geodesics between two close points can be computed, not limited to CAT(0) cubical complexes.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Geodesic, CAT(0) Space, Cubical Complex

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.78

Related Version A full version of the paper is available at `https://arxiv.org/abs/1710.09932`.

Funding The work was supported by JSPS KAKENHI Grant Number 17K00029, and by JST ERATO Grant Number JPMJER1201, Japan.

Acknowledgements I thank Hiroshi Hirai for introducing me to this problem and for helpful comments and careful reading.

1 Introduction

Computing a shortest path in a polyhedral domain in Euclidean space is a fundamental and important algorithmic problem, which is intensively studied in computational geometry [16]. This problem is relatively easy to solve in the two-dimensional case; it can generally be reduced to a discrete graph searching problem where some combinatorial approaches can be applied. In three or more dimensions, however, the problem becomes much harder; it is not even discrete. In fact, it was proved by Canny and Reif [8] that the shortest path problem in a polyhedral domain is NP-hard. Mitchell and Sharir [17] have shown that the problem of finding a shortest obstacle-avoiding path is NP-hard even for the case of a region with obstacles that are disjoint axis-aligned boxes. On the other hand, there are some cases where one can obtain polynomial time complexity. For instance, it was shown by Sharir [24] that a shortest obstacle-avoiding path among k disjoint convex polyhedra having altogether n vertices, can be found in $n^{O(k)}$ time, which implies that this problem is polynomially solvable if k is a small constant.

What determines the tractability of the shortest path problem in geometric domains? One of promising answers to this challenging question is *global non-positive curvature*, or *CAT(0) property* [14]. CAT(0) spaces are metric spaces in which geodesic triangles are "not thicker" than those in the Euclidean plane, and enjoy various fascinating properties generalizing those in Euclidean and hyperbolic spaces. As Ghrist and LaValle [13] observed, no NP-hard example in [17] is a CAT(0) space. One of the significant properties of CAT(0)

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 78; pp. 78:1–78:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

spaces is the uniqueness of geodesics: Every pair of points can be joined by a unique geodesic. Computational and algorithmic theory on CAT(0) spaces is itself a challenging research field [5].

One of fundamental and familiar CAT(0) spaces is a *CAT(0) cubical complex*. A cubical complex is a polyhedral complex where each cell is isometric to a unit cube of some dimension and the intersection of any two cells is empty or a single face. Gromov [14] gave a purely combinatorial characterization of cubical complexes of non-positive curvature as cubical complexes in which the link of each vertex is a flag simplicial complex. Chepoi [9] and Roller [22] established that the 1-skeletons of CAT(0) cubical complexes are exactly *median graphs*, i.e., graphs in which any three vertices admit a unique median vertex. It is also shown by Barthélemy and Constantin [4] that median graphs are exactly the domains of event structures [18]. These nice combinatorial characterizations are one of the main reasons why CAT(0) cubical complexes frequently appear in mathematics, for instance, in geometric group theory [22, 23], metric graph theory [3], concurrency theory in computer science [18], theory of reconfigurable systems [1, 12], and phylogenetics [6].

There has been several polynomial time algorithms to find shortest paths in some CAT(0) cubical complexes. A noteworthy example is for a *tree space*, introduced by Billera, Holmes and Vogtmann [6] as a continuous space of phylogenetic trees. This space is shown to be CAT(0), and consequently provides a powerful tool for comparing two phylogenetic trees through the unique geodesic. Owen and Provan [19, 20] gave a polynomial time algorithm for finding geodesics in tree spaces, which was generalized by Miller et al. [15] to *CAT(0) orthant spaces*, i.e., complexes of Euclidean orthants that are CAT(0). Chepoi and Maftuleac [10] gave an efficient polynomial time algorithm to compute geodesics in a two dimensional CAT(0) cubical complex. These meaningful polynomiality results naturally lead to a question: What about arbitrary CAT(0) cubical complexes?

Ardila, Owen and Sullivant [2] gave a combinatorial description of CAT(0) cubical complexes, employing a poset endowed with an additional relation, called a *poset with inconsistent pairs (PIP)*. This can be viewed as a generalization of Birkhoff's theorem that gives a compact representation of distributive lattices by posets. In fact, they showed that there is a bijection between CAT(0) cubical complexes and PIPs. (Through the above-mentioned equivalence, this can be viewed as a rediscovery of the result of Barthélemy and Constantin [4], who found a bijection between PIPs and pointed median graphs.) This relationship enables us to express an input CAT(0) cubical complex as a PIP: For a poset with inconsistent pairs P, the corresponding CAT(0) cubical complex $\mathcal{K}_P$ is realized as a subcomplex of the $|P|$-dimensional cube $[0,1]^P$ in which the cells of $\mathcal{K}_P$ are specified by structures of P. Adopting this embedding as an input, they gave the first algorithm to compute geodesics in an arbitrary CAT(0) cubical complex. Their algorithm is based on an iterative method to update a sequence of cubes that may contain the geodesic, where at each iteration it solves a touring problem using second order cone programming [21]. They also showed that the touring problem for general CAT(0) cubical complexes has intrinsic algebraic complexity, and geodesics can have breakpoints whose coordinates have nonsolvable Galois group. This implies that there is no exact simple formula for the geodesic and therefore in general, one can only obtain an approximate one. Unfortunately, even if the touring problem could be solved exactly, it is not known whether or not their algorithm is a polynomial one; that is, no polynomial time algorithm has been known for the shortest path problem in a CAT(0) cubical complex in general dimension.

Main result. In this paper, we present the first polynomial time algorithm to compute geodesics in a CAT(0) cubical complex in general dimension, answering the open question suggested by these previous work; namely we show that:

> Given a CAT(0) cubical complex $\mathcal{K}$ represented by a poset with inconsistent pairs P and two points p, q in $\mathcal{K}$, one can find a path joining p and q of length at most $d(p,q)+\epsilon$ in time polynomial in $|P|$ and $\log(1/\epsilon)$.

The algorithm is quite simple, without depending on any involved techniques such as semidefinite programming. To put it briefly, our algorithm first gives a polygonal path joining p and q with a fixed number (n, say) of breakpoints, and then iteratively updates the breakpoints of the path until it becomes a desired one. To update them, we compute the midpoints of the two close breakpoints by using Miller, Owen and Provan's algorithm. The resulting number of iterations is bounded by a polynomial in n. Key tools that lead to this bound are linear algebraic techniques and the convexity of the metric of CAT(0) spaces, rather than inherent properties of cubical complexes. Due to its simplicity, our algorithm is applicable to any CAT(0) space where geodesics between two close points can be found, not limited to CAT(0) cubical complexes. We believe that our result will be an important step toward developing computational geometry in CAT(0) spaces.

Application. A *reconfigurable system* [1, 12] is a collection of states which change according to local and reversible moves that affect global positions of the system. Examples include robot motion planning, non-collision particles moving around a graph, and protein folding; see [12]. Abrams, Ghrist and Peterson [1, 12] considered a continuous space of all possible positions of a reconfigurable system, called a *state complex*. Any state complex is a cubical complex of non-positively curved [12], and it becomes CAT(0) in many situations. In the robotics literature, geodesics (in the l_2-metric) in the CAT(0) state complex corresponds to the motion planning to get the robot from one position to another one with minimal power consumption. Our algorithm enables us to find such an optimal movement of the robot in polynomial time.

2 Computing geodesics in CAT(0) spaces

In this section we devise an algorithm to compute geodesics in general CAT(0) spaces, not limited to CAT(0) cubical complexes.

2.1 CAT(0) space

Let (X,d) be a metric space. A *geodesic* joining two points $x, y \in X$ is a map $\gamma : [0,1] \to X$ such that $\gamma(0) = x, \gamma(1) = y$ and $d(\gamma(s), \gamma(t)) = d(x,y)|s-t|$ for all $s, t \in [0,1]$. The image of γ is called a *geodesic segment* joining x and y. A metric space X is called *(uniquely) geodesic* if every pair of points $x, y \in X$ is joined by a (unique) geodesic.

For any triple of points x_1, x_2, x_3 in a metric space (X,d), there exists a triple of points $\bar{x}_1, \bar{x}_2, \bar{x}_3$ in the Euclidean plane $\mathbb{E}^2$ such that $d(x_i, x_j) = d_{\mathbb{E}^2}(\bar{x}_i, \bar{x}_j)$ for $i, j \in \{1,2,3\}$. The Euclidean triangle whose vertices are $\bar{x}_1, \bar{x}_2$ and $\bar{x}_3$ is called a *comparison triangle* for x_1, x_2, x_3. (Note that such a triangle is unique up to isometry.) A geodesic metric space (X,d) is called a *CAT(0) space* if for any $x_1, x_2, x_3 \in X$ and any p belonging to a geodesic segment joining x_1 and x_2, the inequality $d(x_3, p) \leq d_{\mathbb{E}^2}(\bar{x}_3, \bar{p})$ holds, where $\bar{p}$ is the unique point in $\mathbb{E}^2$ satisfying $d(\bar{x}_i, \bar{p}) = d_{\mathbb{E}^2}(x_i, p)$ for $i = 1, 2$. See Figure 1.

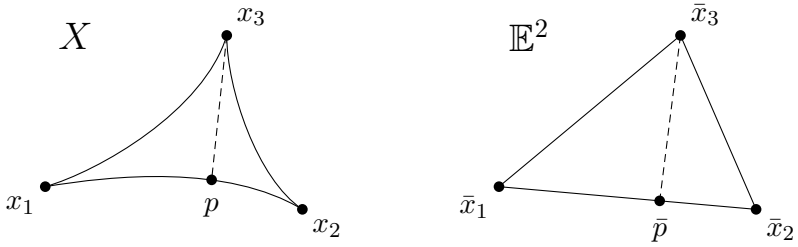


Figure 1 CAT(0) space.

This simple definition yields various significant properties of CAT(0) spaces; see [7] for details. One of the most basic properties of CAT(0) spaces is the convexity of the metric. A geodesic metric space (X, d) is said to be *Busemann convex* if for any two geodesics $\alpha, \beta : [0, 1] \to X$, the function $f : [0, 1] \to \mathbb{R}$ given by $f(t) := d(\alpha(t), \beta(t))$ is convex.

▶ **Lemma 2.1** ([7, Proposition II.2.2]). *Every CAT(0) space is Busemann convex.*

A Busemann convex space X is uniquely geodesic. Indeed, for any two geodesics $\alpha, \beta : [0, 1] \to X$ with $\alpha(0) = \beta(0)$ and $\alpha(1) = \beta(1)$, one can easily see that α and β coincide, since $d(\alpha(t), \beta(t)) \leq (1 - t)d(\alpha(0), \beta(0)) + td(\alpha(1), \beta(1)) = 0$ for all $t \in [0, 1]$. This implies that:

▶ **Theorem 2.2** ([7, Proposition II.1.4]). *Every CAT(0) space is uniquely geodesic.*

2.2 Algorithm

Let X be a CAT(0) space. We shall refer to an element x in the product space X^{n+1} as a *chain*, and write x_{i-1} to denote the i-th component of x, i.e., $x = (x_0, x_1, \ldots, x_n)$. For any chain $x \in X^{n+1}$, we define the *length* of x by $\sum_{i=0}^{n-1} d(x_i, x_{i+1})$ and denote it by $\ell(x)$. We consider the following problem:

> Given two points $p, q \in X$, a chain $x \in X^{n+1}$ with $x_0 = p$ and $x_n = q$, and a positive parameter $\epsilon > 0$, find a chain $y \in X^{n+1}$ such that $y_0 = p$, $y_n = q$ and $\ell(y) \leq d(p, q) + \epsilon$, (1)

under the situation where we are given an oracle to perform the following operation for some $D > 0$:

> Given two points $p, q \in X$ with $d(p, q) \leq D$, compute the geodesic joining p and q in arbitrary precision. (2)

To explain our algorithm to solve this problem, we need some definitions. Since X is uniquely geodesic, every pair of points $p, q \in X$ has a unique midpoint w satisfying $2d(w, p) = 2d(q, w) = d(p, q)$. For a nonnegative real number $\delta \geq 0$, a *δ-midpoint* of p and q is a point $w' \in X$ satisfying $d(w', w) \leq \delta$, where w is the midpoint of p and q.

▶ **Definition 2.3** (δ-halved chain). Let δ be a nonnegative real number. For any chain $x \in X^{n+1}$, a chain $z \in X^{n+1}$ is called a *δ-halved chain* of x if it satisfies the following:

$z_0 = x_n$, $z_n = x_0$ and z_i is a δ-midpoint of z_{i+1} and x_{n-i} for $i = 1, 2, \ldots, n - 1$.

For an integer $k \geq 0$, we say that $x^{(k)}$ is a *k-th δ-halved chain* of x if there exists a sequence $\{x^{(j)}\}_{j=0}^{k}$ of chains in X^{n+1} such that $x^{(0)} = x$ and $x^{(j)}$ is a δ-halved chain of $x^{(j-1)}$ for $j = 1, 2, \ldots, k$.

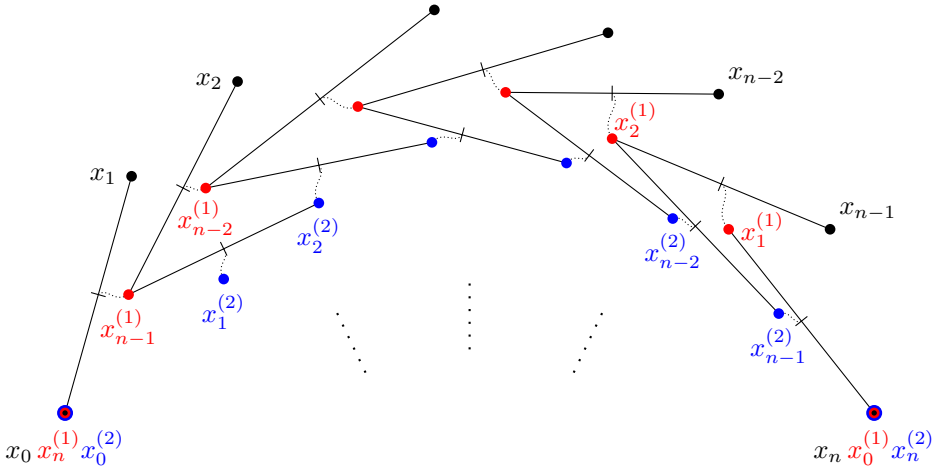


Figure 2 An illustration of Algorithm 1.

Our algorithm can be described as follows. To put it briefly, the algorithm just finds a k-th δ-halved chain of a given chain x for some large k and small δ; see Figure 2 for an illustration. In the algorithm the local optimization is done alternatively "from left to right" and "from right to left" so that the analysis will be easier.

Algorithm 1

Input. Two points $p, q \in X$, a chain $x \in X^{n+1}$ with $x_0 = p$ and $x_n = q$, and parameters $\epsilon > 0, \delta \geq 0$.

⟨**1**⟩ Set $x^{(0)} := x$.

⟨**2**⟩ For $j = 0, 1, 2, \ldots,$ do the following:

⟨**2-1**⟩ Set $z_0 := x_n^{(j)}$ and $z_n := x_0^{(j)}$.

⟨**2-2**⟩ For $i = 1, 2, \ldots, n-1$, do the following:

$$\text{Compute a } \delta\text{-midpoint } w \text{ of } z_{n-i+1} \text{ and } x_i^{(j)}, \text{ and set } z_{n-i} := w. \tag{3}$$

⟨**2-3**⟩ Set $x^{(j+1)} := (z_0, z_1, \ldots, z_n)$.

For any chain $x \in X^{n+1}$, define the *gap* of x by $\max\{d(x_0, x_1), \max_{1 \leq i \leq n-1} 2d(x_i, x_{i+1})\}$ and denote it by $\mathrm{gap}(x)$. The following theorem states that Algorithm 1 solves problem (1).

▶ **Theorem 2.4.** *Let $p, q \in X$ be given two points, $x \in X^{n+1}$ be a given chain with $x_0 = p$ and $x_n = q$, and $\epsilon > 0, 0 \leq \delta \leq \epsilon/(16n^3)$ be parameters.*

(i) *For $j \geq n^2 \log(4n \cdot \ell(x)/\epsilon)$, one has $\ell(x^{(j)}) \leq d(p, q) + \epsilon$.*

(ii) *If $\mathrm{gap}(x) \leq D/2 - \epsilon$ for some $D > 0$, then for all $j \geq 0$ and for $i = 1, 2, \ldots, n-1$, one has $d(z_{n-i+1}, x_i^{(j)}) \leq D$ in (3).*

In particular, for $\mathrm{gap}(x) \leq D/2 - \epsilon$, one can find a chain $y \in X^{n+1}$ such that $y_0 = p$, $y_n = q$ and $\ell(y) \leq d(p, q) + \epsilon$, with $O(n^3 \log(nD/\epsilon))$ calls of an oracle to perform (2).

▶ **Example 2.5.** We give an example of CAT(0) spaces to which our algorithm is applicable. A *B_2-complex* is a two dimensional piecewise Euclidean complex in which each 2-cell is isomorphic to an isosceles right triangle with short side of length one [11]. A CAT(0) B_2-complex is called a *folder complex* [9]; see Figure 3 for an example. One can show that for a

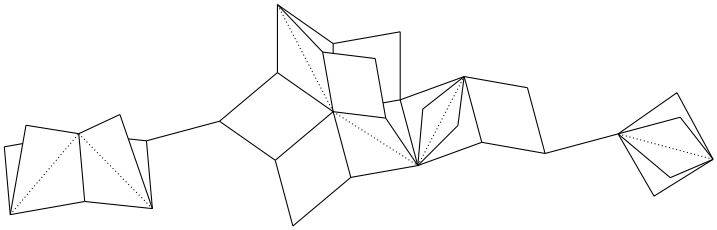


Figure 3 A folder complex.

folder complex $\mathcal{F}$, computing the geodesic between two points $p, q \in \mathcal{F}$ with $d(p,q) \leq 1$ can be reduced to an easy calculation on a subcomplex of $\mathcal{F}$ having a few cells. This implies that our algorithm enables us to find geodesics between two points in a folder complex $\mathcal{F}$ in time bounded by a polynomial in the size of $\mathcal{F}$.

2.3 Analysis

For any chain $x \in X^{n+1}$, we define the *reference chain* $\hat{x} \in X^{n+1}$ of x as follows: $\hat{x}_0 := x_0$ and $\hat{x}_i := \gamma((i+1)/(n+1))$ for $i = 1, 2, \ldots, n$, where $\gamma : [0,1] \to X$ is the geodesic with $\gamma(0) = x_0$ and $\gamma(1) = x_n$. Reference chains are designed not to be equally spaced but to have a double gap in the beginning so that the analysis of the algorithm will be easier. Note that the reference chain $\hat{x}$ of x is determined just by its end components x_0, x_n, and therefore for any chain x and any even δ-halved chain $x^{(2k)}$ of x their reference chains coincide: $\hat{x}^{(2k)} = \hat{x}$. A key observation that leads to Theorem 2.4 is that: For any chain $x \in X^{n+1}$ and any k-th δ-halved chain $x^{(k)}$ of x with k sufficiently large and δ sufficiently small, the distance between $x^{(k)}$ and its reference chain $\hat{x}^{(k)}$ is small enough for its length $\ell(x^{(k)})$ to approximate well $d(x_0, x_n)$; moreover, the value of such a k can be bounded by a polynomial in n. The next lemma states this fact.

▶ **Lemma 2.6.** *Let $x \in X^{n+1}$. Any k-th δ-halved chain $x^{(k)}$ of x satisfies*

$$d(x_i^{(k)}, \hat{x}_i^{(k)}) \leq (5/4)\ell(x)e^{-k/n^2} + 3n^2\delta$$

for $i = 1, 2, \ldots, n-1$, where e is the base of the natural logarithm.

Proof. Let $\{x^{(j)}\}_{j \geq 0}$ be a sequence of chains in X^{n+1} such that $x^{(0)} = x$ and $x^{(j)}$ is a δ-halved chain of $x^{(j-1)}$ for $j \geq 1$. Fix an integer $1 \leq i \leq n-1$ and an integer $k \geq 0$. Note that by definition $x_i^{(k+1)}$ is a δ-midpoint of $x_{i+1}^{(k+1)}$ and $x_{n-i}^{(k)}$ and that $\hat{x}_i^{(k+1)}$ is the midpoint of $\hat{x}_{i+1}^{(k+1)}$ and $\hat{x}_{n-i}^{(k)}$. Hence, by Lemma 2.1 and the triangle inequality, we have

$$2d(x_i^{(k+1)}, \hat{x}_i^{(k+1)}) \leq 2d(w, \hat{x}_i^{(k+1)}) + 2\delta \leq d(x_{i+1}^{(k+1)}, \hat{x}_{i+1}^{(k+1)}) + d(x_{n-i}^{(k)}, \hat{x}_{n-i}^{(k)}) + 2\delta, \tag{4}$$

where w is the midpoint of $x_{i+1}^{(k+1)}$ and $x_{n-i}^{(k)}$.

Let $v^{(k)}$ be a column vector of dimension $n-1$ whose i-th entry equals $d(x_i^{(k)}, \hat{x}_i^{(k)})$ for $i = 1, 2, \ldots, n-1$. Let J be a square matrix of order $n-1$ whose (i,j) entry equals 1 if $i + j = n$ and 0 otherwise. Let K be a square matrix of order $n-1$ whose (i,j) entry equals 1 if $j = i+1$ and 0 otherwise. Then, by (4) we have $2v^{(k+1)} \leq Kv^{(k+1)} + Jv^{(k)} + 2\delta\mathbf{1}$ for each $k \geq 0$, where $\mathbf{1}$ is a column vector with all entries equal to 1. Let A_{n-1} be a square matrix of order $n-1$ whose (i,j) entry equals $(1/2)^{n+1-i-j}$ if $i + j \leq n$ and 0 otherwise. Then one can easily see that $(2I - K)^{-1}J = A_{n-1}$. Hence we have

$$v^{(k+1)} \leq A_{n-1}v^{(k)} + A_{n-1}J^{-1}(2\delta\mathbf{1}) \leq A_{n-1}v^{(k)} + 2\delta\mathbf{1} \tag{5}$$

for each $k \geq 0$. We show that

$$v^{(k)} \leq ((5/4)\ell(x)e^{-k/n^2} + 3n^2\delta)\mathbf{1} \tag{6}$$

for any integer $k \geq 0$. The inequality (5) inductively yields that $v^{(k)} \leq (A_{n-1})^k v^{(0)} + 2\delta(I + A_{n-1} + \cdots + (A_{n-1})^{k-1})\mathbf{1} \leq \ell(x)(A_{n-1})^k\mathbf{1} + 2\delta(I - A_{n-1})^{-1}\mathbf{1}$. Here, the inequality $v^{(0)} \leq \ell(x)\mathbf{1}$ comes from the triangle inequality. Indeed, we have

$$\begin{aligned} d(x_i, \hat{x}_i) &\leq \min\{d(x_0, \hat{x}_i) + \textstyle\sum_{j=0}^{i-1} d(x_j, x_{j+1}),\ d(\hat{x}_i, x_n) + \sum_{j=i}^{n-1} d(x_j, x_{j+1})\} \\ &\leq (d(x_0, x_n) + \ell(x))/2 \leq \ell(x) \end{aligned}$$

for $i = 1, 2, \ldots, n-1$. In Lemma 2.7 below, we prove $(I - A_{n-1})^{-1}\mathbf{1} \leq (5(n-1)^2/4)\mathbf{1}$ (for $n-1 \geq 2$). This yields that $(I - A_{n-1})^{-1}\mathbf{1} \leq (3/2)n^2\mathbf{1}$ for $n \geq 2$. Also, we prove $(A_{n-1})^k\mathbf{1} \leq (5/4)e^{-k/(n-1)^2}\mathbf{1}$ (for $n-1 \geq 2$) in Lemma 2.7. This implies that $(A_{n-1})^k\mathbf{1} \leq (5/4)e^{-k/n^2}\mathbf{1}$ for $n \geq 2$. This proves (6) and therefore completes the proof of the lemma. ◀

Let us now prove Theorem 2.4.

Proof of Theorem 2.4. We may assume that $n \geq 2$. We first show (i). If $\delta \leq \epsilon/(16n^3)$ and $j \geq n^2 \log(4n \cdot \ell(x)/\epsilon)$, then by Lemma 2.6, any j-th δ-halved chain $x^{(j)}$ of x satisfies $d(x_i^{(j)}, \hat{x}_i^{(j)}) \leq 5\epsilon/(16n) + 3\epsilon/(16n) = \epsilon/(2n)$ for $i = 1, 2, \ldots, n-1$. Hence one has

$$\begin{aligned} d(x_i^{(j)}, x_{i+1}^{(j)}) &\leq d(x_i^{(j)}, \hat{x}_i^{(j)}) + d(\hat{x}_i^{(j)}, \hat{x}_{i+1}^{(j)}) + d(\hat{x}_{i+1}^{(j)}, x_{i+1}^{(j)}) \\ &\leq d(\hat{x}_i^{(j)}, \hat{x}_{i+1}^{(j)}) + \epsilon/n \end{aligned} \tag{7}$$

for $i = 0, 1, \ldots, n-1$. This implies that $\ell(x^{(j)}) = \sum_{i=0}^{n-1} d(x_i^{(j)}, x_{i+1}^{(j)}) \leq \sum_{i=0}^{n-1}(d(\hat{x}_i^{(j)}, \hat{x}_{i+1}^{(j)}) + \epsilon/n) = d(x_0, x_n) + \epsilon = d(p, q) + \epsilon$, and therefore completes the proof of (i).

To prove (ii), we first show

$$d(z_{n-i+1}, x_i^{(j)}) \leq \mathrm{gap}(x^{(j)}) + 2\delta \quad (i = 1, 2, \ldots, n; j \geq 0), \tag{8}$$

by induction on i. The case $i = 1$ being trivial, suppose that $i \geq 2$. Since z_{n-i+1} is a δ-midpoint of z_{n-i+2} and $x_{i-1}^{(j)}$, the triangle inequality and the induction yield $d(z_{n-i+1}, x_i^{(j)}) \leq \delta + d(z_{n-i+2}, x_{i-1}^{(j)})/2 + d(x_{i-1}^{(j)}, x_i^{(j)}) \leq \delta + (\mathrm{gap}(x^{(j)})/2 + \delta) + \mathrm{gap}(x^{(j)})/2 = \mathrm{gap}(x^{(j)}) + 2\delta$, which completes the induction.

It follows from (8) that $\mathrm{gap}(x^{(j+1)}) \leq \mathrm{gap}(x^{(j)}) + 4\delta$ for $j \geq 0$. Indeed, the case $i = n$ in (8) implies that $d(z_1, z_0) = d(z_1, x_n^{(j)}) \leq \mathrm{gap}(x^{(j)}) + 2\delta$; on the other hand, by the triangle inequality and (8), one has $d(z_{n-i+1}, z_{n-i}) \leq d(z_{n-i+1}, x_i^{(j)})/2 + \delta \leq \mathrm{gap}(x^{(j)})/2 + 2\delta$ for $i = 1, 2, \ldots, n-1$. Thus, one has $\mathrm{gap}(x^{(j+1)}) \leq \max\{\mathrm{gap}(x^{(j)}) + 2\delta, 2(\mathrm{gap}(x^{(j)})/2 + 2\delta)\} = \mathrm{gap}(x^{(j)}) + 4\delta$.

The inequality (8) implies that in order to prove (ii) it suffices to show that $\mathrm{gap}(x^{(j)}) + 2\delta \leq D$ for all $j \geq 0$. Suppose that $\delta \leq \epsilon/(16n^3)$. We consider two cases.

Case 1: $j \leq n^2 \log(4n \cdot \ell(x)/\epsilon)$. Note that $\ell(x) \leq n \cdot \mathrm{gap}(x)$ and that $\mathrm{gap}(x^{(j)}) \leq \mathrm{gap}(x) + 4j\delta$.

However roughly one estimates an upper bound of $4j\delta$, one can get

$$4j\delta \leq 4 \cdot \frac{\epsilon}{16n^3} \cdot n^2 \log \frac{4n^2 \cdot \mathrm{gap}(x)}{\epsilon} = \frac{\epsilon}{4n}\left(\log \frac{\mathrm{gap}(x)}{\epsilon} + 2\log 2n\right) \leq \frac{\mathrm{gap}(x)}{4ne} + \frac{\epsilon}{e},$$

where the last inequality comes from the fact that $\log t \leq t/e$ for any $t > 0$. It is easy to see that $\mathrm{gap}(x^{(j)}) + 2\delta \leq \mathrm{gap}(x) + \mathrm{gap}(x)/(4ne) + \epsilon/e + \epsilon/(8n^3) \leq D$, provided $\mathrm{gap}(x) \leq D/2 - \epsilon$.

Case 2: $j \geq n^2 \log(4n \cdot \ell(x)/\epsilon)$. Recall (7). Since $d(x_0, x_n)/(n+1) \leq \mathrm{gap}(x)/2$, we have

$$\mathrm{gap}(x^{(j)}) \leq \max\{\mathrm{gap}(x) + \epsilon/n, 2(\mathrm{gap}(x)/2 + \epsilon/n)\} = \mathrm{gap}(x) + 2\epsilon/n.$$

It is easy to see that $\mathrm{gap}(x^{(j)}) + 2\delta \leq \mathrm{gap}(x) + 2\epsilon/n + \epsilon/(8n^3) \leq D$, provided $\mathrm{gap}(x) \leq D/2 - \epsilon$.

From (i) and (ii), we can show the last part of the theorem. Indeed, for $k := \lceil n^2 \log(4n \cdot \ell(x)/\epsilon) \rceil$, one can find a k-th δ-halved chain $x^{(k)}$ of x with $O(nk) = O(n^3 \log(nD/\epsilon))$ oracle calls, from (ii); its length $\ell(x^{(k)})$ is at most $d(p,q) + \epsilon$, from (i). ◀

We end this section by showing the lemma used in the proof of Lemma 2.6. Let A_n be an $n \times n$ matrix whose (i,j) entry is defined by

$$(A_n)_{ij} := \begin{cases} (1/2)^{n+2-i-j} & (i+j \leq n+1), \\ 0 & (\text{otherwise}) \end{cases} \tag{9}$$

for $i, j = 1, 2, \ldots, n$. Since A_n is a nonnegative matrix, its spectral radius $\rho(A_n)$ is at most the maximum row sum of A_n, which immediately yields that $\rho(A_n) \leq 1 - (1/2)^n$. This inequality, however, is not tight unless $n = 1$. In fact, one can obtain a more useful upper bound of $\rho(A_n)$.

▶ **Lemma 2.7.** *Let $n > 1$ be an integer, and let A_n be an $n \times n$ matrix defined by (9). Then its spectral radius $\rho(A_n)$ is at most $1 - 1/n^2$. In addition, one has $(I - A_n)^{-1}\mathbf{1} \leq (5n^2/4)\mathbf{1}$ and $(A_n)^k \mathbf{1} \leq (5/4)e^{-k/n^2}\mathbf{1}$ for any integer $k \geq 0$.*

Proof. Let $A := A_n$ for simplicity. Let u be a positive column vector of dimension n whose k-th entry is defined by $u_k := k(n-k) + n^2$ for $k = 1, 2, \ldots, n$. By the Collatz–Wielandt inequality, in order to show $\rho(A) \leq 1 - 1/n^2$ it suffices to show that $Au \leq (1 - 1/n^2)u$. The k-th entry of the vector Au is

$$(Au)_k = \sum_{j=1}^{n+1-k} \frac{u_j}{2^{n+2-k-j}} = \frac{1}{2^{n+2-k}} \sum_{j=1}^{n+1-k} 2^j(-j^2 + nj + n^2).$$

Hence, using the general formulas

$$\sum_{j=1}^{m} j \cdot 2^j = 2 + 2^{m+1}(m-1) \quad \text{and} \quad \sum_{j=1}^{m} j^2 \cdot 2^j = -6 + 2^{m+1}((m-1)^2 + 2),$$

we have

$$(Au)_k = u_k - 2 - \frac{n^2 - n - 3}{2^{n+1-k}}.$$

It is easy to see that for $n \geq 2$ and $1 \leq k \leq n$ one has

$$\frac{u_k}{n^2} = 1 + \frac{k(n-k)}{n^2} \leq \frac{5}{4} \leq \left(2 - \frac{1}{2^{n+1-k}}\right) + \frac{(n-2)(n+1)}{2^{n+1-k}},$$

which implies that

$$\frac{u_k}{n^2} \leq 2 + \frac{n^2 - n - 3}{2^{n+1-k}} \quad (k = 1, 2, \ldots, n).$$

This completes the proof of the inequality $Au \leq (1 - 1/n^2)u$.

Let us show the latter part of the lemma. Note that $\mathbf{1} \leq (1/n^2)u \leq (5/4)\mathbf{1}$. Since $(1/n^2)u \leq (I-A)u$ and $(I-A)^{-1}$ is a nonnegative matrix (as $\rho(A) < 1$), we have $(I-A)^{-1}\mathbf{1} \leq (1/n^2)(I-A)^{-1}u \leq u \leq (5n^2/4)\mathbf{1}$.

Since $Au \leq (1-1/n^2)u \leq e^{-1/n^2}u$, we have $A^k u \leq e^{-k/n^2}u$ for any integer $k \geq 0$. Hence, $A^k\mathbf{1} \leq (1/n^2)A^k u \leq (1/n^2)e^{-k/n^2}u \leq (5/4)e^{-k/n^2}\mathbf{1}$. ◀

▶ Remark. In proving Theorem 2.4, we utilized only the convexity of the metric of X. Hence our algorithm works even when X is a Busemann convex space.

3 Computing geodesics in CAT(0) cubical complexes

In this section we give an algorithm to compute geodesics in CAT(0) cubical complexes, with an aid of the result of the preceding section. In Section 3.1 to 3.4, we recall CAT(0) cubical complexes, median graphs, PIPs and CAT(0) orthant spaces. Section 3.5 is devoted to proving our main theorem.

3.1 CAT(0) cubical complex

A *cubical complex* $\mathcal{K}$ is a polyhedral complex where each k-dimensional cell is isometric to the unit cube $[0,1]^k$ and the intersection of any two cells is empty or a single face. The *underlying graph* of $\mathcal{K}$ is the graph $G(\mathcal{K}) = (V(\mathcal{K}), E(\mathcal{K}))$, where $V(\mathcal{K})$ denotes the set of *vertices* (0-dimensional faces) of $\mathcal{K}$ and $E(\mathcal{K})$ denotes the set of *edges* (1-dimensional faces) of $\mathcal{K}$. A cubical complex $\mathcal{K}$ has an intrinsic metric induced by the l_2-metric on each cell. For two points $p, q \in \mathcal{K}$, a *string* in $\mathcal{K}$ from p to q is a sequence of points $p = x_0, x_1, \ldots, x_{m-1}, x_m = q$ in $\mathcal{K}$ such that for each $i = 0, 1, \ldots, m-1$ there exists a cell C_i containing x_i and x_{i+1}, and its *length* is defined to be $\sum_{i=0}^{m-1} d(x_i, x_{i+1})$, where $d(x_i, x_{i+1})$ is measured inside C_i by the l_2-metric. The distance between two points $p, q \in \mathcal{K}$ is defined to be the infimum of the lengths of strings from p to q.

Gromov [14] gave a combinatorial criterion which allows us to easily decide whether or not a cubical complex $\mathcal{K}$ is non-positively curved. The *link* of a vertex v of $\mathcal{K}$ is the abstract simplicial complex whose vertices are the edges of $\mathcal{K}$ containing v and where k edges $e_1, \ldots, e_k$ span a simplex if and only if they are contained in a common k-dimensional cell of $\mathcal{K}$. An abstract simplicial complex $\mathcal{L}$ is called *flag* if any set of vertices is a simplex of $\mathcal{L}$ whenever each pair of its vertices spans a simplex.

▶ **Theorem 3.1** (Gromov [14]). *A cubical complex $\mathcal{K}$ is CAT(0) if and only if $\mathcal{K}$ is simply connected and the link of each vertex is flag.*

3.2 Median graph

Let $G = (V, E)$ be a simple undirected graph. The distance $d_G(u, v)$ between two vertices u and v is the length of a shortest path between u and v. The *interval* $I_G(u, v)$ between u and v is the set of vertices $w \in V$ with $d_G(u, v) = d_G(u, w) + d_G(w, v)$. A vertex subset $U \subseteq V$ is said to be *convex* if $I_G(u, v)$ is contained in U for all $u, v \in U$. A graph G is called a *median graph* if for all $u, v, w \in V$ the set $I_G(u, v) \cap I_G(v, w) \cap I_G(w, u)$ contains exactly one element, called the *median* of u, v, w. Median graphs are connected and bipartite. A *median complex* is a cubical complex derived from a median graph G by replacing all cube-subgraphs of G by solid cubes. It has been shown independently by Chepoi [9] and Roller [22] that median complexes and CAT(0) cubical complexes constitute the same objects:

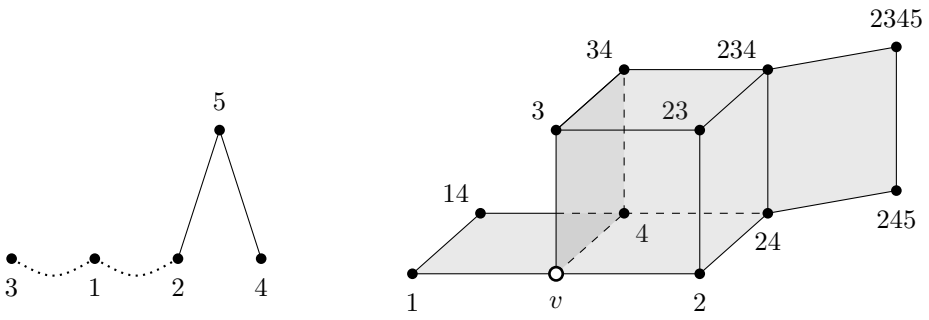


Figure 4 A poset with inconsistent pairs and the corresponding rooted CAT(0) cubical complex. Dotted line represents minimal inconsistent pairs, where an inconsistent pair $\{a, b\}$ is said to be *minimal* if there is no other inconsistent pair $\{a', b'\}$ with $a' \preceq a$ and $b' \preceq b$.

▶ **Theorem 3.2** (Chepoi [9], Roller [22]). *The underlying graph of every CAT(0) cubical complex is a median graph, and conversely, every median complex is a CAT(0) cubical complex.*

3.3 Poset with inconsistent pairs (PIP)

Barthélemy and Constantin [4] established a Birkhoff-type representation theorem for median graphs, by employing a poset with an additional relation. This structure was rediscovered by Ardila et al. [2] in the context of CAT(0) cubical complexes. An *antichain* of a poset P is a subset of P that contains no two comparable elements. A subset I of P is called an *order ideal* of P if $a \in I$ and $b \preceq a$ imply $b \in I$. A poset P is *locally finite* if every interval $[a, b] = \{c \in P \mid a \preceq c \preceq b\}$ is finite, and it has *finite width* if every antichain is finite.

▶ **Definition 3.3.** A *poset with inconsistent pairs* (or, briefly, a *PIP*) is a locally finite poset P of finite width, endowed with a symmetric binary relation $\smile$ satisfying:
1) If $a \smile b$, then a and b are incomparable.
2) If $a \smile b$, $a \preceq a'$ and $b \preceq b'$, then $a' \smile b'$.
A pair $\{a, b\}$ with $a \smile b$ is called an *inconsistent pair*. An order ideal of P is called *consistent* if it contains no inconsistent pairs.

For a CAT(0) cubical complex $\mathcal{K}$ and a vertex v of $\mathcal{K}$, the pair $(\mathcal{K}, v)$ is called a *rooted CAT(0) cubical complex*. Given a poset with inconsistent pairs P, one can construct a cubical complex $\mathcal{K}_P$ as follows: The underlying graph $G(\mathcal{K}_P)$ is a graph G_P whose vertices are consistent order ideals of P and where two consistent order ideals I, J are adjacent if and only if $|I \Delta J| = 1$; replace all cube-subgraphs (i.e., subgraphs isomorphic to cubes of some dimensions) of G_P by solid cubes. See Figure 4 for an example. In fact, the resulting cubical complex $\mathcal{K}_P$ is CAT(0), and moreover:

▶ **Theorem 3.4** (Ardila et al. [2]). *The map $P \mapsto \mathcal{K}_P$ is a bijection between posets with inconsistent pairs and rooted CAT(0) cubical complexes.*

This bijection can also be derived from Theorem 3.2 and the result of Barthélemy and Constantin [4], who found a bijection between PIPs and pointed median graphs.

Given a poset with inconsistent pairs P, one can embed $\mathcal{K}_P$ into a unit cube in the Euclidean space as follows, which we call the *standard embedding* of P [2]:

$$\mathcal{K}_P = \{(x_i)_{i \in P} \in [0,1]^P \mid i \prec j \text{ and } x_i < 1 \Rightarrow x_j = 0, \text{ and } i \smile j \Rightarrow x_i x_j = 0\}.$$

For each pair (I, M) of a consistent order ideal I of P and a subset $M \subseteq I_{\max}$, where $I_{\max}$ is the set of maximal elements of I, the subspace

$$C^I_M := \{x \in \mathcal{K}_P \mid i \in I \backslash M \Rightarrow x_i = 1, \text{ and } i \notin I \Rightarrow x_i = 0\} = \{1\}^{I \backslash M} \times [0,1]^M \times \{0\}^{P \backslash I}$$

corresponds to a unique $|M|$-dimensional cell of $\mathcal{K}_P$.

3.4 CAT(0) orthant space

Let $\mathbb{R}_+$ denote the set of nonnegative real numbers. Let $\mathcal{L}$ be an abstract simplicial complex on a finite set V. The *orthant space* $\mathcal{O}(\mathcal{L})$ for $\mathcal{L}$ is a subspace of $|V|$-dimensional orthant $\mathbb{R}^V_+$ constructed by taking a union of all subcones $\{\mathcal{O}_S \mid S \in \mathcal{L}\}$ associated with simplices of $\mathcal{L}$, where $\mathcal{O}_S$ is defined by $\mathcal{O}_S := \mathbb{R}^S_+ \times \{0\}^{V \backslash S}$ for each simplex $S \in \mathcal{L}$; namely, $\mathcal{O}(\mathcal{L}) = \bigcup_{S \in \mathcal{L}} \{x \in \mathbb{R}^V_+ \mid x_v = 0 \text{ for each } v \notin S\}$. The distance between two points $x, y \in \mathcal{O}(\mathcal{L})$ is defined in a similar way as in the case of cubical complexes. An orthant space is a special instance of cubical complexes.

▶ **Theorem 3.5** (Gromov [14]). *The orthant space $\mathcal{O}(\mathcal{L})$ for an abstract simplicial complex $\mathcal{L}$ is a CAT(0) space if and only if $\mathcal{L}$ is a flag complex.*

A typical example of CAT(0) orthant spaces is a *tree space* [6]. Owen and Provan [19, 20] gave a polynomial time algorithm to compute geodesics in tree spaces, which was generalized to CAT(0) orthant spaces by Miller et al. [15].

▶ **Theorem 3.6** ([15, 19, 20]). *Let $\mathcal{L}$ be a flag abstract simplicial complex on a finite set V and $\mathcal{O}(\mathcal{L})$ be the CAT(0) orthant space for $\mathcal{L}$. Let $x, y \in \mathcal{O}(\mathcal{L})$, and let S_1 and S_2 be the inclusion-wise minimal simplices such that $x \in \mathcal{O}_{S_1}$ and $y \in \mathcal{O}_{S_2}$. Then one can find the explicit description of the geodesic joining x and y in $O((|S_1| + |S_2|)^4)$ time.*

An interesting thing about their algorithm is that it solves as a subproblem a combinatorial optimization problem: the Maximum Weight Stable Set problem on a bipartite graph whose color classes have at most $|S_1|, |S_2|$ vertices, respectively. We should note that the above explicit descriptions of geodesics are radical expressions. Computationally, for a point p on a geodesic, one can compute a rational point $p' \in \mathcal{O}(\mathcal{L})$ such that $d(p', p) \leq \delta$ and the number of bits required for each coordinate of p' is bounded by $O(\log(|V|/\delta))$. For a real number $r > 0$, the subspace $\mathcal{O}(\mathcal{L}) \cap [0, r]^V$ of $\mathcal{O}(\mathcal{L})$, denoted by $\mathcal{O}(\mathcal{L})|_{[0,r]}$, is called a *truncated CAT(0) orthant space*. Actually, from the explicit descriptions of geodesics in $\mathcal{O}(\mathcal{L})$, one can see that $\mathcal{O}(\mathcal{L})|_{[0,r]}$ is convex in $\mathcal{O}(\mathcal{L})$, and thus:

▶ **Theorem 3.7** ([15]). *Given two points x, y in a truncated CAT(0) orthant space $\mathcal{O}(\mathcal{L})|_{[0,r]}$, one can find the explicit description of the geodesic joining x and y in $O(|V|^4)$ time.*

3.5 Main theorem

Our main result is the following theorem. It should be remarked that as stated in [2] there are no simple formulas for the breakpoints in geodesics in CAT(0) cubical complexes due to their algebraic complexity, and hence one can only compute them approximately. Also note that for the shortest path problem in a general CAT(0) cubical complex there has been no algorithm that runs in time polynomial in the size of the complex, much less the size of the compact representation PIP.

▶ **Problem 3.8.** Given a poset with inconsistent pairs P, two points p, q in the standard embedding $\mathcal{K}_P$ of P, and a positive parameter $\epsilon > 0$, find a sequence of points $p = x_0, x_1, \ldots, x_{n-1}, x_n = q$ in $\mathcal{K}_P$ with $\sum_{i=0}^{n-1} d(x_i, x_{i+1}) \leq d(p, q) + \epsilon$ and compute the geodesic joining x_i and x_{i+1} for $i = 0, 1, \ldots, n-1$.

▶ **Theorem 3.9.** *Problem 3.8 can be solved in $O(|P|^7 \log(|P|/\epsilon))$ time. Moreover, the number of bits required for each coordinate of points in $\mathcal{K}_P$ occurring throughout the algorithm can be bounded by $O(\log(|P|/\epsilon))$.*

Let us show this theorem. Let m denote the number of elements of P and let $D < 1$ be a positive constant close to 1 (e.g., set $D := 0.9$). Theorem 2.4 implies that in order to prove Theorem 3.9 it suffices to show that:

(a) Given two points $p, q \in \mathcal{K}_P$, one can find a sequence of points $p = x_0, x_1, \ldots, x_{n-1}, x_n = q$ in $\mathcal{K}_P$ such that $n = O(m)$ and $d(x_i, x_{i+1}) \leq D/4 - \epsilon$ for $i = 0, 1, \ldots, n-1$.

(b) Given two points $p, q \in \mathcal{K}_P$ with $d(p, q) \leq D$, one can compute the geodesic joining p and q in $O(m^4)$ time and find a δ-midpoint w of p and q with $O(\log(m/\delta))$ bits enough for each coordinate of w.

It is relatively easy to show (a), by considering a curve $c(p, q)$ issuing at p, going through an edge geodesic (a shortest path in the underlying graph of $\mathcal{K}_P$) between some vertices of cells containing p, q, and ending at q. (Note that one can easily find an edge geodesic between vertices u and v of $\mathcal{K}_P$. Reroot the complex $\mathcal{K}_P$ at u. In other words, construct a poset P' for which $\mathcal{K}_{P'} \cong \mathcal{K}_P$ and u is the root of $\mathcal{K}_{P'}$; this construction is implicitly stated in [2]. Then the edge geodesic in $\mathcal{K}_{P'}$ from the root $u = \emptyset$ to $v = I$, where I is a consistent order ideal of P', can be found by considering a linear extension of the elements of I.) Since such a curve $c(p, q)$ has length at most $O(m)$, dividing it into parts appropriately, one can get a desired sequence of points. To show (b), we need the following two lemmas, whose detailed proofs can be found in the full version of this work.

▶ **Lemma 3.10.** *Let $\mathcal{K}$ be a CAT(0) cubical complex and v be a vertex of $\mathcal{K}$. Then the star $\mathrm{St}(v, \mathcal{K})$ of v in $\mathcal{K}$, i.e., the subcomplex spanned by all cells containing v, is convex in $\mathcal{K}$.*

Sketch of Proof. The lemma follows from the well-known fact that the vertex set of a star in $\mathcal{K}$ is convex in the underlying graph $G(\mathcal{K})$ and the result of [10] that the subcomplex $\mathcal{K}(S)$ of $\mathcal{K}$ induced by a convex vertex subset S of $G(\mathcal{K})$ is convex in $\mathcal{K}$ in the ℓ_2-metric. ◀

▶ **Lemma 3.11.** *Let $\mathcal{K}$ be a CAT(0) cubical complex. Let p, q be two points in $\mathcal{K}$ with $d(p, q) < 1$ and R_1, R_2 be the minimal cells of $\mathcal{K}$ containing p, q, respectively. Then $R_1 \cap R_2 \neq \emptyset$.*

Sketch of Proof. One can show that there exists a vertex u_i of R_i for $i = 1, 2$, such that $d(u_1, u_2) = d(R_1, R_2) := \inf_{x \in R_1, y \in R_2} d(x, y)$. Since $d(R_1, R_2) \leq d(p, q) < 1$, one has $d(u_1, u_2) < 1$. Hence u_1 and u_2 should be the same vertex, and thus $R_1 \cap R_2 \neq \emptyset$. ◀

Using these lemmas, we show (b). Suppose that we are given two points $p, q \in \mathcal{K}_P$ with $d(p, q) \leq D$. First notice that one can find in linear time the minimal cells R_1 and R_2 of $\mathcal{K}_P$ that contain p and q, respectively, just by checking their coordinates. (Indeed, one has $R_1 = C_M^I$ for $I = \{i \in P \mid p_i > 0\}$ and $M = \{i \in P \mid 0 < p_i < 1\}$.) Since $d(p, q) \leq D < 1$, from Lemma 3.11 we know that $R_1 \cap R_2 \neq \emptyset$. Let v be a vertex of $R_1 \cap R_2$. Then p and q are contained in the star $\mathrm{St}(v, \mathcal{K}_P)$ of v. Since $\mathrm{St}(v, \mathcal{K}_P)$ is convex in $\mathcal{K}_P$ by Lemma 3.10, we only have to compute the geodesic in $\mathrm{St}(v, \mathcal{K}_P)$. Obviously, $\mathrm{St}(v, \mathcal{K}_P)$ is a truncated CAT(0) orthant space, and hence one can compute the geodesic between p and q in $\mathrm{St}(v, \mathcal{K}_P)$ in $O(m^4)$ time, by Theorem 3.7. In addition, one can find a δ-midpoint $w \in \mathrm{St}(v, \mathcal{K}_P)$ of p and q such that the number of bits required for each coordinate of w is bounded by $O(\log(m/\delta))$. This implies (b) and therefore completes the proof of Theorem 3.9.

References

1 Aaron Abrams and Robert Ghrist. State complexes for metamorphic robots. *The International Journal of Robotics Research*, 23:811–826, 2004.

2 Federico Ardila, Megan Owen, and Seth Sullivant. Geodesics in CAT(0) cubical complexes. *Adv. in Appl. Math.*, 48:142–163, 2012.

3 Hans-Jürgen Bandelt and Victor Chepoi. Metric graph theory and geometry: a survey. In Jacob E. Goodman, János Pach, and Richard Pollack, editors, *Surveys on discrete and computational geometry*, volume 453 of *Contemp. Math.*, pages 49–86. Amer. Math. Soc., Providence, RI, 2008.

4 Jean-Pierre Barthélemy and Julien Constantin. Median graphs, parallelism and posets. *Discrete Math.*, 111:49–63, 1993.

5 Miroslav Bačák. *Convex analysis and optimization in Hadamard spaces*. De Gruyter, Berlin, 2014.

6 Louis J. Billera, Susan P. Holmes, and Karen Vogtmann. Geometry of the space of phylogenetic trees. *Adv. in Appl. Math.*, 27:733–767, 2001.

7 Martin R. Bridson and André Haefliger. *Metric spaces of non-positive curvature*. Springer-Verlag, Berlin, 1999.

8 John Canny and John Reif. New lower bound techniques for robot motion planning problems. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 49–60, 1987.

9 Victor Chepoi. Graphs of some CAT(0) complexes. *Adv. in Appl. Math.*, 24:125–179, 2000.

10 Victor Chepoi and Daniela Maftuleac. Shortest path problem in rectangular complexes of global nonpositive curvature. *Comput. Geom.*, 46:51–64, 2013.

11 S. M. Gersten and H. Short. Small cancellation theory and automatic groups. II. *Invent. Math.*, 105:641–662, 1991.

12 R. Ghrist and V. Peterson. The geometry and topology of reconfiguration. *Adv. in Appl. Math.*, 38:302–323, 2007.

13 Robert Ghrist and Steven M. LaValle. Nonpositive curvature and Pareto optimal coordination of robots. *SIAM J. Control Optim.*, 45:1697–1713, 2006.

14 M. Gromov. Hyperbolic groups. In S. M. Gersten, editor, *Essays in group theory*, volume 8 of *Math. Sci. Res. Inst. Publ.*, pages 75–263. Springer, New York, 1987.

15 Ezra Miller, Megan Owen, and J. Scott Provan. Polyhedral computational geometry for averaging metric phylogenetic trees. *Adv. in Appl. Math.*, 68:51–91, 2015.

16 Joseph S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of computational geometry*, pages 633–701. North-Holland, Amsterdam, 2000.

17 Joseph S. B. Mitchell and Micha Sharir. New results on shortest paths in three dimensions. In *Proceedings of the 20th Annual Symposium on Computational Geometry*, pages 124–133, 2004.

18 Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures and domains. I. *Theoret. Comput. Sci.*, 13:85–108, 1981.

19 Megan Owen. Computing geodesic distances in tree space. *SIAM J. Discrete Math.*, 25:1506–1529, 2011.

20 Megan Owen and J. Scott Provan. A fast algorithm for computing geodesic distances in tree space. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8:2–13, 2011.

21 Valentin Polishchuk and Joseph S. B. Mitchell. Touring convex bodies - A conic programming solution. In *Proceedings of the 17th Canadian Conference on Computational Geometry*, pages 101–104, 2005.

22 Martin Roller. Poc sets, median algebras and group actions. preprint, University of Southampton, 1998.

23 Michah Sageev. Ends of group pairs and non-positively curved cube complexes. *Proc. London Math. Soc.*, 71:585–617, 1995.

24 Micha Sharir. On shortest paths amidst convex polyhedra. *SIAM J. Comput.*, 16:561–572, 1987.

Online Vertex-Weighted Bipartite Matching: Beating $1 - \frac{1}{e}$ with Random Arrivals

Zhiyi Huang[1]
Department of Computer Sicence, The University of Hong Kong, Hong Kong
zhiyi@cs.hku.hk

Zhihao Gavin Tang[2]
Department of Computer Sicence, The University of Hong Kong, Hong Kong
zhtang@cs.hku.hk

Xiaowei Wu[3]
Department of Computing, The Hong Kong Polytechnic University, Hong Kong
wxw0711@gmail.com

Yuhao Zhang
Department of Computer Sicence, The University of Hong Kong, Hong Kong
yhzhang2@cs.hku.hk

Abstract

We introduce a weighted version of the ranking algorithm by Karp et al. (STOC 1990), and prove a competitive ratio of 0.6534 for the vertex-weighted online bipartite matching problem when online vertices arrive in random order. Our result shows that random arrivals help beating the 1-1/e barrier even in the vertex-weighted case. We build on the randomized primal-dual framework by Devanur et al. (SODA 2013) and design a two dimensional gain sharing function, which depends not only on the rank of the offline vertex, but also on the arrival time of the online vertex. To our knowledge, this is the first competitive ratio strictly larger than 1-1/e for an online bipartite matching problem achieved under the randomized primal-dual framework. Our algorithm has a natural interpretation that offline vertices offer a larger portion of their weights to the online vertices as time goes by, and each online vertex matches the neighbor with the highest offer at its arrival.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis, Theory of computation → Online algorithms, Theory of computation → Linear programming

Keywords and phrases Vertex Weighted, Online Bipartite Matching, Randomized Primal-Dual

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.79

Related Version A full version of the paper can be found at `https://arxiv.org/abs/1804.07458`.

Acknowledgements The first author would like to thank Nikhil Devanur, Ankit Sharma, and Mohit Singh with whom he made an initial attempt to reproduce the results of Mahdian and Yan using the randomized primal-dual framework.

[1] Partially supported by the Hong Kong RGC under the grant HKU17202115E.
[2] Partially supported by his supervisor Hubert Chan's Hong Kong RGC grant 17202715.
[3] Part of the work was done when the author was a postdoc at the University of Hong Kong.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 79; pp. 79:1–79:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

With a wide range of applications, Online Bipartite Matching and its variants are a focal point in the online algorithms literature. Consider a bipartite graph $G(L \cup R, E)$ on vertices $L \cup R$, where the set L of offline vertices is known in advance and vertices in R arrive online. On the arrival of an online vertex, its incident edges are revealed and the algorithm must irrevocably either match it to one of its unmatched neighbors or leave it unmatched. In a seminal paper, Karp et al. [19] proposed the Ranking algorithm, which picks at the beginning a random permutation over the offline vertices L, and matches each online vertex to the first unmatched neighbor according to the permutation. They proved a tight competitive ratio $1 - \frac{1}{e}$ of Ranking, when online vertices arrive in an arbitrary order. The analysis has been simplified in a series of subsequent works [14, 5, 12]. Further, the Ranking algorithm has been extended to other variants of the Online Bipartite Matching problem, including the vertex-weighted case [2], the random arrival model [18, 21], and the Adwords problem [23, 7, 11].

As a natural generalization, Online Vertex-Weighted Bipartite Matching was considered by Aggarwal et al. [2]. In this problem, each offline vertex $v \in L$ has a non-negative weight w_v, and the objective is to maximize the total weight of the matched offline vertices. A weighted version of the Ranking algorithm was proposed in [2] and shown to be $(1 - \frac{1}{e})$-competitive, matching the problem hardness in the unweighted version. They fix a non-increasing perturbation function $\psi : [0, 1] \to [0, 1]$, and draw a rank $y_v \in [0, 1]$ uniformly and independently for each offline vertex $v \in L$. The offline vertices are then sorted in decreasing order of the *perturbed weight* $w_v \cdot \psi(y_v)$. Each online vertex matches the first unmatched neighbor on the list upon its arrival. It is shown that by choosing the perturbation function $\psi(y) := 1 - e^{y-1}$, the weighted Ranking algorithm achieves a tight competitive ratio $1 - \frac{1}{e}$. In a subsequent work, Devanur et al. [12] simplified the analysis under the randomized primal-dual framework and gave an alternative interpretation of the algorithm: each offline vertex v makes an offer of value $w_v \cdot (1 - g(y_v))$ as long as it is not matched, where $g(y) := e^{y-1}$, and each online vertex matches the neighbor that offers the highest.

Motivated by the practical importance of Online Bipartite Matching and its applications for online advertisements, another line of research seeks for a better theoretical bound beyond the worst-case hardness result provided by Karp et al. [19]. Online Bipartite Matching with random arrivals was considered independently by Karande et al. [18] and Mahdian et al. [21]. They both studied the performance of Ranking assuming that online vertices arrive in a uniform random order and proved competitive ratios 0.653 and 0.696 respectively. On the negative side, Karande et al. [18] explicitly constructed an instance for which Ranking performs no better than 0.727, which is later improved to 0.724 by Chan et al. [9]. In terms of problem hardness, Manshadi et al. [22] showed that no algorithm can achieve a competitive ratio larger than 0.823.

The natural next step is then to consider Online Vertex-Weighted Bipartite Matching with random arrivals. *Do random arrivals help beating* $1 - \frac{1}{e}$ *even in the vertex-weighted case?*

	Arbitrary Arrivals	Random Arrivals
Unweighted	$1 - \frac{1}{e} \approx 0.632$ [19, 5, 12, 14]	0.696 [21]
Vertex-weighted	$1 - \frac{1}{e} \approx 0.632$ [2, 12]	**0.6534 (this paper)**

1.1 Our Results and Techniques

We answer this affirmatively by showing that a generalized version of the Ranking algorithm achieves a competitive ratio 0.6534.

▶ **Theorem 1.** *There exists a* 0.6534*-competitive algorithm for the vertex-weighted Online Bipartite Matching with random arrivals.*

Interestingly, we do not obtain our result by generalizing existing works that break the $1 - \frac{1}{e}$ barrier on the unweighted case [18, 21] to the vertex-weighted case. Instead, we take a totally different path, and build our analysis on the randomized primal-dual technique introduced by Devanur et al. [12], which was used to provide a more unified analysis of the algorithms for the Online Bipartite Matching with arbitrary arrival order and its extensions.

We first briefly review the proof of Devanur et al. [12]. The randomized primal-dual technique can be viewed as a charging argument for sharing the gain of each matched edge between its two endpoints. Recall that in the algorithm of [2, 12], each unmatched offline vertex offers a value of $w_v \cdot (1 - g(y_v))$ to online vertices, and each online vertex matches the neighbor that offers the highest at its arrival. Whenever an edge (u, v) is added to the matching, where $v \in L$ is an offline vertex and $u \in R$ is an online vertex, imagine a total gain of w_v being shared between u and v such that u gets $w_v \cdot (1 - g(y_v))$ and v gets $w_v \cdot g(y_v)$. Since g is non-decreasing, the smaller the rank of v, the smaller share it gets. They showed that by fixing $g(y) = e^{y-1}$, for any edge (u, v) and any fixed ranks of offline vertices other than v, the expected gains of u and v (from all of their incident edges) combined is at least $(1 - \frac{1}{e}) \cdot w_v$ over the randomness of y_v, which implies the $1 - \frac{1}{e}$ competitive ratio.

Now we consider the problem with random arrivals.

Analogous to the offline vertices, as the online vertices arrive in random order, in the gain sharing process, it is natural to give an online vertex u a smaller share if u arrives early (as it is more likely be get matched), and a larger share when u arrives late. Thus we consider the following version of the weighted Ranking algorithm.

Let y_u be the arrival time of online vertex $u \in R$, which is chosen uniformly at random from $[0, 1]$. Analogous to the ranks of the offline vertices, we also call y_u the rank of $u \in R$. Fix a function $g : [0, 1]^2 \to [0, 1]$ that is non-decreasing in the first dimension and non-increasing in the second dimension. On the arrival of $u \in R$, each unmatched neighbor $v \in L$ of u makes an offer of value $w_v \cdot (1 - g(y_v, y_u))$, and u matches the neighbor with the highest offer. This algorithm straightforwardly leads to a gain sharing rule for dual assignments: whenever $u \in R$ matches $v \in L$, let the gain of u be $w_v \cdot (1 - g(y_v, y_u))$ and the gain of v be $w_v \cdot g(y_v, y_u)$. It suffices to show that, for an appropriate function g, the expected gain of u and v combined is at least $0.6534 \cdot w_v$ over the randomness of both y_u and y_v.

The main difficulty of the analysis is to give a good characterization of the behavior of the algorithm when we vary the ranks of both $u \in R$ and $v \in L$, while fixing the ranks of all other vertices arbitrarily. The previous analysis for the unweighted case with random arrivals [18, 21] heavily relies on a symmetry between the random ranks of offline vertices and online vertices: Properties developed for the offline vertices in previous work directly translate to their online counterparts. Unfortunately, the online and offline sides are no longer symmetric in the vertex-weighted case. In particular, for the offline vertex v, an important property is that for any given rank y_u of the online vertex u, we can define a unique marginal rank θ such that v will be matched if and only if its rank $y_v < \theta$. However, it is not possible to define such a marginal rank for the online vertex u in the vertex-weighted case: As its arrival time changes, its matching status may change back and forth. The most important technical ingredient of our analysis is an appropriate lower bound on the expected gain

which allows us to partially characterize the worst-case scenario (in the sense of minimizing the lower bound on the expected gain). Further, the worst-case scenario does admit simple marginal ranks even for the online vertex u. This allows us to design a symmetric gain sharing function g and complete the competitive analysis of 0.6534.

1.2 Other Related Works

There is a vast literature on problems related to Online Bipartite Matching. For space reasons, we only list some of the most related here.

Kesselheim et al. [20] considered the edge-weighted Online Bipartite Matching with random arrivals, and proposed a $\frac{1}{e}$-competitive algorithm. The competitive ratio is tight as it matches the lower bound on the classical secretary problem [8]. Wang and Wong [24] considered a different model of Online Bipartite Matching with both sides of vertices arriving online (in an arbitrary order): A vertex can only actively match other vertices at its arrival; if it fails to match at its arrival, it may still get matched passively by other vertices later. They showed a 0.526-competitive algorithm for a fractional version of the problem.

Recently, Cohen and Wajc [10] considered the Online Bipartite Matching (with arbitrary arrival order) on regular graphs, and provided a $(1 - O(\sqrt{\log d/d}))$-competitive algorithm, where d is the degree of vertices. Very recently, Huang et al. [16] proposed a fully online matching model, in which all vertices of the graph arrive online (in an arbitrary order). Extending the randomized primal-dual technique, they obtained competitive ratios above 0.5 for both bipartite graphs and general graphs.

Similar but different from the Online Bipartite Matching with random arrivals, in the stochastic Online Bipartite Matching, the online vertices arrive according to some known probability distribution (with repetition). Competitive ratios breaking the $1 - \frac{1}{e}$ barrier have been achieved for the unweighted case [13, 4, 6] and the vertex-weighted case [15, 17, 6].

The Online Bipartite Matching with random arrivals is closely related to the oblivious matching problem [3, 9, 1] (on bipartite graphs). It can be easily shown that Ranking has equivalent performance on the two problems. Thus competitive ratios above $1 - \frac{1}{e}$ [18, 21] directly translate to the oblivious matching problem. Generalizations of the problem to arbitrary graphs have also been considered, and competitive ratios above half are achieved for the unweighted case [3, 9] and vertex-weighted case [1].

2 Preliminaries

We consider the Online Vertex-Weighted Bipartite Matching with random arrival order. Let $G(L \cup R, E)$ be the underlying graph, where vertices in L are given in advance and vertices in R arrive online in random order. Each offline vertex $v \in L$ is associated with a non-negative weight w_v. Without loss of generality, we assume the arrival time y_u of each online vertex $u \in R$ is drawn independently and uniformly from $[0, 1]$. Mahdian and Yan [21] use another interpretation for the random arrival model. They denote the order of arrival of online vertices by a permutation π and assume that π is drawn uniformly at random from the permutation group S_n. It is easy to see the equivalence between two interpretations[4].

[4] Mapping from an arrival time vector to a permutation is immediate. Given a permutation π, we independently draw n random variables uniformly from $[0, 1]$ and assign these values to be the arrival times of all vertices according to the permutation π, from the smallest to the largest.

Weighted Ranking. Fix a function $g : [0,1]^2 \to [0,1]$ such that $\frac{\partial g(x,y)}{\partial x} \geq 0$ and $\frac{\partial g(x,y)}{\partial y} \leq 0$. Each offline vertex $v \in L$ draws independently a random rank $y_v \in [0,1]$ uniformly at random. Upon the arrival of online vertex $u \in R$, u is matched to its unmatched neighbor v with maximum $w_v \cdot (1 - g(y_v, y_u))$.

▶ Remark. In the adversarial model, Aggarwal et al.'s algorithm [2] can be interpreted as choosing $g(y_v, y_u) := e^{y_v - 1}$ in our algorithm. Our algorithm is a direct generalization of theirs to the random arrival model.

For simplicity, for each $u \in R$, we also call its arrival time y_u the rank of u. We use $\vec{y} : L \cup R \to [0,1]$ to denote the vector of all ranks.

Consider the linear program relaxation of the bipartite matching problem and its dual.

$$\begin{array}{lllllll} \max: & \sum_{(u,v)\in E} w_v \cdot x_{uv} & & & \min: & \sum_{u\in V} \alpha_u & \\ \text{s.t.} & \sum_{v:(u,v)\in E} x_{uv} \leq 1 & \forall u \in L \cup R & & \text{s.t.} & \alpha_u + \alpha_v \geq w_v & \forall (u,v) \in E \\ & x_{uv} \geq 0 & \forall (u,v) \in E & & & \alpha_u \geq 0 & \forall u \in L \cup R \end{array}$$

Randomized Primal-Dual. Our analysis builds on the randomized primal-dual technique by Devanur et al. [12]. We set the primal variables according to the matching produced by Ranking, i.e. $x_{uv} = 1$ if and only if u is matched to v by Ranking, and set the dual variables so that the dual objective equals the primal. In particular, we split the gain w_v of each matched edge (u,v) between vertices u and v; the dual variable for each vertex then equals the share it gets. Given primal feasibility and equal objectives, the usual primal-dual techniques would further seek to show approximate dual feasibility, namely, $\alpha_u + \alpha_v \geq F \cdot w_v$ for every edge (u,v), where F is the target competitive ratio. Observe that the above primal and dual assignments are themselves random variables. Devanur et al. [12] claimed that the primal-dual argument goes through given approximate dual feasibility in expectation. We formulate this insight in the following lemma and include a proof for completeness.

▶ **Lemma 2.** *Ranking is F-competitive if we can set (non-negative) dual variables such that*

- $\sum_{(u,v)\in E} x_{uv} = \sum_{u\in V} \alpha_u$*; and*
- $\mathbf{E}_{\vec{y}}[\alpha_u + \alpha_v] \geq F \cdot w_v$ *for all* $(u,v) \in E$*.*

Proof. We can set a feasible dual solution $\tilde{\alpha}_u := \mathbf{E}_{\vec{y}}[\alpha_u]/F$ for all $u \in V$. It's feasible because we have $\tilde{\alpha}_u + \tilde{\alpha}_v = \mathbf{E}_{\vec{y}}[\alpha_u + \alpha_v]/F \geq w_v$ for all $(u,v) \in E$. Then by duality we know that the dual solution is at least the optimal primal solution PRIMAL, which is also at least the optimal offline solution of the problem: $\sum_{u\in V} \tilde{\alpha}_u \geq \mathsf{PRIMAL} \geq \mathsf{OPT}$. Then by the first assumption, we have $\mathsf{OPT} \leq \sum_{u\in V} \tilde{\alpha}_u = \sum_{u\in V} \frac{\mathbf{E}_{\vec{y}}[\alpha_u]}{F} = \frac{1}{F}\mathbf{E}_{\vec{y}}\left[\sum_{u\in V} \alpha_u\right] = \frac{1}{F}\mathbf{E}_{\vec{y}}\left[\sum_{(u,v)\in E} w_v \cdot x_{uv}\right] = \frac{1}{F}\mathbf{E}[\mathsf{ALG}]$, which implies an F competitive ratio. ◀

In the rest of the paper, we set

$$g(x,y) = \frac{1}{2}\big(h(x) + 1 - h(y)\big), \qquad \forall x, y \in [0,1]$$

where $h : [0,1] \to [0,1]$ is a non-decreasing function (to be fixed later) with $h'(x) \leq h(x)$ for all $x \in [0,1]$. Observe that $\frac{\partial g(x,y)}{\partial x} = \frac{1}{2}h'(x) \geq 0$ and $\frac{\partial g(x,y)}{\partial y} = -\frac{1}{2}h'(y) \leq 0$. By definition of g, we have $g(x,y) + g(y,x) = 1$. Moreover, for any $x, y \in [0,1]$, we have the following fact that will be useful for our analysis.

▶ **Claim 2.1.** $\frac{\partial g(x,y)}{\partial y} \geq g(x,y) - 1$.

Proof. $\frac{\partial g(x,y)}{\partial y} = -\frac{1}{2}h'(y) \geq -\frac{1}{2}h(y) \geq \frac{1}{2}(h(x) + 1 - h(y)) - 1 = g(x,y) - 1$. ◀

3 A Simple Lower Bound

In this section, we prove a slightly smaller competitive ratio, $\frac{5}{4} - e^{-0.5} \approx 0.6434$, as a warm-up of the later analysis.

We reinterpret our algorithm as follows. As time t goes, each unmatched offline vertex $v \in L$ is dynamically priced at $w_v \cdot g(y_v, t)$. Since g is non-increasing in the second dimension, the prices do not increase as time goes by. Upon the arrival of $u \in R$, u can choose from its unmatched neighbors by paying the corresponding price. The utility of u derived by choosing v equals $w_v - w_v \cdot g(y_v, y_u)$. Then u chooses the one that gives the highest utility. Recall that g is non-decreasing in the first dimension. Thus, u prefers offline vertices with smaller ranks, as they offer lower prices.

This leads to the following monotonicity property as in previous works [2, 12].

▶ **Fact 3.1** (Monotonicity). *For any $\vec{y}$, if $v \in L$ is unmatched when $u \in R$ arrives, then when y_v increases, v remains unmatched when u arrives. Equivalently, if $v \in L$ is matched when $u \in R$ arrives, then when y_v decreases, v remains matched when u arrives.*

Gain Sharing. The above interpretation induces a straightforward gain sharing rule: whenever $u \in R$ is matched to $v \in L$, let $\alpha_v := w_v \cdot g(y_v, y_u)$ and $\alpha_u := w_v \cdot (1 - g(y_v, y_u)) = w_v \cdot g(y_u, y_v)$.

Note that the gain of an offline vertex is larger if it is matched earlier, i.e., being matched earlier is more beneficial for offline vertices (α_v is larger). However, the fact does not hold for online vertices. For each online vertex $u \in R$, the earlier u arrives (smaller y_u is), the more offers u sees. On the other hand, the prices of offline vertices are higher when u comes earlier. Thus, it is not guaranteed that earlier arrival time y_u induces larger α_u.

This is where our algorithm deviates from previous ones [2, 12], in which the prices of offline vertices are static (independent of time). The above observation is crucial and necessary for breaking the $1 - \frac{1}{e}$ barrier in the random arrival model.

To apply Lemma 2, we consider a pair of neighbors $v \in L$ and $u \in R$. We fix an arbitrary assignment of ranks to all vertices but u, v. Our goal is to establish a lower bound of $\frac{1}{w_v} \cdot \mathbf{E}[\alpha_u + \alpha_v]$, where the expectation is simultaneously taken over y_u and y_v.

▶ **Lemma 3.** *For each $y \in [0, 1]$, there exist thresholds $1 \geq \theta(y) \geq \beta(y) \geq 0$ such that when u arrives at time $y_u = y$,*

- *if $y_v < \beta(y)$, v is matched when u arrives;*
- *if $y_v \in (\beta(y), \theta(y))$, v is matched to u;*
- *if $y_v > \theta(y)$, v is unmatched after u's arrival.*

Moreover, $\beta(y)$ is a non-decreasing function and if $\theta(x) = 1$ for some $x \in [0, 1]$, then $\theta(x') = 1$ for all $x' \geq x$.

Proof. Consider the moment when u arrives. By Fact 3.1, there exists a threshold $\beta(y_u)$ such that v is matched when u arrives iff $y_v < \beta(y_u)$. Now suppose $y_v > \beta(y_u)$, in which case v is unmatched when u arrives. Thus v is priced at $w_v \cdot g(y_v, y_u)$ and u can get utility $w_v \cdot g(y_u, y_v)$ by choosing v.

Recall that $g(y_u, y_v)$ is non-increasing in terms of y_v. Let $\theta(y_u) \geq \beta(y_u)$ be the minimum value of y_v such that v is not chosen by u. In other words, when $\beta(y_u) < y_v < \theta(y_u)$, v is matched to u and when $y_v > \theta(y_u)$, v is unmatched after u's arrival.

Next we show that β is a non-decreasing function of y_u. By definition, if $y_v < \beta(y_u)$, then v is matched when u arrives. Straightforwardly, when y_u increases to y'_u (arrives even

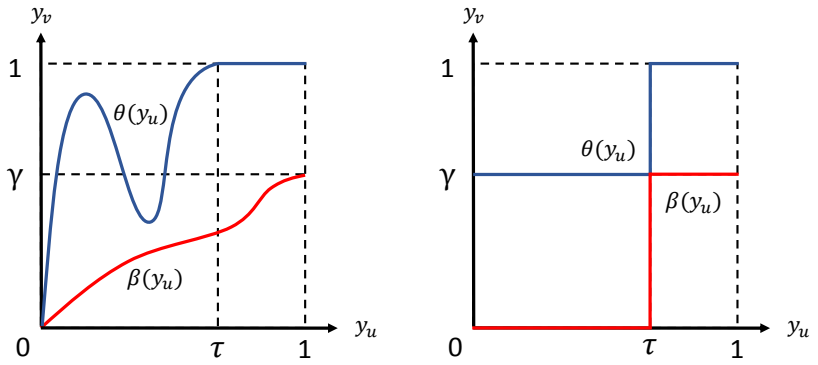


Figure 1 $\theta(y_u)$ and $\beta(y_u)$ (left hand side); truncated $\theta(y_u)$ and $\beta(y_u)$ (right hand side).

later), v would remain matched. Hence, we have $\beta(y'_u) \geq \beta(y_u)$ for all $y'_u > y_u$, i.e. β is non-decreasing (refer to Figure 1).

Finally, we show that if $\theta(x) = 1$ for some $x \in [0,1]$, then $\theta(x') = 1$ for all $x' \geq x$. Assume for the sake of contradiction that $\theta(x') < 1$ for some $x' > x$. In other words, when $y_u = x'$ and $y_v = 1$, v is unmatched when u arrives, but u chooses some vertex $z \neq v$, such that $w_z \cdot g(x', y_z) > w_v \cdot g(x', 1)$.

Now consider the case when u arrives at time $y_u = x$. Recall that we have $\theta(x) = 1$, which means that u matches v when $y_u = x$ and $y_v = 1$. By our assumption, both v and z are unmatched when u arrives at time x'. Thus when u arrives at an earlier time x, both v and z are unmatched. Moreover, choosing z induces utility

$$\begin{aligned} w_z \cdot g(x, y_z) &= w_z \cdot g(x', y_z) \cdot \frac{g(x, y_z)}{g(x', y_z)} > w_v \cdot g(x', 1) \cdot \frac{g(x, y_z)}{g(x', y_z)} \\ &= w_v \cdot g(x', 1) \cdot \frac{h(x) + 1 - h(y_z)}{h(x') + 1 - h(y_z)} \geq w_v \cdot g(x', 1) \cdot \frac{h(x) + 1 - h(1)}{h(x') + 1 - h(1)} \\ &= w_v \cdot g(x', 1) \cdot \frac{g(x, 1)}{g(x', 1)} = w_v \cdot g(x, 1), \end{aligned}$$

where the second inequality holds since h is a non-decreasing function and $x < x'$.

This gives a contradiction, since when $y_u = x$ and $y_v = 1$, u chooses v, while choosing z gives strictly higher utility. ◀

▶ Remark. Observe that the function θ is not necessarily monotone. This comes from the fact that u may prefer v to z when u arrives at time t but prefer z to v when u arrives later at time $t' > t$. Note that this happens only when the offline vertices have general weights: for the unweighted case, it is easy to show that θ must be non-decreasing.

We define $\tau, \gamma \in [0,1]$, which depend on the input instance, as follows.

If $\theta(y) < 1$ for all $y \in [0,1]$, then let $\tau = 1$; otherwise let τ be the minimum value such that $\theta(\tau) = 1$. Let $\gamma := \beta(1)$. Note that it is possible that $\gamma \in \{0,1\}$.

Since β is non-decreasing, we define $\beta^{-1}(x) := \sup\{y : \beta(y) = x\}$ for all $x \leq \gamma$.

In the following, we establish a lower bound for $\frac{1}{w_v} \cdot \mathbf{E}[\alpha_u + \alpha_v]$.

▶ **Lemma 4** (Main Lemma). *For each pair of neighbors $u \in R$ and $v \in L$, we have*

$$\frac{1}{w_v} \cdot \mathbf{E}[\alpha_u + \alpha_v] \geq \min_{0 \leq \gamma, \tau \leq 1} \left\{ (1-\tau) \cdot (1-\gamma) + \int_0^{\gamma} g(x, \tau) dx + \int_0^{\tau} g(x, \gamma) dx \right\}.$$

We prove Lemma 4 by the following three lemmas.

Observe that for any $y_u \in [0,1]$, if $y_v \in (\beta(y_u), \theta(y_u))$, u, v are matched to each other, which implies $\alpha_u + \alpha_v = w_v$. Hence we have the following lemma immediately.

▶ **Lemma 5** (Corner Gain). $\mathbf{E}\left[(\alpha_u + \alpha_v) \cdot \mathbb{1}(y_u > \tau, y_v > \gamma)\right] = w_v \cdot (1-\tau) \cdot (1-\gamma)$.

Now we give a lower bound for the gain of v when $y_v < \gamma$, i.e., $\alpha_v \cdot \mathbb{1}(y_v < \gamma)$, plus the gain of u when $y_v < \gamma$ and $y_u > \tau$, i.e., $\alpha_u \cdot \mathbb{1}(y_v < \gamma, y_u > \tau)$. The key to prove the lemma is to show that for all $y_v < \gamma$, no matter when u arrives, we always have $\alpha_v \geq w_v \cdot g(y_v, \beta^{-1}(y_v))$.

▶ **Lemma 6** (v's Gain). $\mathbf{E}\left[\alpha_v \cdot \mathbb{1}(y_v < \gamma) + \alpha_u \cdot \mathbb{1}(y_v < \gamma, y_u > \tau)\right] \geq w_v \cdot \int_0^\gamma g(x,\tau)dx$.

Proof. Fix $y_v = x < \gamma$. We first show that for all $y_u \in [0,1]$, $\alpha_v \geq w_v \cdot g(x, \beta^{-1}(x))$. By definition, we have $\beta^{-1}(x) < 1$. Hence when $y_u > \beta^{-1}(x)$, v is already matched when u arrives. Suppose v is matched to some $z \in R$, then we have $y_z \leq \beta^{-1}(x)$ and hence $\alpha_v \geq w_v \cdot g(x, \beta^{-1}(x))$. Now consider when u arrives at time $y < \beta^{-1}(x)$. If $y > y_z$, then v is still matched to z when u arrives, and $\alpha_v \geq w_v \cdot g(x, \beta^{-1}(x))$ holds. Now suppose $y < y_z$. We compare the two processes, namely when $y_u > \beta^{-1}(x)$ and when $y_u = y$.

We show that for each vertex $w \in L$, the time it is matched is not later in the second case (compared to the first case). In other words, we show that decreasing the rank of any online vertex is not harmful for all offline vertices. Suppose otherwise, let w be the first vertex in L that is matched later when $y_u = y$ than when $y_u > \beta^{-1}(x)$. I.e. among all these vertices, w's matched neighbor arrives the earliest when $y_u > \beta^{-1}(x)$.

Let u_1 be the vertex w is matched to when $y_u > \beta^{-1}(x)$ and u_2 be the vertex w is matched to when $y_u = y$. By assumption, we have $y_{u_2} > y_{u_1}$. Consider when $y_u = y$ and the moment when u_1 arrives, w remains unmatched but is not chosen by u_1. However, w is the first vertex that is matched later than it was when $y_u > \beta^{-1}(x)$, we know that at u_1's arrival, the set of unmatched neighbor of u_1 is a subset of that when $y_u > \beta^{-1}(x)$. This leads to a contradiction, since w gives the highest utility, but is not chosen by u_1.

In particular, this property holds for vertex v, i.e. v is matched earlier or at the arrival of z and hence $\alpha_v \geq w_v \cdot g(x, y_z) \geq w_v \cdot g(x, \beta^{-1}(x))$, as claimed.

Observe that for $y_v < \gamma$ and $y_u \in (\tau, \beta^{-1}(y_v))$, we have $\alpha_u + \alpha_v = w_v$. Thus for $y_v = x < \gamma$, we lower bound $\frac{1}{w_v} \cdot \mathbf{E}_{y_u}\left[\alpha_v \cdot \mathbb{1}(y_v < \gamma) + \alpha_u \cdot \mathbb{1}(y_v < \gamma, y_u > \tau)\right]$ by

$$f(x, \beta^{-1}(x)) := g(x, \beta^{-1}(x)) + \max\{0, \beta^{-1}(x) - \tau\} \cdot (1 - g(x, \beta^{-1}(x))).$$

It suffices to show that $f(x, \beta^{-1}(x)) \geq g(x, \tau)$. Consider the following two cases.

1. If $\beta^{-1}(x) < \tau$, then $f(x, \beta^{-1}(x)) = g(x, \beta^{-1}(x)) \geq g(x, \tau)$, since $\frac{\partial g(x,y)}{\partial y} \leq 0$.
2. If $\beta^{-1}(x) \geq \tau$, then $f(x, \beta^{-1}(x))$ is non-decreasing in the second dimension, since

$$\frac{\partial f(x, \beta^{-1}(x))}{\partial \beta^{-1}(x)} = \frac{\partial g(x, \beta^{-1}(x))}{\partial \beta^{-1}(x)} + 1 - g(x, \beta^{-1}(x)) - (\beta^{-1}(x) - \tau) \cdot \frac{\partial g(x, \beta^{-1}(x))}{\partial \beta^{-1}(x)} \geq 0,$$

where the inequality follows from Claim 2.1 and $\frac{\partial g(x, \beta^{-1}(x))}{\partial \beta^{-1}(x)} \leq 0$. Therefore, we have $f(x, \beta^{-1}(x)) \geq f(x, \tau) = g(x, \tau)$.

Hence for every fixed $y_v = x < \gamma$ we have $\mathbf{E}_{y_u}\left[\alpha_v \cdot \mathbb{1}(y_v < \gamma) + \alpha_u \cdot \mathbb{1}(y_v < \gamma, y_u > \tau)\right] \geq w_v \cdot g(x, \tau)$. Taking integration over $x \in (0, \gamma)$ concludes the lemma. ◀

Next we give a lower bound for the gain of u when $y_u < \tau$, i.e., $\alpha_u \cdot \mathbb{1}(y_u < \tau)$, plus the gain of v when $y_u < \tau$ and $y_v > \gamma$, i.e., $\alpha_v \cdot \mathbb{1}(y_u < \tau, y_v > \gamma)$. The following proof is in the

same spirit as in the proof of Lemma 6, although the ranks of offline vertices have different meaning from the ranks (arrival times) of online vertices.

Similar to the proof of Lemma 6, the key is to show that for all $y_u < \tau$, no matter what value y_v is, the gain of α_u is always at least $w_v \cdot g(y_u, \theta(y_u))$.

▶ **Lemma 7** (u's Gain). $\mathbf{E}\left[\alpha_u \cdot \mathbb{1}(y_u < \tau) + \alpha_v \cdot \mathbb{1}(y_u < \tau, y_v > \gamma)\right] \geq w_v \cdot \int_0^\tau g(x,\gamma)dx$.

Proof. Fix $y_u = x < \tau$. By definition we have $\theta(x) < 1$. The analysis is similar to the previous. We first show that for all $y_v \in [0,1]$, we have $\alpha_u \geq w_v \cdot g(x, \theta(x))$.

We use θ to denote the value that is arbitrarily close to, but larger than $\theta(x)$. By definition, when $y_v = \theta$, u matches some vertex other than v. Thus we have $\alpha_u \geq w_v \cdot g(x, \theta(x))$. Hence, when $y_v > \theta$, i.e. v has a higher price, u would choose the same vertex as when $y_v = \theta$, and $\alpha_u \geq w_v \cdot g(x, \theta(x))$ still holds.

Now consider the case when $y_v = y < \theta$.

As in the analysis of Lemma 6, we compare two processes, when $y_v = \theta$ and when $y_v = y < \theta$. We show that for each vertex $w \in R$ (including u) with $y_w \leq x = y_u$, the utility of w when $y_v = y$ is not worse than its utility when $y_v = \theta$. Suppose otherwise, let w be such a vertex with earliest arrival time.

Let v' be the vertex that is matched to w when $y_v = \theta$. Then we know that (when $y_v = y$) at w's arrival, w chooses a vertex that gives less utility comparing to v'. Hence, at this moment v' is already matched to some w' with $y_{w'} < y_w$. This implies that when $y_v = \theta$, v' (which is matched to w) is unmatched when w' arrives, but not chosen by w'. Therefore, w' has lower utility when $y_v = y$ compared to the case when $y_v = \theta$, which contradicts the assumption that w is the first such vertex.

Observe that when $y_v \in (\gamma, \theta(x))$, we have $\alpha_u + \alpha_v = w_v$. Thus for any fixed $y_u = x < \tau$, we lower bound $\frac{1}{w_v} \cdot \mathbf{E}_{y_v}\left[\alpha_u \cdot \mathbb{1}(y_u < \tau) + \alpha_v \cdot \mathbb{1}(y_u < \tau, y_v > \gamma)\right]$ by

$$f(x, \theta(x)) := g(x, \theta(x)) + \max\{0, \theta(x) - \gamma\} \cdot (1 - g(x, \theta(x))).$$

In the following, we show that $f(x, \theta(x)) \geq g(x, \gamma)$. Consider the following two cases.

1. If $\theta(x) \leq \gamma$, then $f(x, \theta(x)) = g(x, \theta(x)) \geq g(x, \gamma)$, since $\frac{\partial g(x,y)}{\partial y} \leq 0$.
2. If $\theta(x) > \gamma$, then

$$\frac{\partial f(x, \theta(x))}{\partial \theta(x)} = \frac{\partial g(x, \theta(x))}{\partial \theta(x)} + 1 - g(x, \theta(x)) - (\theta(x) - \gamma) \cdot \frac{\partial g(x, \theta(x))}{\partial \theta(x)} \geq 0,$$

where the inequality follows from Claim (2.1) and $\frac{\partial g(x,\theta(x))}{\partial \theta(x)} \leq 0$. Therefore, we have $f(x, \theta(x)) \geq f(x, \gamma) = g(x, \gamma)$.

Finally, take integration over $x \in (0, \tau)$ concludes the lemma. ◀

Proof of Lemma 4. Observe that $\alpha_u + \alpha_v = (\alpha_u + \alpha_v) \cdot \mathbb{1}(y_u > \tau, y_v > \gamma) + \alpha_v \cdot \mathbb{1}(y_v < \gamma) + \alpha_u \cdot \mathbb{1}(y_v < \gamma, y_u > \tau) + \alpha_u \cdot \mathbb{1}(y_u < \tau) + \alpha_v \cdot \mathbb{1}(y_u < \tau, y_v > \gamma)$. Combing Lemma 5, 6 and 7 finishes the proof immediately. ◀

▶ **Theorem 8.** *Fix $h(x) = \min\{1, e^{x-0.5}\}$. For any pair of neighbors u and v, and any fixed ranks of vertices in $L \cup R \setminus \{u, v\}$, we have $\frac{1}{w_v} \cdot \mathbf{E}_{y_u, y_v}\left[\alpha_u + \alpha_v\right] \geq \frac{5}{4} - e^{-0.5} \approx 0.6434$.*

Proof. It suffices to show that the RHS of Lemma 4 is at least $\frac{5}{4} - e^{-0.5}$. Since the expression is symmetric for τ and γ, we assume $\tau \geq \gamma$ without loss of generality.

Let $f(\tau, \gamma)$ be the term on the RHS of Lemma 4 to be minimized. By our choice of g,

$$\begin{aligned} f(\tau,\gamma) =& 1 - \tau - \gamma + \tau\cdot\gamma + \frac{1}{2}\int_0^{\gamma}\big(h(x)+1-h(\tau)\big)dx + \frac{1}{2}\int_0^{\tau}\big(h(x)+1-h(\gamma)\big)dx \\ =& 1 - \frac{\tau}{2}(1+h(\gamma)) - \frac{\gamma}{2}(1+h(\tau)) + \tau\cdot\gamma + \frac{1}{2}\int_0^{\gamma}h(x)dx + \frac{1}{2}\int_0^{\tau}h(x)dx. \end{aligned}$$

Observe that

$$\frac{\partial f(\tau,\gamma)}{\partial \tau} = \gamma - \frac{1}{2}(1+h(\gamma)) - \frac{\gamma}{2}\cdot h'(\tau) + \frac{1}{2}h(\tau).$$

It is easy to check that $\gamma - \frac{1}{2}h(\gamma) \le 0$ when $\gamma \le \frac{1}{2}$; and $\gamma - \frac{1}{2}h(\gamma) > 0$ when $\gamma > \frac{1}{2}$.

Hence when $\gamma \le \frac{1}{2}$, we have $\frac{\partial f(\tau,\gamma)}{\partial \tau} \le 0$, which means that the minimum is attained when $\tau = 1$. Note that when $\gamma \le \frac{1}{2}$, we have

$$f(1,\gamma) = \frac{1}{2}(1-h(\gamma)) + \frac{1}{2}\int_0^{\gamma}h(x)dx + \frac{1}{2}\int_0^{1}h(x)dx,$$

which attains its minimum at $\gamma = 0$ (since $h'(\gamma) = h(\gamma)$ for $\gamma \le \frac{1}{2}$):

$$f(1,0) = \frac{1}{2}(1-e^{-0.5}) + \frac{1}{2}(\frac{1}{2}+1-e^{-0.5}) = \frac{5}{4} - e^{-0.5} \approx 0.6434.$$

When $\tau \ge \gamma > \frac{1}{2}$, we have $\frac{\partial f(\tau,\gamma)}{\partial \tau} = \gamma - \frac{1}{2}h(\gamma) > 0$. Hence the minimum is attained when $\tau = \gamma$, which is $f(\gamma,\gamma) = 1 - 2\gamma + \gamma^2 + \int_0^{\gamma}h(x)dx$. Observe that

$$\frac{df(\gamma,\gamma)}{d\gamma} = -2 + 2\gamma + h(\gamma) \ge -2 + 1 + 1 = 0.$$

The minimum is attained when $\gamma = \frac{1}{2}$, which equals $f(\frac{1}{2},\frac{1}{2}) = \frac{5}{4} - e^{-0.5} \approx 0.6434$. ◀

4 Improving the Competitive Ratio

Observe that in Lemma 4, we relax the total gain of $\alpha_u + \alpha_v$ into two parts: (1) when $y_u \ge \tau$ and $y_v \ge \gamma$, $\alpha_u + \alpha_v = w_v$. (2) for other ranks y_u, y_v, we lower bound α_u and α_v by $w_v \cdot g(y_u,\gamma)$ and $w_v \cdot g(y_v,\tau)$ respectively. For the second part, the inequalities used in the proof of Lemma 6 and 7 are tight only if β, θ are two step functions (refer to Figure 1). On the other hand, given these β, θ, when $y_u \le \tau$ and $y_v \le \gamma$, we actually have $\alpha_u + \alpha_v = w_v$, which is strictly larger than our estimation $w_v \cdot (g(y_u,\gamma) + g(y_v,\tau))$.

With this observation, it is natural to expect an improved bound if we can retrieve this part of gain (even partially). In this section, we prove an improved competitive ratio 0.6534, using a refined lower bound for $\frac{1}{w_v}\cdot\mathbf{E}[\alpha_u+\alpha_v]$ (compared to Lemma 4) as follows.

▶ **Lemma 9** (Improved Bound). *For any pair of neighbors $u \in R$ and $v \in L$, we have*

$$\begin{aligned} \frac{1}{w_v}\cdot\mathbf{E}[\alpha_u+\alpha_v] \ge \min_{0\le\gamma,\tau\le 1}\Bigg\{&(1-\tau)(1-\gamma) + (1-\tau)\int_0^{\gamma}g(x,\tau)dx \\ &+ \int_0^{\tau}\min_{\theta\le\gamma}\Big\{g(x,\theta) + \int_0^{\theta}g(y,x)dy + \int_{\theta}^{\gamma}g(y,\tau)dy\Big\}dx\Bigg\}. \end{aligned}$$

Proof. Let γ and τ be defined as before, i.e., $\gamma = \beta(1)$ and $\tau = \min\{x : \theta(x) = 1\}$.

We divide $\frac{1}{w_v} \cdot \mathbf{E}\left[\alpha_u + \alpha_v\right]$ into three parts, namely (1) when $y_u > \tau$ and $y_v > \gamma$; (2) when $y_u > \tau$ and $y_v < \gamma$; and (3) when $y_u < \tau$:

$$\begin{aligned}\frac{1}{w_v} \cdot \mathbf{E}\left[\alpha_u + \alpha_v\right] = & \frac{1}{w_v} \cdot \mathbf{E}\left[(\alpha_u + \alpha_v) \cdot \mathbb{1}(y_u > \tau, y_v > \gamma)\right] \\ & + \frac{1}{w_v} \cdot \mathbf{E}\left[(\alpha_u + \alpha_v) \cdot \mathbb{1}(y_u > \tau, y_v < \gamma)\right] \\ & + \frac{1}{w_v} \cdot \mathbf{E}\left[(\alpha_u + \alpha_v) \cdot \mathbb{1}(y_u < \tau)\right].\end{aligned}$$

As shown in Lemma 5, the first term is at least $(1-\tau)\cdot(1-\gamma)$, as we have $\alpha_u + \alpha_v = w_v$ for all $y_u > \tau$ and $y_v > \gamma$. Then we consider the second term, the expected gain of $\alpha_u + \alpha_v$ when $y_v < \gamma$ and $y_u > \tau$. For any $y_v < \gamma$, as we have shown in Lemma 6, $\alpha_v \geq w_v \cdot g(y_v, \beta^{-1}(y_v))$ for all $y_u > \tau$. Moreover, when $y_u < \beta^{-1}(y_v)$, we have $\alpha_u + \alpha_v = w_v$. Hence the second term can be lower bounded by

$$\int_0^\gamma \Big((1-\tau) \cdot g(y_v, \beta^{-1}(y_v)) + \max\{0, \beta^{-1}(y_v) - \tau\} \cdot \big(1 - g(y_v, \beta^{-1}(y_v))\big)\Big) dy_v.$$

Now we consider the last term and fix a $y_u < \tau$.

As we have shown in Lemma 7, for all $y_v \in [0,1]$, $\alpha_u \geq w_v \cdot g(y_u, \theta(y_u))$.

Consider the case when $\theta(y_u) > \gamma$, then for $y_v \in (0, \gamma)$, $\alpha_v \geq w_v \cdot g(y_v, y_u)$; for $y_v \in (\gamma, \theta(y_u))$, $\alpha_u + \alpha_v = w_v$. Thus the expected gain of $\alpha_u + \alpha_v$ (taken over the randomness of y_v) can be lower bounded by

$$w_v \cdot \Big(g(y_u, \theta(y_u)) + \int_0^\gamma g(y_v, y_u) dy_v + (\theta(y_u) - \gamma) \cdot (1 - g(y_u, \theta(y_u)))\Big).$$

As we have shown in Lemma 7, the partial derivative over $\theta(y_u)$ is non-negative, thus for the purpose of lower bounding $\frac{1}{w_v} \cdot \mathbf{E}\left[\alpha_u + \alpha_v\right]$, we can assume that $\theta(y_u) \leq \gamma$ for all $y_u < \tau$.

Given that $\theta(y_u) \leq \gamma$, we have $\alpha_v \geq w_v \cdot g(y_v, y_u)$ when $y_v \in (0, , \theta(y_u))$; and $\alpha_v \geq w_v \cdot g(y_v, \beta^{-1}(y_v))$ when $y_v \in (\theta(y_u), \gamma)$.

Hence the third term can be lower bounded by

$$\int_0^\tau \Big(g(y_u, \theta(y_u)) + \int_0^{\theta(y_u)} g(y_v, y_u) dy_v + \int_{\theta(y_u)}^\gamma g(y_v, \beta^{-1}(y_v)) dy_v\Big) dy_u$$

Putting the three lower bounds together and taking the partial derivative over $\beta^{-1}(y_v)$, for those $\beta^{-1}(y_v) > \tau$, we have a non-negative derivative as follows:

$$\frac{\partial g(y_v, \beta^{-1}(y_v))}{\partial \beta^{-1}(y_v)} + 1 - g(y_v, \beta^{-1}(y_v)) - (\beta^{-1}(y_v) - \tau) \cdot \frac{\partial g(y_v, \beta^{-1}(y_v))}{\partial \beta^{-1}(y_v)} \geq 0.$$

Thus for lower bounding $\frac{1}{w_v} \cdot \mathbf{E}\left[\alpha_u + \alpha_v\right]$, we assume $\beta^{-1}(y_v) \leq \tau$ for all $y_v < \gamma$. Hence

$$\begin{aligned}\frac{1}{w_v} \cdot \mathbf{E}\left[\alpha_u + \alpha_v\right] \geq \min_{0 \leq \gamma, \tau \leq 1} \Big\{ & (1-\tau)(1-\gamma) + (1-\tau) \int_0^\gamma g(y_v, \tau) dy_v \\ & + \int_0^\tau \Big(g(y_u, \theta(y_u)) + \int_0^{\theta(y_u)} g(y_v, y_u) dy_v + \int_{\theta(y_u)}^\gamma g(y_v, \tau) dy_v\Big) dy_u \Big\}.\end{aligned}$$

Taking the minimum over $\theta(y_u)$ concludes Lemma 9. ◀

Observe that for any $\theta \le \gamma$, we have

$$g(x,\theta) + \int_0^\theta g(y,x)dy + \int_\theta^\gamma g(y,\tau)dy \ge g(x,\gamma) + \int_0^\gamma g(y,\tau)dy.$$

Thus the lower bound given by Lemma 9 is not worse than Lemma 4.

▶ **Theorem 10.** *Fix $h(x) = \min\{1, \frac{1}{2}e^x\}$. For any pair of neighbors u and v, and any fixed ranks of vertices in $L \cup R \setminus \{u, v\}$, we have $\frac{1}{w_v} \cdot \mathbf{E}_{y_u, y_v}[\alpha_u + \alpha_v] \ge 1 - \frac{\ln 2}{2} \approx 0.6534$.*

We give a proof sketch and defer the complete analysis to the full version of the paper.

Proof Sketch. For $h(x) = \min\{1, \frac{1}{2}e^x\}$, we have $h'(x) = h(x)$ when $x < \ln(2)$, and $h'(x) = 0$, $h(x) = 1$ when $x > \ln(2)$.

Let $f(\tau, \gamma)$ be the expression on the RHS to be minimized in Lemma 9. We first show that for any fixed τ and γ, the minimum (over θ) of $f(\tau, \gamma)$ is obtained when $\theta = \min\{\ln 2, \gamma\}$. Hence we can lower bound $f(\tau, \gamma)$ by

$$(1-\tau)(1-\gamma) + \frac{\gamma}{2}(1-h(\tau)) + \frac{\tau}{2}(1-h(\gamma)) + \frac{\ln 2}{2}\tau \cdot h(\tau) + \frac{1}{2}\int_0^\gamma h(y)dy + \frac{1-\ln 2}{2}\int_0^\tau h(x)dx.$$

Then we show that $f(\tau, \gamma) \ge 1 - \frac{\ln 2}{2} \approx 0.6534$ for all $\tau, \gamma \in [0, 1]$. We show that there exists $\tau^* \approx 0.3574$ (solution for $1 + h(\tau) - 2\tau = 1$) such that for $\tau \le \tau^*$, $\frac{\partial f(\tau,\gamma)}{\partial \gamma} \le 0$. Thus $f(\tau, \gamma) \ge f(\tau, 1)$. Further more, we show that $\frac{\partial f(\tau,1)}{\partial \tau} \ge 0$, which implies

$$f(\tau,\gamma) \ge f(\tau,1) \ge f(0,1) = \frac{1}{2}(1 - h(0)) + \frac{1}{2}\int_0^1 h(y)dy = 1 - \frac{\ln 2}{2} \approx 0.6534.$$

For any fixed $\tau > \tau^*$, we show that the minimum (over γ) of $f(\tau, \gamma)$ is attained when $\gamma = \ln 2$. Hence for $\tau > \tau^*$ we have $f(\tau, \gamma) \ge f(\tau, \ln 2)$. Finally, we show that $\frac{\partial f(\tau,\ln 2)}{\partial \tau} < 0$ when $\tau < \tau_0$; and $\frac{\partial f(\tau,\ln 2)}{\partial \tau} > 0$ when $\tau > \tau_0$, where $\tau_0 \approx 0.564375$, which implies

$$\begin{aligned} f(\tau,\gamma) \ge f(\tau,\ln 2) \ge f(\tau_0,\ln 2) =& (1-\tau_0)(1-\ln 2) + \frac{\ln 2}{4} \cdot (2 - e^{\tau_0} + \tau_0 \cdot e^{\tau_0}) + \frac{1}{4} \\ &+ \frac{1-\ln 2}{4}(e^{\tau_0} - 1) \approx 0.6557 > 1 - \frac{\ln 2}{2}. \end{aligned}$$

Thus for all $\tau, \gamma \in [0, 1]$, we have $f(\tau, \gamma) \ge 1 - \frac{\ln 2}{2}$, as claimed.

5 Conclusion

In this paper, we show that competitive ratios above $1 - \frac{1}{e}$ can be obtained under the randomized primal-dual framework when equipped with a two dimensional gain sharing function. The key of the analysis is to lower bound the expected combined gain of every pair of neighbors (u, v), over the randomness of the rank y_v of the offline vertex, and the arrival time y_u of the online vertex.

Referring to Figure 1, it can be shown that the competitive ratio $F \ge \int_0^1 f(y_u)dy_u$, where

$$\begin{aligned} f(y_u) :=& (1 - \theta(y_u) + \beta(y_u)) \cdot g(y_u, \theta(y_u)) + \theta(y_u) - \beta(y_u) \\ &+ \int_0^{\beta(y_u)} g(y_v, \beta^{-1}(y_v))dy_v + \int_{\theta(y_u)}^1 g(y_v, \beta^{-1}(y_v))dy_v. \end{aligned}$$

Note that here we assume $\beta^{-1}(y_v) = 1$ for all $y_v \geq \gamma$, and $g(x, 1) = 0$ for all $x \in [0, 1]$.

For every fixed g, there exist threshold functions θ and β that minimize the integration. Thus the main difficulty is to find a function g such that the integration has a large lower bound for all functions θ and β (which depend on the input instance). We have shown that there exists a choice of g such that the minimum is attained when θ and β are step functions, based on which we can give a lower bound on the competitive ratio.

It is thus an interesting open problem to know how much the competitive ratio can be improved by (fixing an appropriate function g and) giving a tighter lower bound for the integration. We believe that it is possible to give a lower bound very close to (or even better than) the 0.696 competitive ratio obtained for the unweighted case [21].

References

1. Melika Abolhassani, T.-H. Hubert Chan, Fei Chen, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Hamid Mahini, and Xiaowei Wu. Beating ratio 0.5 for weighted oblivious matching problems. In *ESA*, volume 57 of *LIPIcs*, pages 3:1–3:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
2. Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1253–1264, 2011. doi:10.1137/1.9781611973082.95.
3. Jonathan Aronson, Martin Dyer, Alan Frieze, and Stephen Suen. Randomized greedy matching. ii. *Random Struct. Algorithms*, 6(1):55–73, 1995. doi:10.1002/rsa.3240060107.
4. Bahman Bahmani and Michael Kapralov. Improved bounds for online stochastic matching. In *ESA (1)*, volume 6346 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2010.
5. Benjamin Birnbaum and Claire Mathieu. On-line bipartite matching made simple. *ACM SIGACT News*, 39(1):80–87, 2008.
6. Brian Brubach, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. New algorithms, better bounds, and a novel model for online stochastic matching. In *ESA*, volume 57 of *LIPIcs*, pages 24:1–24:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
7. Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *ESA*, volume 4698 of *Lecture Notes in Computer Science*, pages 253–264. Springer, 2007.
8. Niv Buchbinder, Kamal Jain, and Mohit Singh. Secretary problems via linear programming. *Math. Oper. Res.*, 39(1):190–206, 2014.
9. T.-H. Hubert Chan, Fei Chen, Xiaowei Wu, and Zhichao Zhao. Ranking on arbitrary graphs: Rematch via continuous lp with monotone and boundary condition constraints. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1112–1122. SIAM, 2014. doi:10.1137/1.9781611973402.82.
10. Ilan Reuven Cohen and David Wajc. Randomized online matching in regular graphs. In *SODA*, pages 960–979. SIAM, 2018.
11. Nikhil R. Devanur and Kamal Jain. Online matching with concave returns. In *STOC*, pages 137–144. ACM, 2012.
12. Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In *SODA*, pages 101–107. SIAM, 2013.

13 Jon Feldman, Aranyak Mehta, Vahab S. Mirrokni, and S. Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *FOCS*, pages 117–126. IEEE Computer Society, 2009.

14 Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *SODA*, pages 982–991, 2008. URL: `http://dl.acm.org/citation.cfm?id=1347082.1347189`.

15 Bernhard Haeupler, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Online stochastic weighted matching: Improved approximation algorithms. In *WINE*, volume 7090 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2011.

16 Zhiyi Huang, Ning Kang, Zhihao Gavin Tang, Xiaowei Wu, Yuhao Zhang, and Xue Zhu. How to match when all vertices arrive online. *CoRR (to appear in STOC 2018)*, abs/1802.03905, 2018. `arXiv:1802.03905`.

17 Patrick Jaillet and Xin Lu. Online stochastic matching: New algorithms with better bounds. *Math. Oper. Res.*, 39(3):624–646, 2014.

18 Chinmay Karande, Aranyak Mehta, and Pushkar Tripathi. Online bipartite matching with unknown distributions. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 587–596, 2011. `doi:10.1145/1993636.1993715`.

19 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 352–358, 1990. `doi:10.1145/100216.100262`.

20 Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In *ESA*, volume 8125 of *Lecture Notes in Computer Science*, pages 589–600. Springer, 2013.

21 Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 597–606, 2011. `doi:10.1145/1993636.1993716`.

22 Vahideh H. Manshadi, Shayan Oveis Gharan, and Amin Saberi. Online stochastic matching: Online actions based on offline statistics. *Math. Oper. Res.*, 37(4):559–573, 2012.

23 Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22, 2007.

24 Yajun Wang and Sam Chiu-wai Wong. Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm. In *ICALP (1)*, volume 9134 of *Lecture Notes in Computer Science*, pages 1070–1081. Springer, 2015.

Finding Branch-Decompositions of Matroids, Hypergraphs, and More

Jisu Jeong[1]
Department of Mathematical Sciences, KAIST, Daejeon, Korea
jisujeong89@gmail.com

Eun Jung Kim[2]
Université Paris-Dauphine, PSL Research University, CNRS, Paris, France
eun-jung.kim@dauphine.fr

Sang-il Oum[3]
Department of Mathematical Sciences, KAIST, Daejeon, Korea
sangil@kaist.edu

Abstract

Given n subspaces of a finite-dimensional vector space over a fixed finite field $\mathbb{F}$, we wish to find a "branch-decomposition" of these subspaces of width at most k, that is a subcubic tree T with n leaves mapped bijectively to the subspaces such that for every edge e of T, the sum of subspaces associated to the leaves in one component of $T - e$ and the sum of subspaces associated to the leaves in the other component have the intersection of dimension at most k. This problem includes the problems of computing branch-width of $\mathbb{F}$-represented matroids, rank-width of graphs, branch-width of hypergraphs, and carving-width of graphs.

We present a fixed-parameter algorithm to construct such a branch-decomposition of width at most k, if it exists, for input subspaces of a finite-dimensional vector space over $\mathbb{F}$. Our algorithm is analogous to the algorithm of Bodlaender and Kloks (1996) on tree-width of graphs. To extend their framework to branch-decompositions of vector spaces, we developed highly generic tools for branch-decompositions on vector spaces. For this problem, a fixed-parameter algorithm was known due to Hliněný and Oum (2008). But their method is highly indirect. Their algorithm uses the non-trivial fact by Geelen et al. (2003) that the number of forbidden minors is finite and uses the algorithm of Hliněný (2006) on checking monadic second-order formulas on $\mathbb{F}$-represented matroids of small branch-width. Our result does not depend on such a fact and is completely self-contained, and yet matches their asymptotic running time for each fixed k.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability, Mathematics of computing → Graph algorithms

Keywords and phrases branch-width, rank-width, carving-width, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.80

Related Version A full version of the paper is available at `https://arxiv.org/abs/1711.01381`.

[1] Supported by the National Research Foundation of Korea (NRF) grant (No. NRF-2017R1A2B4005020).
[2] Supported by the National Research Agency (ANR) grant (No. ANR-17-CE40-0028).
[3] Supported by the National Research Foundation of Korea (NRF) grant (No. NRF-2017R1A2B4005020).

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 80; pp. 80:1–80:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Let $\mathbb{F}$ be a finite field and r be a positive integer. A *subspace arrangement* $\mathcal{V}$ is a (multi)set of subspaces of $\mathbb{F}^r$, which can be represented by an $r \times m$ matrix M with an ordered partition $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of $\{1, 2, \ldots, m\}$ such that for every $1 \leq i \leq n$, the i-th element of $\mathcal{V}$ is the column space of the submatrix of M induced by the columns in I_i.[4] Here, an *ordered partition* $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of $\{1, 2, \ldots, m\}$ is a partition of $\{1, 2, \ldots, m\}$ such that $x < y$ for all $x \in I_i, y \in I_j$ with $i < j$.

Robertson and Seymour [12] introduced the notion of branch-width for graphs, hypergraphs, and more generally, for connectivity functions. We are going to define the branch-width of a subspace arrangement as follows. First, a tree is *subcubic* if every node has degree at most 3. We define a *leaf* of a tree as a node of degree at most 1. A *branch-decomposition* of $\mathcal{V}$ is a pair $(T, \mathcal{L})$ of a subcubic tree T with no degree-2 nodes and a bijective function $\mathcal{L}$ from the set of all leaves of T to $\mathcal{V}$. For a node v of T and an edge e incident with v, let us write $A_v(T - e)$ to denote the set of all leaves of T in the component of $T - e$ containing v. For a branch-decomposition $(T, \mathcal{L})$ of $\mathcal{V}$ and each edge $e = uv$ of T, we define the *width* of e to be $\dim\left(\sum_{x \in A_u(T-e)} \mathcal{L}(x)\right) \cap \left(\sum_{y \in A_v(T-e)} \mathcal{L}(y)\right)$. The *width* of $(T, \mathcal{L})$ is the maximum width of all edges of T. (If T has no edges, then the width of $(T, \mathcal{L})$ is 0.) The *branch-width* of $\mathcal{V}$ is the minimum k such that there exists a branch-decomposition of $\mathcal{V}$ having width at most k.

We aim to solve the following problem, and Theorem 1.1 is our main theorem.

BRANCH-WIDTH
Parameters: A finite field $\mathbb{F}$ and an integer k.
Input: An $r \times m$ matrix M over $\mathbb{F}$ with an ordered partition $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of $\{1, 2, \ldots, m\}$ and an integer k.
Output: A branch-decomposition $(T, \mathcal{L})$ of width at most k of a subspace arrangement $\mathcal{V}$ consisting of the column space of the submatrix of M induced by the columns in I_i for each i or a confirmation that the branch-width of $\mathcal{V}$ is larger than k.

▶ **Theorem 1.1.** *Let $\mathbb{F}$ be a finite field, let r be a positive integer, and let k be a nonnegative integer. Let $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$ be a subspace arrangement of subspaces of $\mathbb{F}^r$ where each V_i is given by its spanning set of d_i vectors and $m = \sum_{i=1}^{n} d_i$. In time $O(rm^2 + (k+1)rmn + k^3n^3 + f(|\mathbb{F}|, k)n^2)$ for some function f, one can either find a branch-decomposition of $\mathcal{V}$ having width at most k or confirm that no such branch-decomposition exists.*

Various width parameters of discrete structures have been introduced and used for algorithmic and structural applications. One popular way of creating a width parameter of a discrete structure is to define it as the branch-width of some connectivity function defined on that discrete structure. Theorem 1.1 immediately gives rise to analogous algorithms for many of them, such as carving-width of graphs, rank-width of graphs, and branch-width of graphs, hypergraphs, and matroids. We will give a brief overview of each application.

Branch-width of matroids represented over a finite field $\mathbb{F}$. Let $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$ be a subspace arrangement of subspaces of $\mathbb{F}^r$. If each V_i is the span of a vector v_i in $\mathbb{F}^r$ for each $i = 1, 2, \ldots, n$, then $\mathcal{V}$ can be identified with the matroid M represented by the vectors v_1, v_2, …, v_n. Furthermore, branch-width and branch-decompositions of M are precisely branch-width and branch-decompositions of $\mathcal{V}$, respectively.

[4] Subspace arrangements can be regarded as representable *partitioned matroids* used in [6]. A partitioned matroid is a matroid equipped with a partition of its ground set.

Rank-width of graphs. Rank-width, introduced by Oum and Seymour [10], is a width parameter of graphs expressing how easy it is to decompose a graph into a tree-like structure, called a *rank-decomposition*, while keeping every edge cut to have a small 'complexity', called the *width* of a rank-decomposition, where the complexity is measured by the matrix rank function. Each vertex of a graph G can be associated with a subspace of dimension at most 2 so that the subspace arrangement $\mathcal{V}$ consisting of all subspaces associated with the vertices of G has branch-width $2k$ if and only if G has rank-width k (See appendix). Furthermore, a branch-decomposition of $\mathcal{V}$ of width $2k$ corresponds to a rank-decomposition of G of width k.

Branch-width of hypergraphs. Robertson and Seymour defined the notion of a branch-width [12] not only for graphs but also for hypergraphs. Let $\mathbb{F} = GF(2)$ be the binary field and let $\{v_1, v_2, \ldots, v_n\}$ be the standard basis of $\mathbb{F}^n$. For a hypergraph G with n vertices $v_1, v_2, \ldots, v_n$, we associate each edge e with the span of the vertices incident with e. Let $\mathcal{V}$ be the subspace arrangement consisting of all subspaces associated with the edges of G. Then it is not difficult to show that branch-width and branch-decomposition of G are precisely branch-width and branch-decomposition of $\mathcal{V}$, respectively.

Carving-width of graphs. Seymour and Thomas [13] introduced carving-width of graphs. Let $\mathbb{F} = GF(2)$ be the binary field and let $\{e_1, e_2, \ldots, e_m\}$ be the standard basis of $\mathbb{F}^m$. For a graph G with edges $e_1, e_2, \ldots, e_m$, we associate each vertex v with the span of the edges incident with v. If $\mathcal{V}$ is the subspace arrangement consisting of all subspaces associated with the vertices of G, then carving-width and carving of G are precisely branch-width and branch-decomposition of $\mathcal{V}$, respectively.

For the first two applications, the analogous theorems were proved earlier by Hliněný and Oum [6]. However, their approach was completely indirect; they use a non-trivial fact shown by Geelen et al. [3] that the class of matroids of branch-width at most k has finitely many forbidden minors, each having at most $O(6^k)$ elements. Then they use a monadic second-order formula to describe whether a matroid contains a fixed minor and use the dynamic programming algorithm to decide a monadic second-order formula aided by a given branch-decomposition of bounded width. So far this describes the decision algorithm of Hliněný [5] that decides whether branch-width is at most k. On top of this algorithm, Hliněný and Oum use a sophisticated reduction to modify the input and use the decision algorithm repeatedly to recover a branch-decomposition. Roughly speaking, this reduction attaches a gadget to the input matroid and this step requires extending the underlying finite field to an extension field, because this gadget is not representable if the underlying field is too small. As the list of forbidden minors is unknown, their algorithm should generate the list of minor-minimal matroids having branch-width larger than k. Thus, even for small values of k, it would be practically impossible to implement their algorithm. Contrary to the previous algorithm, our algorithm does not depend on the finiteness of obstructions and yet matches their asymptotic running time for each fixed k.

We do not know any previous analogous theorems for branch-width of hypergraphs. For branch-width of graphs, Thilikos and Bodlaender [14] posted a 50-page long technical report in 2000 proving that for every fixed k, one can check in linear time whether a graph has branch-width at most k and if so, output a branch-decomposition of minimum width. This work was presented at a conference in 1997 [2]. For carving-width of graphs, the conference paper of Thilikos, Serna, and Bodlaender [15] presented a linear-time algorithm for each fixed k that determines whether the carving-width of an input graph G is at most k, and if so, constructs a carving of G with minimum carving-width.

Our approach and technical ingredients. We develop a framework inspired by the approach of Bodlaender and Kloks [1] on their work on tree-width of graphs. (A similar framework was also given independently by Lagergren and Arnborg [9].) They created a linear-time algorithm that can find a tree-decomposition of width at most k or confirm that the tree-width of an input graph is larger than k for each fixed k. They used dynamic programming based on a given tree-decomposition of bounded width. For the dynamic programming, they designed a special encoding of all possible tree-decompositions of width at most k that can arise from certain parts of a graph.

We also use dynamic programming approach, taking advantage of having a tree-like structure from the given branch-decomposition. Then, how do we generate a branch-decomposition of small width in the first place? For this purpose, we use the technique called the iterative compression, which is initiated by Reed, Smith, and Vetta [11] and used by [4] for computing the branch-width of linear matroid. In Branch-Width Compression, we are given a branch-decomposition of width at most $2k$ of a subset of $\mathcal{V}$ and solve Branch-width. The obtained branch-decomposition of width at most k at each step is incremented by a new element of $\mathcal{V}$, which serves as a given branch-decomposition of width at most $2k$ for the next step.

To use a branch-decomposition for dynamic programming, we need a concept of a 'boundary', that plays the role of a bag in a tree-decomposition. For a branch-decomposition $(T, \mathcal{L})$ of a subspace arrangement $\mathcal{V}$ and an edge e of T, we consider the boundary B as the intersection of the sum of subspaces associated to the leaves in one component of $T - e$ and the sum of subspaces associated to the leaves of the other component of $T - e$. As the branch-width is at most k, the boundary B has dimension at most k. Furthermore, we restrict our attention to the finite field $\mathbb{F}$ and so the number of subspaces of B is finite. For the convenience of dynamic programming on branch-decompositions, we define *transcripts* of a branch-decomposition in Section 2, which is essentially a precomputed list of bases and linear transformations useful for computations with boundaries.

As usual, we need a compact encoding scheme to store partial solutions that may be extended to a branch-decomposition of width at most k, if it exists. We have two important aspects here.

First of all, we will restrict our search to a smaller set of branch-decompositions. Namely, if $(T', \mathcal{L}')$ is a given branch-decomposition of width at most $2k$ in Branch-Width Compression, then the algorithm will find a branch-decomposition of width at most k that is *totally pure with respect to* $(T', \mathcal{L}')$. In order to efficiently compress the partial solutions at each step of dynamic programming, it is crucial to ensure that some part of a partial solution can be forgotten (and can be retrieved from the unforgotten part later). A part of a partial solution can be ignored only when there is a guarantee that the said part does not need to be 'mixed' with another partial solution in the future. In other words, if there is a branch-decomposition of width at most k which is obtained via mixing, then there also exists as good a decomposition which can be obtained while mixing is avoided. This general idea lies at the core of every dynamic programming based on decomposition of small width. The first technical barrier to implement this principle for our problem is how to formalize what constitute those forgettable parts and what it means to avoid mixing. The 'forgettable part' is formalized as the notion of 'blocked' (plus some more) nodes introduced in Section 5. Intuitively, a totally pure branch-decomposition with respect to $(T', \mathcal{L}')$ is a decomposition which has successfully avoided the 'mixing' at every descendant so far. The two operations introduced in Section 3, fork and split, are technical tools developed in order to prove that it is possible to avoid mixing. Using these operations, we show that if there is a branch-decomposition of width at most k, then there is a branch-decomposition of width at

most k that is totally pure with respect to $(T', \mathcal{L}')$ in Section 3. The procedure of obtaining a compact encoding of a partial solution, introduced as *trimming* in Section 4, is essentially discarding the forgettable parts.

The second technical barrier is to devise an encoding of a (partial) branch-decomposition and all computational operations, necessary for dynamic programming, compatible with this encoding scheme. For this purpose, we will define a B-namu[5] in Section 4. Here, B is going to be the boundary for some edge in $(T', \mathcal{L}')$. A B-namu is, roughly speaking, a subcubic tree whose incidences are decorated by subspaces of B and whose edges are labeled by a nonnegative integer so that it represents the 'shadow' of a branch-decomposition of width at most k on B. We define an operation τ on B-namus that compresses a B-namu into a 'compact' B-namu and prove that there are only finitely many compact B-namus of width at most k, when B has bounded dimension and $\mathbb{F}$ is a finite field. This operation τ on B-namus consists of two steps: one is the aforementioned trimming, another is *compressing* to compress an integer sequence introduced by Bodlaender and Kloks [1] for their work on tree-decompositions. A part of B-namu processed by compressing step can be potentially mixed with another partial solution in the future, but a desired decomposition can be always retrieved if one exists. In contrast, a trimmed part is forgotten and never gets mixed in the future. Our computational operations on B-namus are designed so that the trimmed parts can be efficiently retrieved. Computational operations on B-namus for dynamic programming are defined including comparison.

Difference with our previous work. In the previous work [8], the authors found a similar algorithm for path-decompositions of a subspace arrangement. A path-decomposition of a subspace arrangement is a linearized variant of a branch-decomposition that restricts the subcubic trees to caterpillar trees. Here are the key technical differences.

First, the concept of totally pure branch-decompositions was not needed in [8]. In the previous work, two linear orderings are merged into another linear ordering and there is no need to consider the possibility of 'avoiding mixing'. In the end, compressing an integer sequence was sufficient to obtain a short encoding. For branch-decompositions, we sometimes insert a whole subtree into a branch-decomposition and this requests a new concept such as totally pure branch-decompositions. Also, 'summing' two partial solution encodings for join operation in dynamic programming is much more delicate in this work.

Second, we needed the concept of k-safeness in order to extend our algorithm for path-decompositions to the algorithm for branch-decompositions. When we sum two B-namus, some edges of trees in the B-namus are in common but some edges are not shared and will be forgotten. Although an edge in one B-namu is not in another B-namu, the width assigned to the edge can potentially increase. We hope the width of an edge not to exceed k even when this edge is 'forgotten', and thus we need to handle this carefully.

Lastly, we improve the running time of an algorithm computing transcripts. In [8], the idea of transcripts was used as well although the notion was not formally introduced. If we adapt our new method to the result of [8], we also get an $O(n^3)$-time algorithm for path-decompositions of subspace arrangements based on iterative compressions, saving a factor of $O(n)$.

Section 2, Section 3, and Section 4 give both definitions and some properties of transcripts, totally pure branch-decompositions, and B-namus, respectively. Section 5 presents the algorithm to solve the Branch-Width problem. We remark that many proofs are omitted because of the page limit. However, the detailed proofs are contained in the full version.

[5] 'Namu' is a tree in Korean.

2 Transcripts

Dynamic programming algorithms on tree-decomposition benefit from the small width by encoding solutions with respect to the bags. While the bags are explicit in a given tree-decomposition, a branch-decomposition of a subspace arrangement does not provide an easy-to-handle metric for encoding solutions to our problem. In order to make it more useful, we need some extra information.

Let $(T, \mathcal{L})$ be a branch-decomposition of a subspace arrangement $\mathcal{V}$. We will assume that T is a rooted binary tree by picking an arbitrary edge e and subdividing e to create a degree-2 root node r, and call $(T, \mathcal{L})$ a *rooted* branch-decomposition. For a node v of T, let $\mathcal{V}_v$ be the set of all elements of $\mathcal{V}$ associated with v and its descendants by $\mathcal{L}$. For a set X of vectors from a vector space over a field $\mathbb{F}$, the *span* $\langle X \rangle$ of X is the subspace consisting of all (finite) linear combinations of vectors in X, where the scalars are taken from $\mathbb{F}$. For two subspaces X and Y, we denote the subspace $\{x+y : x \in X, y \in Y\}$ by $X+Y$. For a set $\mathcal{X}$ of subspaces, let $\langle \mathcal{X} \rangle = \sum_{X \in \mathcal{X}} X$. The *boundary space* B_v at v is defined as $B_v = \langle \mathcal{V}_v \rangle \cap \langle \mathcal{V} - \mathcal{V}_v \rangle$. Later, we shall encode partial branch-decompositions with respect to the boundary spaces of a given branch-decomposition $(T, \mathcal{L})$. For this, we need to know B_v in advance.

A *transcript* of $(T, \mathcal{L})$ is a pair $\Lambda = (\{\mathfrak{B}_v\}_{v \in V(T)}, \{\mathfrak{B}'_v\}_{v \in V(T)})$ of sets of ordered bases $\mathfrak{B}_v$ and $\mathfrak{B}'_v$ of subspaces $B_v = \langle \mathfrak{B}_v \rangle$ and $B'_v = \langle \mathfrak{B}'_v \rangle$ of $\mathbb{F}^r$, respectively, such that

- the first $|\mathfrak{B}_v|$ elements of $\mathfrak{B}'_v$ are precisely $\mathfrak{B}_v$ for each node v,
- $\langle \mathfrak{B}'_v \rangle = \langle \mathfrak{B}_{w_1} \rangle + \langle \mathfrak{B}_{w_2} \rangle$ for each node v having two children w_1 and w_2,
- $\langle \mathfrak{B}'_v \rangle = \langle \mathfrak{B}_v \rangle$ for each leaf v.

If a node u of T is a parent of a node v of T, then $B_v = \langle \mathfrak{B}_v \rangle$ is a subspace of $B'_u = \langle \mathfrak{B}'_u \rangle$ and therefore there exists the unique $|\mathfrak{B}'_u| \times |\mathfrak{B}_v|$ matrix T_v over $\mathbb{F}$ such that $T_v[x]_{\mathfrak{B}_v} = [x]_{\mathfrak{B}'_u}$ for all $x \in \mathfrak{B}_v$. This matrix T_v is called the *transition matrix* of Λ at a node v. (For the root node r, let T_r be the null matrix. For a vector x in a vector space with a basis $\mathfrak{B}$ over a field $\mathbb{F}$, $[x]_{\mathfrak{B}}$ denotes the coordinate vector with respect to the basis $\mathfrak{B}$, which is a $|\mathfrak{B}| \times 1$ matrix over $\mathbb{F}$.)

We can compute the transcript of a given branch-decomposition as follows.

▶ **Theorem 2.1.** *Let $\mathcal{V}$ be a subspace arrangement of $\mathbb{F}^r$ represented by an $r \times m$ matrix M in reduced row echelon form with no zero rows such that each $V \in \mathcal{V}$ has dimension at most k. Let $n = |\mathcal{V}|$. Given branch-decomposition $(T, \mathcal{L})$ of $\mathcal{V}$, in time $O(k^3 n^2)$, one can correctly compute a basis of $\langle \mathcal{V}_v \rangle \cap \langle \mathcal{V} - \mathcal{V}_v \rangle$ for all nodes v of T or confirm that $(T, \mathcal{L})$ has width larger than k. In addition, if $(T, \mathcal{L})$ has width at most k, then we can compute the transcript $\Lambda = (\{\mathfrak{B}_v\}, \{\mathfrak{B}'_v\})$ of $(T, \mathcal{L})$ with its transition matrices in time $O(k^3 n^2)$.*

3 Pure branch-decompositions

We are going to assume that a subspace arrangement $\mathcal{V}$ and its rooted branch-decomposition $(T^b, \mathcal{L}^b)$ are given. For two nodes x, y of T^b, we say that $x \leq y$ if either $x = y$ or x is a descendant of y. We write $x < y$ if $x \leq y$ and $x \neq y$. For a node x of T^b, let $\mathcal{V}_x$ be the set of all subspaces $\mathcal{L}^b(\ell)$ where ℓ is a leaf of T^b with $\ell \leq x$ and let $B_x = \langle \mathcal{V}_x \rangle \cap \langle \mathcal{V} - \mathcal{V}_x \rangle$. Let $(T, \mathcal{L})$ be a branch-decomposition[6] of $\mathcal{V}_0 \subseteq \mathcal{V}$. Let x be a node of T^b such that $\mathcal{V}_x \subseteq \mathcal{V}_0$. We define $\mathcal{L}(T, u, v) = \{\mathcal{L}(w) : w \in A_v(T - uv)\}$ and write $\mathcal{L}_x(T, u, v) = \mathcal{L}(T, u, v) \cap \mathcal{V}_x$. (We remark that $\mathcal{L}(T, u, v)$ is a set of subspaces and $\langle \mathcal{L}(T, u, v) \rangle$ is the sum of members of $L(T, u, v)$.)

[6] One may consider $(T, \mathcal{L})$ as a (partial) solution whereas $(T^b, \mathcal{L}^b)$ is the given branch-decomposition over which dynamic programming is executed.

(a) T **(b)** T'

Figure 1 Constructing T' by forking at v by $\mathcal{V}_x$. ■ represents a leaf node mapped to an element of $\mathcal{V}_x$ by $\mathcal{L}$ and ○ represents a leaf node mapped to an element of $\mathcal{V}_0 - \mathcal{V}_x$ by $\mathcal{L}$.

We say that an edge uv of T *x-guards* its end v if $\langle \mathcal{L}_x(T,u,v)\rangle \cap B_x \subsetneq \langle \mathcal{L}_x(T,v,u)\rangle \cap B_x$. An edge uv of T is *x-degenerate* if $\langle \mathcal{L}_x(T,u,v)\rangle \cap B_x = \langle \mathcal{L}_x(T,v,u)\rangle \cap B_x$. A 2-edge path uvw of T is an *x-blocking path* if $\langle \mathcal{L}_x(T,u,v)\rangle \cap B_x = \langle \mathcal{L}_x(T,v,w)\rangle \cap B_x$, $\langle \mathcal{L}_x(T,w,v)\rangle \cap B_x = \langle \mathcal{L}_x(T,v,u)\rangle \cap B_x$, and neither uv nor vw is x-degenerate or x-guarding.

We give a general idea behind the notion of totally pure branch-decompositions. For $\mathcal{V}_x \subseteq \mathcal{V}_0$, one can consider the branch-decomposition 'induced' by $\mathcal{V}_x$ from $(T,\mathcal{L})$; such a branch-decomposition can be canonically defined by choosing a minimal subtree of T whose leaf set is mapped to $\mathcal{V}_x$ by $\mathcal{L}$ (and smoothing degree-2 nodes if necessary). Similarly, the branch-decomposition 'induced' by $\mathcal{V}_0 - \mathcal{V}_x$ can be obtained. Let $(T_x,\mathcal{L}_x)$ and $(T_{\bar{x}},\mathcal{L}_{\bar{x}})$ be the respective branch-decompositions. If uvw is an x-blocking path of T_x, then it can be shown that the connected component of $T_x - uv - vw$ containing v does not need to be mixed with another branch-decomposition in the future. Specifically, if the subtree of T homeomorphic to T_x is 'mixed' with some subtree of $T_{\bar{x}}$ in T, then one can 'untangle' the mixing: one 'lifts' the former subtree and 'plants' it on the x-blocking path uvw (so as to be rooted at a new node subdividing vw provided $\dim\langle \mathcal{L}_x(T,u,v)\rangle \cap \langle \mathcal{L}_x(T,v,u)\rangle \geq \dim\langle \mathcal{L}_x(T,v,w)\rangle \cap \langle \mathcal{L}_x(T,w,v)\rangle)$. This operation on $(T,\mathcal{L})$ is called the *forking* (see Figure 1). It can be proved that forking operations under above assumption do not increase the width. This is why we can 'forget' a subtree of T_x, namely the subtree of $T_x - uv - vw$ containing v. A similar observation can be made in regards to x-guarding edges, for which the related operation is *splitting*.

Then how do we know whether there is unwanted mixing in regards to an x-guarding edge or an x-blocking path? The following notions formalize this. An edge uv of T that x-guards v is called *improper* x-guarding if v has two neighbors v_1, v_2 in $T - uv$ such that $\mathcal{L}_x(T,v,v_1)$, $\mathcal{L}_x(T,v,v_2)$, and $\mathcal{L}(T,u,v) \cap (\mathcal{V}_0 - \mathcal{V}_x)$ are nonempty. An x-blocking path uvw of T is *improper* if v has a neighbor t in $T - u - w$ such that $\mathcal{L}_x(T,v,u)$, $\mathcal{L}_x(T,v,w)$, $\mathcal{L}_x(T,v,t)$, and $\mathcal{L}(T,v,t) \cap (\mathcal{V}_0 - \mathcal{V}_x)$ are nonempty.

When T has an x-degenerate edge e, it turns out that we can apply splitting operations at any x-degenerate edge and untangle T_x and $T_{\bar{x}}$ so that the new branch-decomposition is a disjoint union of T_x and $T_{\bar{x}}$ connected by a single edge (which will be incident with a subdividing node of the x-degenerate edge e). Hence, it is conceivable that we might be able to forget all nodes of T_x, possibly except for one node as a placeholder representing T_x. In this way, we request that any extension of T_x in the future shall be in the form of disjoint union plus one edge. However, it is possible that e is also a z-degenerate edge for some $z < x$. In this case, forcing the join of T_x and $T_{\bar{x}}$ at a subdividing node of e can violate the disjointness of T_z and $T_{\bar{z}}$. We want to prevent this, and it leads us to the definition of x-degenerate branch-decompositions.

For $\mathcal{S} \subseteq \mathcal{V}_0$, an edge uv of T is said to *cut* $\mathcal{S}$ if $\mathcal{L}(T,u,v) \cap \mathcal{S} \neq \emptyset$ and $\mathcal{L}(T,v,u) \cap \mathcal{S} \neq \emptyset$. We say that $(T,\mathcal{L})$ is *x-degenerate* if T has an x-degenerate edge uv such that uv cuts $\mathcal{V}_x$ and for all $z < x$, if $(T,\mathcal{L})$ is z-degenerate, then uv does not cut $\mathcal{V}_z$. Such an edge uv is called *improper* x-degenerate. Note that if x is a leaf of T^b, then $(T,\mathcal{L})$ is not x-degenerate because T has no edge cutting $\mathcal{V}_x$. We say that $(T,\mathcal{L})$ is *x-disjoint* if $\mathcal{V}_0 = \mathcal{V}_x$ or T has an edge uv such that $\mathcal{L}(T,u,v) = \mathcal{V}_x$ and v is incident with an improper x-degenerate edge.

We say that $(T, \mathcal{L})$ is *x-pure* if the following hold.

- If $(T, \mathcal{L})$ is x-degenerate, then $(T, \mathcal{L})$ is x-disjoint.
- If $(T, \mathcal{L})$ is not x-degenerate, then all x-blocking paths and all x-guarding edges of T are not improper.

We say that a branch-decomposition $(T, \mathcal{L})$ of $\mathcal{V}_0$ is *totally pure with respect to* $(T^b, \mathcal{L}^b)$ if $(T, \mathcal{L})$ is x-pure for all nodes x of T^b with $\mathcal{V}_x \subseteq \mathcal{V}_0$. We prove that if the branch-width of a subspace arrangement is at most k, then there exists a totally pure branch-decomposition of the subspace arrangement whose width is at most k. The proof strategy is to apply forking and splitting operations for every node x of T^b in a bottom-up manner. If $(T, \mathcal{L})$ is x-degenerate, then applying the operations will create a new branch-decomposition which is x-disjoint. If it is not x-degenerate, then the operations will resolve entanglements at the improper x-guarding edges and at the improper x-blocking paths so that no improper ones are left. For this approach to work, we need to ensure that applying these operations do not create new entanglements at nodes z of T^b where disentanglement already happened (or is happening now). That is, z-disjointness is preserved, and no new improper z-guarding edge or z-blocking path is created.

▶ **Proposition 3.1.** *Let $(T^b, \mathcal{L}^b)$ be a rooted branch-decomposition of a subspace arrangement $\mathcal{V}$ and let $\mathcal{V}_0 \subseteq \mathcal{V}$. If the branch-width of $\mathcal{V}_0$ is at most k, then $\mathcal{V}_0$ has a branch-decomposition of width at most k that is totally pure with respect to $(T^b, \mathcal{L}^b)$.*

4 Namus

Let $\mathbb{F}$ be a finite field and let B be a subspace of $\mathbb{F}^r$ of dimension θ. In this section, we introduce the data structure for encoding partial solutions and operations on this data structure required for dynamic programming. For a tree T, an *incidence* is a pair (v, e) of a node v of T and an edge e incident with v. Let $\mathcal{I}(T)$ be the union of $\{(*, \emptyset), (0, \emptyset)\}$ and the set of all incidences of T. A *B-namu* Γ is a quadruple (T, α, λ, U) of

- a subcubic tree T having at least one node,
- a function α from $\mathcal{I}(T)$ to the set of all subspaces of B,
- a function λ from the union of $\{\emptyset\}$ and the set of all edges of T to the set of integers, and
- a subspace U of B

such that

- **(i)** for every two-edge path v_0, e_1, v_1, e_2, v_2 in T, $\alpha(v_0, e_1)$ is a subspace of $\alpha(v_1, e_2)$,
- **(ii)** for all incidences (v, e) of T, $\alpha(v, e)$ is a subspace of U,
- **(iii)** $\alpha(*, \emptyset) = U$, $\alpha(0, \emptyset) = \{0\}$, and $\lambda(\emptyset) = 0$,
- **(iv)** for every edge $e = uv$ of T, $\lambda(e) \geq \dim \alpha(v, e) \cap \alpha(u, e)$.

The *width* of a B-namu $\Gamma = (T, \alpha, \lambda, U)$ is the maximum of $\lambda(e)$ over all edges $e = uv$ of T. (If T has no edges, then the width of Γ is defined to be 0.) For a B-namu $\Gamma = (T, \alpha, \lambda, U)$, we write $T(\Gamma) = T$.

Canonical B-namus. The *canonical B-namu* of a branch-decomposition $(T, \mathcal{L})$ of a subspace arrangement $\mathcal{V}$ is the B-namu (T, α, λ, U) such that

- $\alpha(v, e) = B \cap \sum_{x \in A_v(T-e)} \mathcal{L}(x)$ for each node v of T and an edge e incident with v,
- $\lambda(e) = \dim \sum_{x \in A_u(T-e)} \mathcal{L}(x) \cap \sum_{y \in A_v(T-e)} \mathcal{L}(y)$ for each edge $e = uv$ of T,
- $U = B \cap \sum_{x \in A(T)} \mathcal{L}(x)$ where $A(T)$ is the set of all leaves of a tree T.

Projections. For two subspaces B and B' with $B' \subseteq B$, we define the *projection* $\Gamma|_{B'}$ of a B-namu $\Gamma = (T, \alpha, \lambda, U)$ on B' as the B'-namu $(T, \alpha', \lambda', U')$ such that $U' = U \cap B'$, $\alpha'(v, e) = \alpha(v, e) \cap B'$ for all incidences (v, e) of T, and $\lambda'(e) = \lambda(e)$ for all edges e of T.

Compact B-namus. We have two operations, trimming and compressing, on B-namus, which will transform a B-namu into a 'compact' B-namu. Roughly speaking, *trimming* is an operation to remove irrelevant edges and *compressing* is an operation to suppress redundant edges. Irrelevant edges are those edges which can be 'untangled' by forking or splitting operations as described in Section 3. As addressed in Proposition 3.1, there exists a branch-decomposition of minimum width that is totally pure with respect to $(T^b, \mathcal{L}^b)$. Such a branch-decomposition is minimally 'mixed' at every node x of T^b in the sense that forking and splitting operations has been fully applied at every x without causing additional mixing. The idea behind trimming is that, for keeping track of totally pure branch-decompositions in the dynamic programming algorithm, those edges to be untangled by forking or splitting operations can be ignored. For a B-namu Γ, let $\mathrm{trim}(\Gamma)$ denote the B-namu obtained by trimming Γ.

For a node x of $(T^b, \mathcal{L}^b)$, can we bound the size of an arbitrary trimmed B_x-namu Γ, namely the size of $T(\Gamma)$? As $T(\Gamma)$ is a subcubic tree, bounding the size of $T(\Gamma)$ is equivalent to bounding the diameter of $T(\Gamma)$. By condition (i) in the definition of B-namu, it is not difficult to see that $T(\Gamma)$ has a large diameter if and only if $T(\Gamma)$ contains a long path in which any length-two path is x-blocking. Assuming that Γ is trimmed, such a long path induces a substructure in which every internal node has degree two and α maps every incidence to the same subspace. That is, the information on such a path dictated by Γ is almost uniform except that the values of λ changes over the edges and the values of λ can be viewed as an integer sequence. Now, the idea of compressing operation is to keep only the edges associated with local minimum and maximum values of this integer sequence and ignore all other edges. An integer sequence obtained in this way is called a *typical sequence* in the literature [1] and it is known to have length at most $2k + 1$ when the integers are in the range $\{0, \ldots, k\}$.

We say that a B-namu is *compact* if it contains no 'irrelevant' nodes or edges so that trimming or compressing does not affect to the B-namu. Let $U_k(B)$ be the set of all compact B-namus Γ of width at most k such that $V(T(\Gamma)) = \{1, 2, \ldots, n\}$ for some integer n. The previous discussion is summarized in the next statement, which ensures in Section 5 that the number of partial solutions stored at each node of $(T^b, \mathcal{L}^b)$ for the dynamic programming algorithm will be bounded.

▶ **Lemma 4.1.** *The set $U_k(B)$ contains at most $f(k, \theta, |\mathbb{F}|)$ elements and can be generated from B in $g(k, \theta, |\mathbb{F}|)$ steps for some functions f and g.*

Sum of two B-namus. For two B-namus $\Gamma_1 = (T_1, \alpha_1, \lambda_1, U_1)$ and $\Gamma_2 = (T_2, \alpha_2, \lambda_2, U_2)$, we define a *sum* $(T, \alpha, \lambda, U_1 + U_2)$ of Γ_1 and Γ_2. Roughly speaking, we first take a tree T such that a subdivision of T_1 is a subtree of T and a subdivision of T_2 is a subtree of T (see Figure 2). For each incidence (v, e) of T, if it corresponds to both an incidence (v_1, e_1) of T_1 and an incidence (v_2, e_2) of T_2, then $\alpha(v, e)$ is the sum of $\alpha_1(v_1, e_1)$ and $\alpha_2(v_2, e_2)$, and if it corresponds to only one of (v_1, e_1) and (v_2, e_2), say (v_1, e_1), then $\alpha(v, e) = \alpha_1(v_1, e_1)$. Similarly, we can define λ on every edge of T (with some correction term). The formal definition is given in the full version. Note that a sum of two B-namus is not unique because there are many choices of taking a tree T. Given B-namus Γ_1 and Γ_2, let us denote by $\Gamma_1 \oplus \Gamma_2$ the set of all sums of Γ_1 and Γ_2.

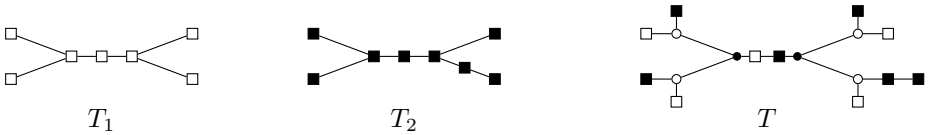


Figure 2 Obtaining a sum of two B-namus.

▶ **Lemma 4.2.** *Let Γ_1, Γ_2 be compact B-namus of width at most k. Then the set $\Gamma_1 \oplus \Gamma_2$ contains at most $2^{2^{2(2\theta k+\theta+k+3)}}$ B-namus.*

Comparing two B-namus. A B-namu $(T', \alpha', \lambda', U)$ is a *subdivision* of a B-namu $\Gamma = (T, \alpha, \lambda, U)$ if T' is a subdivision of T, $\alpha'(v', e') = \alpha(v, e)$, and $\lambda'(e') = \lambda(e)$ for every incidence (v', e') of T' and its corresponding incidence (v, e) of T.

For two B-namus $\Gamma_1 = (T_1, \alpha_1, \lambda_1, U_1)$ and $\Gamma_2 = (T_2, \alpha_2, \lambda_2, U_2)$, we say that $\Gamma_1 \leq \Gamma_2$ if $T_1 = T_2$, $\alpha_1 = \alpha_2$, $U_1 = U_2$ and $\lambda_1(e) \leq \lambda_2(e)$ for every edge e of T_1. For two B-namus Γ_1 and Γ_2, we say that $\Gamma_1 \preccurlyeq \Gamma_2$ if there exist a subdivision Γ_1' of Γ_1 and a subdivision Γ_2' of Γ_2 such that $\Gamma_1' \leq \Gamma_2'$.

▶ **Lemma 4.3.** *For two B-namus Δ and Γ, we can decide whether $\Delta \preccurlyeq \Gamma$ by executing at most $f(|V(T(\Delta))|, |V(T(\Gamma))|, \theta, |\mathbb{F}|)$ comparison operations (on integers and on subspaces of B) for some function f.*

5 The algorithm

We present an algorithm to solve the Branch-Width problem. Given a matrix M and a set Y of column indices, $M[Y]$ denotes the submatrix of M induced by columns indexed by Y.

Preprocessing. We will first describe the preprocessing steps to reduce the input size. The subspace arrangement of n subspaces is given by an $r \times m$ matrix where r and m could be arbitrary large. Our aim here is to reduce r and m or confirm that branch-width is larger than k. Eventually, we will convert the input into a smaller one. Furthermore, we will convert M into the reduced row echelon form, which is crucial for our algorithm for computing the transcript of a branch-decomposition in Theorem 2.1. In the Branch-Width problem, if a branch-decomposition of width at most k exists, then we say that $(M, \mathcal{I}, k)$ is a YES instance. Otherwise, it is a NO instance. For a matrix M, let $\mathrm{col}(M)$ be the column space of M, that is the span of all column vectors of M.

▶ **Lemma 5.1.** *Let $\mathbb{F}$ be a finite field and let k be a nonnegative integer. Let $n \geq 2$. Let M be an $r \times m$ matrix over $\mathbb{F}$ with an ordered partition $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ of $\{1, 2, \ldots, m\}$ and let V_i be the column space of $M[I_i]$ for every i. In time $O(rm^2 + (k+1)rmn)$, we can either find $i \in \{1, 2, \ldots, n\}$ such that $\dim(V_i \cap (\sum_{j \neq i} V_i)) > k$ or find an $r' \times m'$ matrix M' over $\mathbb{F}$ with an ordered partition $\mathcal{I}' = \{I_1', I_2', \ldots, I_n'\}$ of $\{1, 2, \ldots, m'\}$ such that*

- **(i)** *$r' \leq m' \leq kn$,*
- **(ii)** *M' is of the reduced row echelon form with no zero rows,*
- **(iii)** *for each i, the column vectors of $M'[I_i']$ are linearly independent and $|I_i'| \leq k$,*
- **(iv)** *for each i, $\mathrm{col}(M'[I_i']) \subseteq \mathrm{col}(M'[\{1, 2, \ldots, m'\} - I_i'])$,*
- **(v)** *$(M, \mathcal{I}, k)$ is a YES instance with a branch-decomposition $(T, \mathcal{L})$ if and only if $(M', \mathcal{I}', k)$ is a YES instance with $(T, \mathcal{L}')$ where $\mathcal{L}'$ maps a leaf v to $\mathrm{col}(M'[I_i'])$ whenever $\mathcal{L}$ maps v to $\mathrm{col}(M[I_i])$.*

The full set. Our dynamic programming algorithm constructs a set of compact B_x-namus of width at most k at each node x of the given branch-decomposition $(T^b, \mathcal{L}^b)$ in a bottom-up manner. This set is called the *full set at x of width k with respect to* $(T^b, \mathcal{L}^b)$ and written as $\mathrm{FS}_k(x; T^b, \mathcal{L}^b)$ or $\mathrm{FS}_k(x)$ for brevity. The full set needs to be defined so that at the root node $\mathfrak{r}$ of T^b, $\mathrm{FS}_k(\mathfrak{r}) \neq \emptyset$ if and only if the branch-width of $\mathcal{V}$ is at most k. Moreover, we need to ensure that the full set at every node of T^b can be constructed from the full sets of its children.

Roughly speaking, the full set at x is an upward closed set of $(U_k(B_x), \preccurlyeq)$, where $U_k(B_x)$ and $\preccurlyeq$ are defined in the previous section. Every minimal[7] element in this upward closed set is a B-namu that can be obtained from a branch-decomposition $(T, \mathcal{L})$ of $\mathcal{V}_x$ having width at most k that is totally pure with respect to $(T^b, \mathcal{L}^b)$ by discarding some subtrees of T. This concept is captured in the notion of *reduced B-namus* below. Any discarded subtrees keep the rest of T connected. The subtrees of T that will not be mixed with another partial solution shall qualify as the disposable parts. Two types of disposable parts arise: one is a subtree consisting of x-blocked nodes (which will be defined soon), and the other type is a subtree whose entire leaf set is precisely mapped with $\mathcal{V}_z$ for some $z \leq x$ such that $(T, \mathcal{L})$ is z-degenerate. In particular, if $(T, \mathcal{L})$ is x-degenerate, then we discard all nodes except one node. Because we only consider totally pure branch-decompositions $(T, \mathcal{L})$, if $(T, \mathcal{L})$ is z-degenerate for some $z < x$, then $(T, \mathcal{L})$ is a disjoint union of $\mathcal{V}_z$ and $\mathcal{V}_x - \mathcal{V}_z$ joined via a single edge. It is intuitively easy to understand that we want to keep the subtree containing $\mathcal{V}_z$ intact from any mixing in the future and thus want to discard this part. However, implementing this idea with full technical details is quite tricky.

Moreover, not every reduced B-namu obtained in this way can be a member of a full set. A technical condition called *k-safeness* must be met by any edge that gets discarded. This condition is expressed as an inequality, indicating that when the current partial solution grows into a branch-decomposition for $\mathcal{V}$, the width at the forgotten edge is at most k.

For a B-namu $\Gamma = (T, \alpha, \lambda, U)$ and a subtree T' of T, we say that a B-namu $\Gamma' = (T', \alpha', \lambda', U)$ is *induced by T' from* Γ if $\alpha'(v, e) = \alpha(v, e)$ and $\lambda'(e) = \lambda(e)$ for every incidence (v, e) of T'. For a node x of T^b and a branch-decomposition $(T, \mathcal{L})$ of $\mathcal{V}_x$ which is totally pure with respect to $(T^b, \mathcal{L}^b)$, we obtain the *reduced B_x-namu of* $(T, \mathcal{L})$ induced by a subtree T' of T from the canonical B_x-namu of $(T, \mathcal{L})$ where T' is obtained by the following rule.

- If $(T, \mathcal{L})$ is x-degenerate, then T' is a subtree having only one node of T.
- If $(T, \mathcal{L})$ is not x-degenerate, then T' is obtained by deleting every node w if
  - **(i)** there is an x-blocking path $v_1 v v_2$ centered at $v \neq w$ such that there is a path from w to v in $T - vv_1 - vv_2$, or
  - **(ii)** some edge uv x-guards $v \neq w$ such that there is a path from w to v in $T - uv$, or
  - **(iii)** w is a node of a subtree T'' of T with a root $v \neq w$ having two children such that the leaf set of T'' is precisely mapped to $\mathcal{V}_z$ for some $z < x$ and $(T, \mathcal{L})$ is z-degenerate.

A node w satisfying (i) or (ii) is said to be *x-blocked*. A branch-decomposition $(T, \mathcal{L})$ is *k-safe with respect to x* if for every edge uv of T which is not contained in T',

$$\dim \sum_{s \in A_v(T-uv)} \mathcal{L}(s) \cap \sum_{t \in A_u(T-uv)} \mathcal{L}(t) + \dim B_x - \dim B_x \cap \sum_{t \in A_u(T-uv)} \mathcal{L}(t) \leq k.$$

Now, for each node x of T^b, *the full set at x of width k with respect to* $(T^b, \mathcal{L}^b)$ is defined

[7] Technically, a minimal element in our definition might not be a compact B_x-namu but just a trimmed B_x-namu. However, a compact B-namu of the minimal element defines the same upper set due to the transitivity of $\preccurlyeq$.

as the set of all Γ in $U_k(B_x)$ such that $\Delta \preccurlyeq \Gamma$ for the reduced B_x-namu Δ of some branch-decomposition $(T, \mathcal{L})$ of $\mathcal{V}_x$ having width at most k that is k-safe with respect to x, and totally pure with respect to $(T^b, \mathcal{L}^b)$.

Dynamic programming. For computing the full sets, we assume the following are given:

- A rooted branch-decomposition $(T^b, \mathcal{L}^b)$ of $\mathcal{V}$ of width at most θ.
- A set of transition matrices $\{T_v\}_{v \in V(T^b)}$ of some transcript Λ of $(T^b, \mathcal{L}^b)$.

A B-namu $\Gamma = (T, \alpha, \lambda, U)$ is a *k-safe extension of* $\mathrm{trim}(\Gamma)$ if for every edge uv in $E(T) - E(\mathrm{trim}(T))$, we have $\lambda(uv) + \dim U - \max(\dim \alpha(v, uv), \dim \alpha(u, uv)) \leq k$. For two sets $\mathcal{R}_1$, $\mathcal{R}_2$ of B-namus, we define $\mathcal{R}_1 \oplus \mathcal{R}_2$ as the set $\bigcup_{\Gamma_1 \in \mathcal{R}_1, \Gamma_2 \in \mathcal{R}_2} \Gamma_1 \oplus \Gamma_2$. Note that for two children x_1, x_2 of a node x in T^b, when we compute $\mathrm{FS}_k(x_1) \oplus \mathrm{FS}_k(x_2)$, we regard $\mathrm{FS}_k(x_1)$, $\mathrm{FS}_k(x_2)$ as the sets of $(B_{x_1} + B_{x_2})$-namus. Thus, $\mathrm{FS}_k(x_1) \oplus \mathrm{FS}_k(x_2)$ is well defined. If B' is a subspace of B, then we define $\mathcal{R}|_{B'}$ as the set of projections $\Gamma|_{B'}$ for all $\Gamma \in \mathcal{R}$. For a subspace B of $\mathbb{F}^r$ and a set $\mathcal{R}$ of B-namus, the set $\mathsf{up}_k(\mathcal{R}, B)$ is the collection of all B-namus $\Gamma \in U_k(B)$ with $\mathrm{trim}(\Gamma') \preccurlyeq \Gamma$ for some $\Gamma' \in \mathcal{R}$ such that Γ' is a k-safe extension of $\mathrm{trim}(\Gamma')$.

▶ **Proposition 5.2.** *Let k be a nonnegative integer. Let $(T^b, \mathcal{L}^b)$ be a rooted branch-decomposition of a subspace arrangement $\mathcal{V}$ over a finite field $\mathbb{F}$ of width at most θ.*

- *For a leaf ℓ of T^b, we have $\mathrm{FS}_k(\ell) = \{\Delta_\ell\}$ where $\Delta_\ell = (T, \alpha, \lambda, B_\ell)$ is the B_ℓ-namu such that T is a tree with $V(T) = \{1\}$.*
- *For two children x_1 and x_2 of a node x in T^b, $\mathrm{FS}_k(x) = \mathsf{up}_k((\mathrm{FS}_k(x_1) \oplus \mathrm{FS}_k(x_2))|_{B_x}, B_x)$.*
- *For the root node $\mathfrak{r}$ of T^b, $\mathrm{FS}_k(\mathfrak{r}) \neq \emptyset$ if and only if the branch-width of $\mathcal{V}$ is at most k.*

Moreover, we can compute $\mathrm{FS}_k(\mathfrak{r})$, and construct a (rooted) branch-decomposition of $\mathcal{V}$ of width at most k if $\mathrm{FS}_k(\mathfrak{r}) \neq \emptyset$, in time $f(k, \theta, |\mathbb{F}|)|\mathcal{V}|$ for some function f.

Proposition 5.2 states that when $\mathrm{FS}_k(x) \neq \emptyset$, the same B-namu or a better one can be constructed by conducting operations on B-namus in the full set at its child nodes. Therefore, when $\mathrm{FS}_k(\mathfrak{r}) \neq \emptyset$, one can identify a B-namu Γ_x at each node x of T^b which participates in the construction of the element in $\mathrm{FS}_k(\mathfrak{r})$. Additionally, we have the information including how Γ_x's are combined and how the combined B-namu is related to the B-namu at its parent. We construct a branch-decomposition of $\mathcal{V}$ having width at most k by backtracking based on such information. However, proving the correctness of this backtracking algorithm is highly nontrivial. For this, we introduce the notion of *witnesses*. Details are in the full version.

Summary. Now we are ready to present the proof of Theorem 1.1.

Proof of Theorem 1.1. We preprocess the input by applying Lemma 5.1 to $(M, \mathcal{I}, k)$ in time $O(rm^2 + (k+1)rmn)$ and obtain an equivalent instance $(M', \mathcal{I}', k)$ as described in Lemma 5.1 and otherwise, we confirm that the branch-width of $\mathcal{V}$ exceeds k. We may assume that $k > 0$ because if $I'_i = \emptyset$ for all i, then every branch-decomposition has width 0. Henceforth, we assume $M = M'$, $\mathcal{I} = \mathcal{I}'$, $V_i = \mathrm{col}(M'[I'_i])$ to simplify notations.

We may also assume that $\dim V_i \neq 0$ for all i because otherwise we delete all such V_i and later we can extend a branch-decomposition of $\mathcal{V} - \{V_i\}$ to that of $\mathcal{V}$ of the same width. After the preprocessing, if $n = 1$, then an arbitrary branch-decomposition has width 0 and so we simply output an arbitrary branch-decomposition of $\mathcal{V}$. If $n = 2$, then the branch-width is at most k because $\dim V_1, \dim V_2 \leq k$ by (iii) of Lemma 5.1. So we may assume that $n \geq 3$.

We will apply iterative compression on $\mathcal{V}_i = \{V_1, \ldots, V_i\}$ for $i = 3, \ldots, n$. We initially start with a trivial branch-decomposition $(T_2, \mathcal{L}_2)$ of $\mathcal{V}_2 = \{V_1, V_2\}$ having width at most k. We carry out a COMPRESSION STEP for each $i = 3, \ldots, n$ as follows.

(1) By adding a new leaf v to T_{i-1} and extending $\mathcal{L}_{i-1}$ to map v to V_i, we create a branch-decomposition $(T'_i, \mathcal{L}'_i)$ of $\mathcal{V}_i$. Note that the width of $(T'_i, \mathcal{L}'_i)$ is at most $2k$ because $(T_{i-1}, \mathcal{L}_{i-1})$ has width at most k and V_i has dimension at most k.

(2) We use the algorithm in Theorem 2.1 to compute transition matrices $\{T_v\}_{v \in V(T'_i)}$ of the transcript for $(T'_i, \mathcal{L}'_i)$ in time $O(k^3 n^2)$. Note that the submatrix $M[I_1 \cup I_2 \cup \cdots \cup I_i]$ is in reduced row echelon form and so we can apply Theorem 2.1 by ignoring zero rows.

(3) Given a rooted branch-decomposition $(T'_i, \mathcal{L}'_i)$ of $\mathcal{V}_i$ of width at most $2k$ and a set of the transition matrices $\{T_v\}_{v \in V(T'_i)}$, we compute the full set in time $g(k, 2k, |\mathbb{F}|)i$ for some function g by Proposition 5.2. If the full set at the root is empty, then the branch-width of $\mathcal{V}_i$ is larger than k. If so, we conclude that the branch-width of $\mathcal{V}$ is larger than k and stop. If the full set at the root is nonempty, then the algorithm in Proposition 5.2 also provides a branch-decomposition $(T_i, \mathcal{L}_i)$ of $\mathcal{V}_i$ having width at most k.

If this algorithm finds $(T_n, \mathcal{L}_n)$, then $(T_n, \mathcal{L}_n)$ is a branch-decomposition of $\mathcal{V}$ having width at most k. For each i, (1)–(3) runs in at most $O(k^3 n^2) + f(k, |\mathbb{F}|)n$ time for some function f and therefore the total running time of this step is $O(k^3 n^3) + f(k, |\mathbb{F}|)n^2$. ◀

References

1 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996. doi:10.1006/jagm.1996.0049.

2 Hans L. Bodlaender and Dimitrios M. Thilikos. Constructive linear time algorithms for branchwidth. In *Automata, languages and programming (Bologna, 1997)*, volume 1256 of *Lecture Notes in Comput. Sci.*, pages 627–637. Springer, Berlin, 1997. doi:10.1007/3-540-63165-8_217.

3 James F. Geelen, A. M. H. Gerards, Neil Robertson, and Geoff Whittle. On the excluded minors for the matroids of branch-width k. *J. Combin. Theory Ser. B*, 88(2):261–265, 2003. doi:10.1016/S0095-8956(02)00046-1.

4 Petr Hliněný. A parametrized algorithm for matroid branch-width. *SIAM J. Comput.*, 35(2):259–277, 2005. doi:10.1137/S0097539702418589.

5 Petr Hliněný. Branch-width, parse trees, and monadic second-order logic for matroids. *J. Combin. Theory Ser. B*, 96(3):325–351, 2006. doi:10.1016/j.jctb.2005.08.005.

6 Petr Hliněný and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008. doi:10.1137/070685920.

7 Jisu Jeong, Eun Jung Kim, and Sang-il Oum. Constructive algorithm for path-width of matroids. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 1695–1704, Philadelphia, PA, USA, 2016. Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611974331.ch116.

8 Jisu Jeong, Eun Jung Kim, and Sang-il Oum. The "art of trellis decoding" is fixed-parameter tractable. *IEEE Trans. Inform. Theory*, 63(11):7178–7205, 2017. An extended abstract appeared in a conference proceeding [7]. doi:10.1109/TIT.2017.2740283.

9 Jens Lagergren and Stefan Arnborg. Finding minimal forbidden minors using a finite congruence. In *Automata, languages and programming (Madrid, 1991)*, volume 510 of *Lecture Notes in Comput. Sci.*, pages 532–543. Springer, Berlin, 1991. doi:10.1007/3-540-54233-7_161.

10 Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

11 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.

12 Neil Robertson and Paul D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Combin. Theory Ser. B*, 52(2):153–190, 1991. doi:10.1016/0095-8956(91)90061-N.

13 Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994. doi:10.1007/BF01215352.

14 Dimitrios M. Thilikos and Hans L. Bodlaender. Constructive linear time algorithms for branchwidth. Technical Report UU-CS 2000-38, Universiteit Utrecht, 2000. URL: http://archive.cs.uu.nl/pub/RUU/CS/techreps/CS-2000/2000-38.pdf.

15 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Constructive linear time algorithms for small cutwidth and carving-width. In *Algorithms and computation (Taipei, 2000)*, volume 1969 of *Lecture Notes in Comput. Sci.*, pages 192–203. Springer, Berlin, 2000. doi:10.1007/3-540-40996-3_17.

Optimally Sorting Evolving Data

Juan Jose Besa
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
jjbesavi@uci.edu
https://orcid.org/0000-0002-5676-7011

William E. Devanny[1]
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
wdevanny@uci.edu

David Eppstein
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
eppstein@uci.edu

Michael T. Goodrich
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
goodrich@uci.edu

Timothy Johnson
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
tujohnso@uci.edu

Abstract

We give optimal sorting algorithms in the *evolving data* framework, where an algorithm's input data is changing while the algorithm is executing. In this framework, instead of producing a final output, an algorithm attempts to maintain an output close to the correct output for the current state of the data, repeatedly updating its best estimate of a correct output over time. We show that a simple repeated insertion-sort algorithm can maintain an $O(n)$ Kendall tau distance, with high probability, between a maintained list and an underlying total order of n items in an evolving data model where each comparison is followed by a swap between a random consecutive pair of items in the underlying total order. This result is asymptotically optimal, since there is an $\Omega(n)$ lower bound for Kendall tau distance for this problem. Our result closes the gap between this lower bound and the previous best algorithm for this problem, which maintains a Kendall tau distance of $O(n \log \log n)$ with high probability. It also confirms previous experimental results that suggested that insertion sort tends to perform better than quicksort in practice.

2012 ACM Subject Classification Theory of computation → Sorting and searching

Keywords and phrases Sorting, Evolving data, Insertion sort

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.81

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.03350`.

Funding This article reports on work supported by the DARPA under agreement no. AFRL FA8750-15-2-0092. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. This work was also supported in part from NSF grants 1228639, 1526631, 1217322, 1618301, and 1616248.

[1] Supported by an NSF Graduate Research Fellowship under grant DGE-1321846.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 81; pp. 81:1–81:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the classic version of the sorting problem, we are given a set, S, of n comparable items coming from a fixed total order and asked to compute a permutation that places the items from S into non-decreasing order, and it is well-known that this can be done using $O(n \log n)$ comparisons, which is asymptotically optimal (e.g., see [6, 8, 14]). However, there are a number of interesting applications where this classic version of the sorting problem doesn't apply.

For instance, consider the problem of maintaining a ranking of a set of sports teams based on the results of head-to-head matches. A typical approach to this sorting problem is to assume there is a fixed underlying total order for the teams, but that the outcomes of head-to-head matches (i.e., comparisons) are "noisy" in some way. In this formulation, the ranking problem becomes a one-shot optimization problem of finding the most-likely fixed total order given the outcomes of the matches (e.g., see [5, 7, 9, 10, 15]). In this paper, we study an alternative, complementary motivating scenario, however, where instead of there being a fixed total order and noisy comparisons we have a scenario where comparisons are accurate but the underlying total order is evolving. This scenario, for instance, captures the real-world phenomenon where sports teams make mid-season changes to their player rosters and/or coaching staffs that result in improved or degraded competitiveness relative to other teams. That is, we are interested in the sorting problem for *evolving data*.

1.1 Related Prior Work for Evolving Data

Anagnostopoulos *et al.* [1] introduce the *evolving data* framework, where an input data set is changing while an algorithm is processing it. In this framework, instead of an algorithm taking a single input and producing a single output, an algorithm attempts to maintain an output close to the correct output for the current state of the data, repeatedly updating its best estimate of the correct output over time. For instance, Anagnostopoulos *et al.* [1] mention the motivation of maintaining an Internet ranking website that displays an ordering of entities, such as political candidates, movies, or vacation spots, based on evolving preferences.

Researchers have subsequently studied other interesting problems in the evolving data framework, including the work of Kanade *et al.* [13] on stable matching with evolving preferences, the work of Huang *et al.* [12] on selecting top-k elements with evolving rankings, the work of Zhang and Li [18] on shortest paths in evolving graphs, the work of Anagnostopoulos *et al.* [2] on st-connectivity and minimum spanning trees in evolving graphs, and the work of Bahmani *et al.* [3] on PageRank in evolving graphs. In each case, the goal is to maintain an output close to the correct one even as the underlying data is changing at a rate commensurate to the speed of the algorithm. By way of analogy, classical algorithms are to evolving-data algorithms as throwing is to juggling.

1.2 Problem Formulation for Sorting Evolving Data

With respect to the sorting problem for evolving data, following the formulation of Anagnostopoulos *et al.* [1], we assume that we have a set, S, of n distinct items that are properly ordered according to a total order relation, "$<$". In any given time step, we are allowed to compare any pair of items, x and y, in S according to the "$<$" relation and we learn the correct outcome of this comparison. After we perform such a comparison, α pairs of items that are currently consecutive according to the "$<$" relation are chosen uniformly at random and their relative order is swapped. As in previous work [1], we focus on the case

where $\alpha = 1$, but one can also consider versions of the problem where the ratio between comparisons and random consecutive swaps is something other than one-to-one. Still, this simplified version with a one-to-one ratio already raises some interesting questions.

Since it is impossible in this scenario to maintain a list that always reflects a strict ordering according to the "$<$" relation, our goal is to maintain a list with small *Kendall tau* distance, which counts the number of inversions, relative to the correct order.[2] Anagnostopoulos *et al.* [1] show that, for $\alpha = 1$, the Kendall tau distance between the maintained list and the underlying total order is $\Omega(n)$ in both expectation and with high probability. They also show how to maintain this distance to be $O(n \log \log n)$, with high probability, by performing a multiplexed batch of quicksort algorithms on small overlapping intervals of the list. Recently, Besa Vial *et al.* [4] empirically show that repeated versions of quadratic-time algorithms such as bubble sort and insertion sort seem to maintain an asymptotically optimal distance of $O(n)$. In fact, this linear upper bound seems to hold even if we allow α, the number of random swaps at each step, to be a much larger constant.

1.3 Our Contributions

The main contribution of the present paper is to prove that repeated insertion sort maintains an asymptotically optimal Kendall tau distance, with high probability, for sorting evolving data. This algorithm repeatedly makes in-place insertion-sort passes (e.g., see [6, 8]) over the list, l_t, maintained by our algorithm at each step t. Each such pass moves the item at position j to an earlier position in the list so long as it is bigger than its predecessor in the list. With each comparison done by this repeated insertion-sort algorithm, we assume that a consecutive pair of elements in the underlying ordered list, l'_t, are chosen uniformly at random and swapped. In spite of the uncertainty involved in sorting evolving data in this way, we prove the following theorem, which is the main result of this paper.

▶ **Theorem 1.** *Running repeated insertion-sorts algorithm, for every step $t = \Omega(n^2)$, the Kendall tau distance between the maintained list, l_t, and the underlying ordered list, l'_t, is $O(n)$ with exponentially high probability.*

That is, after an initialization period of $\Theta(n^2)$ steps, the repeated insertion-sort algorithm converges to a steady state having an asymptotically optimal Kendall tau distance between the maintained list and the underlying total order, with exponentially high probability. We also show how to reduce this initialization period to be $\Theta(n \log n)$ steps, with high probability, by first performing a quicksort algorithm and then following that with the repeated insertion-sort algorithm.

Intuitively, our proof of Theorem 1 relies on two ideas: the adaptivity of insertion sort and that, as time progresses, a constant fraction of the random swaps fix inversions. Ignoring the random swaps for now, when there are k inversions, a complete execution of insertion sort performs roughly $k + n$ comparisons and fixes the k inversions (e.g., see [6, 8]). If an ϵ fraction of the random swaps fix inversions, then during insertion sort $\epsilon(k + n)$ inversions are fixed by the random swaps and $(1 - \epsilon)(k + n)$ are introduced. Naively the total change in the number of inversions is then $(1 - 2\epsilon)(k + n) - k$ and when $k > \frac{1-2\epsilon}{2\epsilon}n$, the number of inversions decreases. So the number of inversions will decrease until $k = O(n)$.

This simplistic intuition ignores two competing forces involved in the heavy interplay between the random swaps and insertion sort's runtime, however, in the evolving data model,

[2] Recall that an *inversion* is a pair of items u and v such that u comes before v in a list but $u > v$. An *inversion* in a permutation π is a pair of elements $x \neq y$ with $x < y$ and $\pi(x) > \pi(y)$.

Algorithm 1 Repeated insertion sort pseudocode

```
function REPEATED_INSERTION_SORT(l)
    while true do
        for i ← 1 to n − 1 do
            j ← i
            while j > 0 and l[j] < l[j − 1] do
                swap l[j] and l[j − 1]
                j ← j − 1
```

which necessarily complicates our proof. First, random swaps can cause an insertion-sort pass to end too early, thereby causing insertion sort to fix fewer inversions than normal. Second, as insertion sort progresses, it decreases the chance for a random swap to fix an inversion. Analyzing these two interactions comprises the majority of our proof of Theorem 1.

In Section 3, we present a complete proof of Theorem 1. The most difficult component of Theorem 1's proof is Lemma 6, which lower bounds the runtime of insertion sort in the evolving data model. The proof of Lemma 6 is presented separately in Section 4.

Due to space requirements, some proofs are left to the Arxiv version of the paper.

2 Preliminaries

The sorting algorithm we analyze in this paper for the evolving data model is the repeated insertion-sort algorithm whose pseudocode is shown in Algorithm 1.

Formally, at time t, we denote the sorting algorithms' list as l_t and we denote the underlying total order as l'_t. Together these two lists define a permutation, σ_t, of the indices, where $\sigma_t(x) = y$ if the element at index x in l_t is at position y in l'_t. We define the *simulated final state at time t* to be the state of l obtained by freezing the current underlying total order, l'_t, (i.e., no more random swaps) and simulating the rest of the current round of insertion sort (we refer to each iteration of the **while-true** loop in Algorithm 1 as a *round*). We then define a *frozen-state* permutation, $\hat{\sigma}_t$, where $\hat{\sigma}_t(x) = y$ if the element at index x in the simulated final state at time t as at index y in l'_t.

Let us denote the number of inversions at time t, in σ_t, with I_t. Throughout the paper, we may choose to drop time subscripts if our meaning is clear. The Kendall tau distance between two permutations π_1 and π_2 is the number of pairs of elements $x \neq y$ such that $\pi_1(x) < \pi_1(y)$ and $\pi_2(x) > \pi_2(y)$. That is, the Kendall tau distance between l_t and l'_t is equal to I_t, the number of inversions in σ_t. Figure 1 shows the state of l, l', I, and σ for two steps of an insertion sort (but not in the same round).

As the inner **while**-loop of Algorithm 1 executes, we can view l as being divided into three sets: the set containing just the *active* element, $l[j]$ (which we view as moving to the left, starting from position i, as it is participating in comparisons and swaps), the *semi-sorted* portion, $l[0 : i]$, not including $l[j]$, and the *unsorted* portion, $l[i + 1 : n - 1]$. Note that if no random adjacent swaps were occurring in l' (that is, if we were executing insertion-sort in the classical algorithmic model), then the semi-sorted portion would be in sorted order.

To understand the nature of the inversions in the semi-sorted portion, we will use the *Cartesian tree* [17]. Given a list, L, of m numbers with no two equal numbers, the Cartesian tree of L is a binary rooted tree on the numbers where the root is the minimum element $L[k]$, the left subtree of the root is the Cartesian tree of $L[0 : k - 1]$, and the right subtree of the root is the Cartesian tree of $L[k + 1 : m]$. In our analysis, we will primarily consider the Cartesian tree of the simulated final state at time t where $L[k] = \hat{\sigma}_t(k)$ in the frozen-state

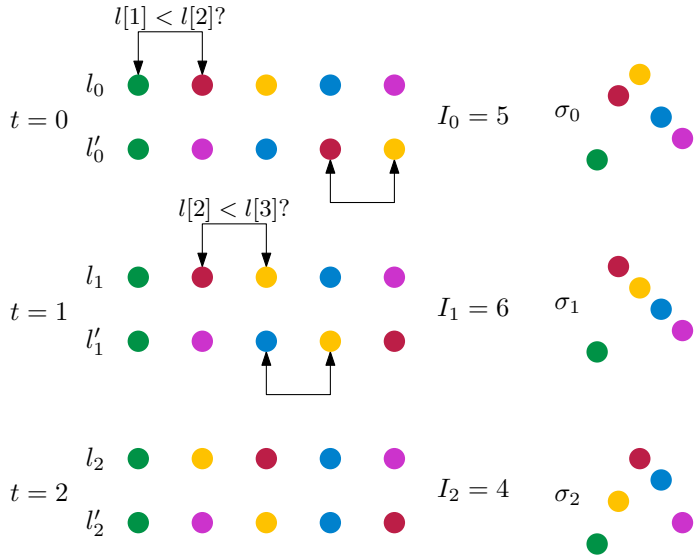


Figure 1 Examples of l, l', I, and σ over two steps of an algorithm. In the first step the green and red elements are compared in l and the red and yellow elements are swapped in l'. In the second step the red and yellow elements are compared and swapped in l and the blue and yellow elements are swapped in l'.

permutation $\hat{\sigma}_t$. We also choose to include two additional elements, $L[-1] = -1$ and $L[n] = n$, for boundary cases.

We call the path from the root to the rightmost leaf of the Cartesian tree the (right-to-left) minima path as the elements on this path are the right-to-left minima in the list. For a minimum, $l[k]$, denote with $M(k)$ the index of the element in the left subtree of $l[k]$ that maximizes $\hat{\sigma}(k)$, i.e., the index of the largest element in the left subtree.

We use the phrase *with high probability* to indicate when an event occurs with probability that tends towards 1 as $n \to \infty$. When an event occurs with probability of the form $1 - e^{-poly(n)}$, we say it occurs with *exponentially high probability*. During our analysis, we will make use of the following facts.

▶ **Lemma 2** (Poisson approximation (Corollary 5.9 in [16])). *Let $X_1^{(m)}, \ldots, X_n^{(m)}$ be the number of balls in each bin when m balls are thrown uniformly at random into n bins. Let $Y_1^{(m)}, \ldots, Y_n^{(m)}$ be independent Poisson random variables with $\lambda = m/n$. Then for any event $\varepsilon(x_1, \ldots, x_n)$:*

$$\Pr\left[\varepsilon\left(X_1^{(m)}, \ldots, X_n^{(m)}\right)\right] \leq e\sqrt{m}\Pr\left[\varepsilon\left(Y_1^{(m)}, \ldots, Y_n^{(m)}\right)\right].$$

▶ **Lemma 3** (Hoeffding's inequality (Theorem 2 in [11])). *If $X_1, \ldots, X_n$ are independent random variables and $a_k \leq X_k \leq b_k$ for $k = 1, \ldots, n$, then for $t > 0$:*

$$\Pr\left[\sum_k X_k - E\left[\sum_k X_k\right] \geq tn\right] \leq e^{-2n^2t^2/(\sum_k (b_k - a_k)^2)}.$$

3 Sorting Evolving Data with Repeated Insertion Sort

Let us begin with some simple bounds with respect to a single round of insertion sort.

▶ **Lemma 4.** *If a round of insertion sort starts at time t_s and finishes at time t_e, then*

1. *$t_e - t_s = F + n - 1$, where F is the number of inversions fixed (at the time of a comparison in the inner **while**-loop) by this round of insertion sort.*
2. *$t_e - t_s < n^2/2$*
3. *for any $t_s \leq t \leq t_e$, $I_t - I_{t_s} < n$.*

Proof. (1): For each iteration of the outer **for**-loop, each comparison in the inner **while**-loop either fixes an inversion (at the time of that comparison) or fails to fix an inversion and completes the inner **while**-loop. Note that this "failed" comparison may not have compared elements of l, but may have short circuited due to $j \leq 0$. Nevertheless, every comparison that doesn't fail fixes an inversion (at the time of that comparison); hence, each non-failing comparison is counted in F.

(2): In any round, there are at most $n(n-1)/2$ comparisons, by the formulations of the outer **for**-loop and inner **while**-loop.

(3): At time t, the round of insertion sort will have executed $t - t_s$ steps. Of those steps, at least $t - t_s - (n-1)$ comparisons resulted in a swap that removed an inversion and at most $n-1$ comparisons did not result in a change to l. The random swaps occurring during these comparisons introduced at most $t - t_s$ inversions. So $I_t - I_{t_s} \leq t - t_s - \big(t - t_s - (n-1)\big) = n - 1$. ◀

We next assert the following two lemmas, which are used in the next section.

▶ **Lemma 5.** *There exists a constant, $0 < \epsilon < 1$, such that, for a round of insertion sort that takes time t^*, at least ϵt^* of the random adjacent swaps in l' decrease I during the round, with exponentially high probability.*

Proof. Proof omitted due to space requirements. ◀

▶ **Lemma 6.** *If a round of insertion sort starts at time t_s with $I_{t_s} \geq (12c^2 + 2c)n$ and finishes at time t_e, then, with exponentially high probability, $t_e - t_s \geq cn$, i.e., the insertion sort round takes at least cn steps.*

Proof. See Section 4. ◀

3.1 Proof of Theorem 1

Armed with the above lemmas (albeit postponing the proofs of Lemma 5 and Lemma 6), let us prove our main theorem.

Theorem 1. *There exists a constant, $0 < \epsilon < 1$, such that, when running the repeated insertion-sort algorithm, for every step $t > (1 + 1/\epsilon)n^2$, the Kendall tau distance between the maintained list, l_t, and the underlying ordered list, l'_t, is $O(n)$, with exponentially high probability.*

Proof. By Lemma 5, there exists a constant $0 < \epsilon < 1$ such that at least an ϵ fraction of all of the random swaps during a round of insertion sort fix inversions. Consider an epoch of the last $(1 + 1/\epsilon)n^2$ steps of the repeated insertion-sort algorithm, that is, from time $t' = t - (1 + 1/\epsilon)n^2$ to t. During this epoch, some number, $m \geq 1$, of complete rounds of insertion sort are performed from start to end (by Lemma 4). Denote with t_k the time at which insertion-sort round k ends (and round $k+1$ begins), and let t_m denote the end time of the final complete round, during this epoch. By construction, observe that $t' \leq t_0$ and

$t_m \leq t$. Furthermore, because the insertion-sort rounds running before t_0 and after t_m take fewer than $n^2/2$ steps (by Lemma 4), $t_m - t_0 \geq n^2/\epsilon$.

The remainder of the proof consists of two parts. In the first part, we show that for some complete round of insertion sort ending at time $t_k \leq t$, I_{t_k} is $O(n)$, with exponentially high probability. In the second part, we show that once we achieve I_{t_k} being $O(n)$, for $t_k \leq t$, then I_t is $O(n)$, with exponentially high probability.

For the first part, suppose, for the sake of a contradiction, $I_{t_k} > \left(12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon}\right)n$, for all $0 \leq k \leq m$. Then, by a union bound over the polynomial number of rounds, Lemma 6 applies to every such round of insertion sort. So, with exponentially high probability, each round takes at least n/ϵ steps. Moreover, by Lemma 5, with exponential probability, an ϵ fraction of the random swaps from t_m to t_0 will decrease the number of inversions. That is, these random swaps increase the number of inversions by at most

$$(1-\epsilon)(t_m - t_0) - \epsilon(t_m - t_0) = (1-2\epsilon)(t_m - t_0),$$

with exponentially high probability. Furthermore, by Lemma 4, at least a $\frac{(1/\epsilon)-1}{1/\epsilon} = 1-\epsilon$ fraction of the insertion-sort steps fix inversions (at the time of a comparison). Therefore, with exponentially high probability, we have the following:

$$\begin{aligned} I_{t_m} &\leq I_{t_0} - (1-\epsilon)(t_m - t_0) + (1-2\epsilon)(t_m - t_0) \\ &= I_{t_0} - \epsilon(t_m - t_0) \\ &\leq I_{t_0} - n^2. \end{aligned}$$

But, since $I_{t_0} < n^2$, the above bound implies that $I_{t_m} < 0$, which is a contradiction. Therefore, with exponentially high probability, there is a $k \leq m$ such that $I_{t_k} \leq (12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon})n$.

For the second part, we show that the probability for a round $\ell > k$ to have $I_{t_\ell} > (12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon} + 1)n$ is exponentially small, by considering two cases (and their implied union-bound argument):

- If $I_{t_{\ell-1}} \leq (12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon})n$, then Lemma 4 implies $I_{t_\ell} \leq (12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon} + 1)n$.
- If $(12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon})n \leq I_{t_{\ell-1}} \leq (12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon} + 1)n$, then, similar to the argument given above, during a round of insertion sort, ℓ, at least a $1-\epsilon$ fraction of the steps fix an inversion, and an ϵ fraction of the steps do nothing. Also at least an ϵ fraction of the random swaps fix inversions, while a $1-\epsilon$ fraction add inversions. Finally, the total length of the round is $t_\ell - t_{\ell-1}$. Thus, with exponentially high probability, the total change in inversions is at most $-\epsilon(t_\ell - t_{\ell-1})$ and $I_{t_\ell} < I_{t_{\ell-1}}$.

Therefore, by a union bound over the polynomial number of insertion-sort rounds, the probability that any $I_{t_\ell} > (12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon} + 1)n$ for $k < \ell \leq m$ is exponentially small. By Lemma 4, $I_t \leq I_{t_m} + n$. So, with exponentially high probability, $I_{t_m} \leq (12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon} + 1)n = O(n)$ and $I_t = O(n)$, completing the proof. ◀

3.2 Improved Convergence Rate

In this subsection, we provide an algorithm that converges to $O(n)$ inversions more quickly. To achieve the steady state of $O(n)$ inversions, repeated insertion sort performs $\Theta(n^2)$ comparisons. But this running time to reach a steady state is a worst-case based on the fact that the running time of insertion sort is $O(n+I)$, where I is the number of initial inversions in the list, and, in the worst case, I is $\Theta(n^2)$. By simply running a round of quicksort on l first, we can achieve a steady state of $O(n)$ inversions after just $\Theta(n \log n)$ comparisons. See Algorithm 2. That is, we have the following.

Algorithm 2 Quicksort followed by repeated insertion sort pseudocode

```
function QUICK_THEN_INSERTION_SORT(l)
    quicksort(l)
    while true do
        for i ← 1 to n − 1 do
            j ← i
            while j > 0 and l[j] < l[j − 1] do
                swap l[j] and l[j − 1]
                j ← j − 1
```

▶ **Theorem 7.** *When running Algorithm 2, for every $t = \Omega(n \log n)$, I_t is $O(n)$ with high probability.*

Proof. By the results of Anagnostopoulos *et al.* [1], the initial round of quicksort takes $\Theta(n \log n)$ comparisons and afterwards the number of inversions (that is, the Kendall tau distance between the maintained list and the true total order) is $O(n \log n)$, with high probability. Using a nearly identical argument to the proof of Theorem 1, and the fact that an insertion-sort round takes $O(I + n)$ time to resolve I inversions, the repeated insertion-sort algorithm will, with high probability, achieve $O(n)$ inversions in an additional $O(n \log n)$ steps. From that point on, it will maintain a Kendall tau distance of $O(n)$, with high probability. ◀

4 Proof of Lemma 6

Recall Lemma 6, which establishes a lower bound for the running time of an insertion-sort round, given a sufficiently large amount of inversions relative to the underlying total order.

Lemma 6. *If a round of insertion sort starts at time t_s with $I_{t_s} \geq (12c^2 + 2c)n$ and finishes at time t_e, then, with exponentially high probability, $t_e - t_s \geq cn$, i.e., the insertion sort round takes at least cn steps.*

The main difficulty in proving Lemma 6 is understanding how the adjacent random swaps in l' affect the runtime of the current round of insertion sort on l. Let S_t be the number of steps left to perform in the current round of insertion sort if there were no more random adjacent swaps in l'. In essence, S can be thought of as an estimate of the remaining time in the current insertion sort round. If a new round of insertion sort is started at time t_s, then $S_{t_s - 1} = 1$ and $I_{t_s} \leq S_{t_s} \leq I_{t_s} + n - 1$. Each step of an insertion sort round decreases S by one and the following random swap may increase or decrease S by some amount. Figure 2 illustrates an example where one random adjacent swap in l' decreases S by a non-constant amount (relative to n).

A random adjacent swap in l' involving two elements in the unsorted portion of l will either increase or decrease S by one depending on whether it introduces or removes an inversion. Random adjacent swaps involving elements in the semi-sorted portion have more complex effects on S.

An inversion currently in the list $\big(l[a], l[b]\big)$ will be fixed by insertion sort if $l[a]$ and $l[b]$ will be compared and the two are swapped. Because $a < b$, $l[b]$ must be the active element during this comparison. An inversion $\big(l[a], l[b]\big)$ will not be fixed by insertion sort if $l[b]$ was already inserted into the semi-sorted portion or there is some element $l[c]$ in the semi-sorted portion with $a < c < b$ and $\sigma(c) < \sigma(b)$. We call an inversion with $l[b]$ in the semi-sorted

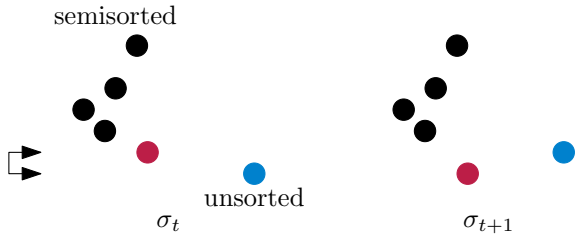


Figure 2 An example where swapping the ordering of the red and blue elements in l' creates multiple blocked inversions between the blue element and the black elements. Recall that our list is partitioned into the semisorted region, which contains elements that have already been compared in this round, and the unsorted region.

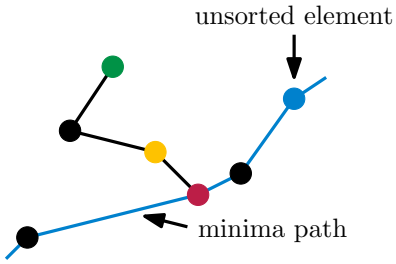


Figure 3 In this Cartesian tree, the green-blue pair is a blocked inversion and the green-yellow pair is a stuck inversion. Both pairs of inversions blame the red element.

portion a *stuck* inversion and an inversion with a smaller semi-sorted element between the pair a *blocked* inversion. We say an element $l[c]$ in the semi-sorted portion of l *blocks* an inversion $\big(l[a], l[b]\big)$ with $a \leq i$ and $l[b]$ either the active element or in the unsorted portion of l, if $l[c]$ is in the semi-sorted portion of l with $a < c < b$ and $\sigma(c) < \sigma(b)$. Note that there may be multiple elements blocking a particular inversion. Figure 3 shows examples of these two types of inversions.

We denote the number of "bad" inversions at time t that will not be fixed with B_t. That is, B_t is the sum of the blocked and stuck inversions. At the end of an insertion-sort round every inversion present at the start was either fixed by the insertion sort, fixed by a random adjacent swap in l', or is currently stuck. No elements can be blocked at the end of an insertion-sort round, because the semi-sorted portion is the entire list. Stuck inversions are either created by random adjacent swaps in l' or were blocked inversions and insertion sort finished inserting the right element of the pair. Blocked inversions are only introduced by the random adjacent swaps in l'. Thus B_t is unaffected by the steps of insertion sort.

Every inversion present at the start must be fixed by a step of insertion sort, be fixed by a random swap, or it will end up "bad". Therefore, for any given time, t, by using naive upper bounds based on the facts that every insertion sort step can fix an inversion and every random adjacent swap can remove an inversion, we can immediately derive the following:

▶ **Lemma 8.** *For an insertion sort round that starts at time t_s and ends at time t_e, if $t_s \leq t \leq t_e$, then $S_t \geq I_{t_s} - 2(t - t_s) - B_t$.*

Since, when an insertion sort round finishes, $S_{t_e - 1} = 1$, Lemma 8 implies $2(t_e - t_s - 1) + B_{t_e} + 1 \geq I_{t_s}$. If we understand how B changes with each random adjacent swap in l', then we can bound how long insertion sort needs to run for this inequality to be true.

We associate the blocked and stuck inversions with elements that we say are *blamed* for the inversions. A blocked inversion $\big(l[a], l[b]\big)$ blames the element $l[c]$ with $a < c < b$ and minimum $\sigma(c)$. Note than $l[c]$ is on the minima path of the modified Cartesian tree, and $l[a]$ is in the left subtree of $l[c]$. A stuck inversion either blames the element on the minima path whose subtree contains both $l[a]$ and $l[b]$ or if they appear in different subtrees, the inversion blames the element $l[c]$ with $a < c < b$ and minimum $\sigma(c)$. Again note that the blamed element is on the minima path and $l[a]$ is in the blamed element's left subtree. The bad inversions in Figure 3 blame the red element.

Whether stuck or blocked, every inversion blames an element on the minima path and the left element of the inverted pair appears in that minimum's subtree. If $l[k]$ is on the minima path, $M(k)$ is the index of the element in $l[k]$'s subtree with maximum $\sigma(M(k))$, and an inversion $\big(l[a], l[b]\big)$ has $l[a]$ in $l[k]$'s subtree, then both $l[a]$ and $l[b]$ are in the range $\sigma(k)$ to $\sigma(M(k))$. So we can upper bound B_t by $\sum_{k=0}^{n-1}(\sigma(M(k)) - \sigma(k))^2$, where we extend M to non-minima indices with $M(k) = k$ if k is not the index of a minima in l.

4.1 Bounding the Number of Blocked and Stuck Inversions with Counters

For the purposes of bounding B_t, we conceptually associate two counters, $Inc(x)$ and $Dec(x)$, with each element, x. The counters are initialized to zero at the start of an insertion sort round. When an element x is increased by a random swap in l', we increment $Inc(x)$ and when x is decreased by a random swap in l', we increment $Dec(x)$. After the random swap occurs, we may choose to exchange some of the counters between pairs of elements, but we will always maintain the following invariant:

Invariant 1. *For an element, $l[k]$, on the minima path,*

$$Inc\big(l[M(k)]\big) + Dec\big(l[k]\big) \geq \sigma\big(M(k)\big) - \sigma(k).$$

This invariant allows us to prove the following Lemma:

▶ **Lemma 9.** *If $\sum_{k=0}^{n-1} Inc\big(l[k]\big)^2 < \kappa$ and $\sum_{k=0}^{n-1} Dec\big(l[k]\big)^2 < \kappa$, then $B_t \leq 4\kappa$.*

Proof.

$$\begin{aligned} B_t &\leq \sum_{k=0}^{n-1} \Big(\sigma(M(k)) - \sigma(k)\Big)^2 \\ &\leq \sum_{k=0}^{n-1} \Big(Inc\big(M(k)\big) + Dec(k)\Big)^2 \qquad \text{By Invariant 1} \qquad (1) \end{aligned}$$

By the assumptions of this lemma, interpreting Inc and Dec as two n-dimensional vectors, we know their lengths are both less than $\sqrt{\kappa}$. Equation 1 is the squared length of the sum of the Dec and Inc vectors with the entries of Inc permuted by the function M. By the triangle inequality, the length of their sum is at most $2\sqrt{\kappa}$ and so the squared length of their sum is at most 4κ.

Therefore, $B_t \leq 4\kappa$. ◀

▶ **Lemma 10.** *There is a counter maintenance strategy that maintains Invariant 1 such that after each random adjacent swap in l', the corresponding counters are incremented and then some counters are exchanged between pairs of elements.*

Proof. Proof omitted due to space requirements. ◀

4.2 Bounding the Counters with Balls and Bins

We model the *Inc* and *Dec* counters each with a balls and bins process and analyze the sum of squares of balls in each bin. Each element in l is associated with one of n bins. When an element's *Inc* counter is increased, throw a ball into the corresponding bin. If a pair of *Inc* counters are exchanged, exchange the set of balls in the two corresponding bins. The *Dec* counters can be modeled similarly.

This process is almost identical to throwing balls into n bins uniformly at random. Note that the exchanging of balls in pairs of bins takes place after a ball has been placed in a chosen bin, effectively permuting two bin labels in between steps. If every bin was equally likely to be hit at each time step, then permuting the bin labels in this way would not change the final sum of squares and the exchanging of counters could be ignored entirely. Unfortunately the bin for the element at $l[n-1]$ in the case of *Inc* counters or $l[0]$ in the case of *Dec* counters cannot be hit, i.e., there is a forbidden bin controlled by the counter swapping strategy. However, even when in each round the forbidden bin is adversarially chosen, the sum of squares of the number of balls in each bin will be stochastically dominated by a strategy of always forbidding the bin with the lowest number of balls. Therefore, the sum of squares of m balls being thrown uniformly at random into $n-1$ bins stochastically dominates the sum of squares of the *Inc* (or *Dec*) counters after m steps.

▶ **Theorem 11.** *If cn balls are each thrown uniformly at random into n bins with $c > e$, then the sum over the bins of the square of the number of balls in each bin is at most $3c^2n$ with exponentially high probability.*

Proof. Let $X_1, \ldots, X_n$ be random variables where X_k is the number of balls in bin k and let $Y_1, \ldots, Y_n$ be independent Poisson random variables with $\lambda = c$.

By the Poisson approximation, Lemma 2,

$$\Pr\left[\sum_k X_k^2 \geq 3c^2n\right] \leq e\sqrt{cn}\Pr\left[\sum_k Y_k^2 \geq 3c^2n\right].$$

Let Z_k be the event that $Y_k \geq ecn^{1/6}$ and Z be the event that at least one Z_k occurs.

$$\Pr[Z] \leq n\Pr[Z_1] \quad \text{by a union bound.}$$

$$\begin{aligned}\Pr[Z_1] &= e^{-c}\sum_{k=ecn^{1/6}}^{\infty}\frac{c^k}{k!} \leq e^{-c}\sum_{k=ecn^{1/6}}^{\infty}\frac{c^k}{e\left(\frac{k}{e}\right)^k}\\ &= e^{-c-1}\sum_{k=ecn^{1/6}}^{\infty}\left(\frac{ec}{k}\right)^k \leq e^{-c-1}\sum_{k=ecn^{1/6}}^{\infty}\left(\frac{1}{n^{1/6}}\right)^k\\ &= e^{-c-1}(n^{1/6})^{-ecn^{1/6}}\sum_{k=0}^{\infty}\frac{1}{n^{1/6}}^k \leq e^{-c}n^{-\frac{ec}{6}n^{1/6}}.\end{aligned}$$

$$\Rightarrow \Pr[Z] \leq \frac{n}{e^c n^{\frac{ec}{6}n^{1/6}}} \leq e^{-\Omega(n^{1/6})}.$$

Letting $Y = \sum_k Y_k^2$:

$$E[Y|\neg Z] \leq E[Y] = nE[Y_1^2] = n\left(c + c^2\right) \leq 2c^2n.$$

Given $\neg Z$, $(Y_k)^2 \in [0, ecn^{1/3}]$. So we can apply Hoeffding's inequality, Lemma 3, to get:

$$\Pr\left[Y - E\left[Y|\neg Z\right] \geq tn|\neg Z\right] \leq e^{-2t^2n^2/\left(n\left(ecn^{1/3}\right)^2\right)}.$$

Setting $t = c^2$, we have:

$$\begin{aligned}\Pr\left[Y - E\left[Y|\neg Z\right] \geq c^2 n|\neg Z\right] &\leq e^{\left(-2c^4n^2\right)/\left(n\left(ecn^{1/3}\right)^2\right)} \\ &\leq e^{-2n^{1/3}}.\end{aligned}$$

Because $E\left[Y|\neg Z\right] \leq 2c^2 n$, we have $\Pr[Y \geq 3c^2 n|\neg Z] \leq e^{-\Omega(n^{1/3})}$.

$$\begin{aligned}\Pr\left[Y \geq 3c^2 n\right] &= \Pr\left[Y \leq 3c^2 n \text{ and } Z\right] + \Pr\left[Y \leq c^2 n \text{ and } \neg Z\right] \\ &\leq \Pr[Z] + \Pr\left[Y \leq 3c^2 n|\neg Z\right] \\ &\leq e^{-\Omega(n^{1/6})} + \Pr[Y - E[Y|\neg Z] \geq c^2 n|\neg Z] \\ &\leq e^{-\Omega(n^{1/6})} + e^{-\Omega(n^{1/3})} \leq 2e^{-\Omega(n^{1/6})}.\end{aligned}$$

Thus, we can conclude $\Pr[\sum_k X_k^2 \geq 3c^2 n] \leq \frac{2e\sqrt{cn}}{e^{\Omega(n^{1/6})}} \leq e^{-\,poly(n)}$. ◀

Recall that by Lemma 8, if an insertion-sort round ends at time t, then $I_{t_s} \leq 2(t - t_s) + B_t + 1$. Theorem 11 and a simple union bound tell us that if $t \leq t_s + cn$, then $\sum_{k=0}^{n-1} Inc\big(l[k]\big)^2 \leq 3c^2(n-1)$ and $\sum_{k=0}^{n-1} Dec\big(l[k]\big)^2 \leq 3c^2(n-1)$ with exponentially high probability. So by Lemma 9, $B_t \leq 12c^2 n$.

Recall that when the insertion sort round finishes, $2(t_e - t_s - 1) + B_{t_e} + 1 \geq I_{t_s}$. If fewer than cn steps have been performed, the left hand side of this inequality is less than $(12c^2+2c)n$ with exponentially high probability. Therefore, if we started with $(12c^2 + 2c)n$ inversions, the current round of insertion sort must perform at least cn steps with exponentially high probability; otherwise, there are unfixed but still "good" inversions. This completes the proof of Lemma 6.

5 Conclusion

We have shown that, although it is much simpler than quicksort and only fixes at most one inversion in each step, repeated insertion sort leads to the asymptotically optimal number of inversions in the evolving data model. We have also shown that by using a single round of quicksort before our repeated insertion sort, we can get to this steady state after an initial phase of $O(n \log n)$ steps, which is also asymptotically optimal.

For future work, it would be interesting to explore whether our results can be composed with other problems involving algorithms for evolving data, where sorting is a subcomponent. In addition, our analysis in this paper is specific to insertion sort, and only applies when exactly one random swap is performed after each comparison. We would like to extend this to other sorting algorithms that have been shown to perform well in practice and to the case in which the number of random swaps per comparison is a larger constant. Finally, it would also be interesting to explore whether one can derive a much better ϵ value than we derived in the proof of Lemma 5.

References

1 Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. Sorting and selection on dynamic data. *Theoretical Computer Science*, 412(24):2564–2576, 2011. Special issue on selected papers from 36th International Colloquium on Automata, Languages and Programming (ICALP 2009). doi:10.1016/j.tcs.2010.10.003.

2 Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, Eli Upfal, and Fabio Vandin. Algorithms on evolving graphs. In *3rd ACM Innovations in Theoretical Computer Science Conference (ITCS)*, pages 149–160, 2012. doi:10.1145/2090236.2090249.

3 Bahman Bahmani, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. Pagerank on an evolving graph. In *18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 24–32, 2012. doi:10.1145/2339530.2339539.

4 Juan Jose Besa Vial, William E. Devanny, David Eppstein, Michael T. Goodrich, and Timothy Johnson. Quadratic time algorithms appear to be optimal for sorting evolving data. In *Proc. Algorithm Engineering & Experiments (ALENEX 2018)*, pages 87–96, 2018. doi:10.1137/1.9781611975055.8.

5 Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In *19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 268–276, 2008.

6 Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

7 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994. doi:10.1137/S0097539791195877.

8 Michael T. Goodrich and Roberto Tamassia. *Algorithm Design and Applications*. Wiley Publishing, 1st edition, 2014.

9 Benoit Groz and Tova Milo. Skyline queries with noisy comparisons. In *34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 185–198, 2015. doi:10.1145/2745754.2745775.

10 Dorit S. Hochbaum. Ranking sports teams and the inverse equal paths problem. In Paul Spirakis, Marios Mavronicolas, and Spyros Kontogiannis, editors, *2nd Int. Workshop on Internet and Network Economics (WINE)*, volume 4286 of *Lecture Notes in Computer Science*, pages 307–318, Berlin, Heidelberg, 2006. Springer. doi:10.1007/11944874_28.

11 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. doi:10.1080/01621459.1963.10500830.

12 Qin Huang, Xingwu Liu, Xiaoming Sun, and Jialin Zhang. Partial sorting problem on evolving data. *Algorithmica*, 79(3):1–24, 2017. doi:10.1007/s00453-017-0295-3.

13 Varun Kanade, Nikos Leonardos, and Frédéric Magniez. Stable Matching with Evolving Preferences. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 60 of *LIPIcs*, pages 36:1–36:13, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2016.36.

14 Donald Ervin Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Pearson Education, 2nd edition, 1998.

15 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Sorting noisy data with partial information. In *4th ACM Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 515–528, 2013. doi:10.1145/2422436.2422492.

16 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

17 Jean Vuillemin. A unifying look at data structures. *Commun. ACM*, 23(4):229–239, 1980. doi:10.1145/358841.358852.

18 Jialin Zhang and Qiang Li. Shortest paths on evolving graphs. In H. Nguyen and V. Snasel, editors, *5th Int. Conf. on Computational Social Networks (CSoNet)*, volume 9795 of *Lecture Notes in Computer Science*, pages 1–13, Berlin, Heidelberg, 2016. Springer. doi:10.1007/978-3-319-42345-6_1.

Generalized Comparison Trees for Point-Location Problems

Daniel M. Kane[1]
Department of Computer Science and Engineering/Department of Mathematics, University of California, San Diego
dakane@ucsd.edu
https://orcid.org/0000-0002-5884-3487

Shachar Lovett[2]
Department of Computer Science and Engineering, University of California, San Diego
slovett@cs.ucsd.edu
https://orcid.org/0000-0003-4552-1443

Shay Moran[3]
Institute for Advanced Study, Princeton
shaymoran@ias.edu
https://orcid.org/0000-0002-8662-2737

Abstract

Let H be an arbitrary family of hyper-planes in d-dimensions. We show that the point-location problem for H can be solved by a linear decision tree that only uses a special type of queries called *generalized comparison queries*. These queries correspond to hyperplanes that can be written as a linear combination of two hyperplanes from H; in particular, if all hyperplanes in H are k-sparse then generalized comparisons are $2k$-sparse. The depth of the obtained linear decision tree is polynomial in d and logarithmic in $|H|$, which is comparable to previous results in the literature that use general linear queries.

This extends the study of comparison trees from a previous work by the authors [Kane et al., FOCS 2017]. The main benefit is that using generalized comparison queries allows to overcome limitations that apply for the more restricted type of comparison queries.

Our analysis combines a seminal result of Forster regarding sets in isotropic position [Forster, JCSS 2002], the margin-based inference dimension analysis for comparison queries from [Kane et al., FOCS 2017], and compactness arguments.

2012 ACM Subject Classification Theory of computation

Keywords and phrases linear decision trees, comparison queries, point location problems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.82

Related Version A full version of the paper is available at [8], `https://eccc.weizmann.ac.il/report/2017/082`.

[1] Supported by NSF CAREER Award ID 1553288 and a Sloan fellowship
[2] Research supported by NSF CAREER award 1350481, CCF award 1614023 and a Sloan fellowship
[3] Research supported by the National Science Foundation under agreement No. CCF-1412958 and by the Simons Foundations

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 82; pp. 82:1–82:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Let $H \subset \mathbb{R}^d$ be a family of $|H| = n$ hyper-planes. H partitions $\mathbb{R}^d$ into $O(n^d)$ cells. The *point-location* problem is to decide, given an input point $x \in \mathbb{R}^d$, to which cell it belongs. That is, to compute the function

$$\mathcal{A}_H(x) := (\text{sign}(\langle x, h\rangle) : h \in H) \in \{-1, 0, 1\}^n.$$

A well-studied computation model for this problem is a *linear decision tree* (LDT): this is a ternary decision tree whose input is $x \in \mathbb{R}^d$ and its internal nodes v make linear/threshold queries of the form $\text{sign}(\langle x, q\rangle)$ for some $q = q(v) \in \mathbb{R}^d$. The three children of v correspond to the three possible outputs of the query : "$-$","0","$+$". The leaves of the tree are labeled with $\{-1, 0, 1\}^n$ with correspondence to the cell in the arrangement that contains x. The complexity of a linear decision tree is its depth, which corresponds to the maximal number of linear queries made on any input.

Comparison queries

A *comparison decision tree* is a special type of an LDT, in which all queries are of one of two types:

- Label query: "sign $(\langle x, h\rangle) = ?$" for $h \in H$.
- Comparison query: "sign $(\langle x, h' - h''\rangle) = ?$" for $h', h'' \in H$.

In [6] it is shown that when H is "nice" then there exist comparison decision trees that computed $\mathcal{A}_H(\cdot)$ and has nearly optimal depth (up to logarithmic factors). For example, for any $H \subset \{-1, 0, 1\}^d$ there is a comparison decision tree with depth $O(d \log d \log|H|)$. This is off by a $\log d$ factor from the basic information theoretical lower bound of $\Omega(d \log|H|)$. Moreover, it is shown there that certain niceness conditions are necessary. Concretely, they give an example of $H \subset \mathbb{R}^3$ such that any comparison decision tree that computes $\mathcal{A}_H(\cdot)$ requires depth $\Omega(|H|)$. This raises the following natural problem: can comparison decision trees be generalized in a way that allows to handle arbitrary point-location problems?

Generalized comparisons

This paper addresses the above question by considering generalized comparison queries. A generalized comparison query allows to re-weight its terms: namely, it is query of the form

"sign $(\langle x, \alpha h' - \beta h''\rangle) =?$"

for $h', h'' \in H$ and some $\alpha, \beta \in \mathbb{R}$. Note that it may be assumed without loss of generality that $|\alpha| + |\beta| = 1$. A generalized comparison decision tree, naturally, is a linear decision tree whose internal linear queries are restricted to be generalized comparisons. Note that generalized comparison queries include as special cases both label queries (setting $\alpha = 1, \beta = 0$) and comparison queries (setting $\alpha = \beta = 1/2$).

Geometrically, generalized comparisons are 1-dimensional in the following sense: let $q = \alpha h' - \beta h''$, with $\alpha, \beta \geq 0$ then q lies on the interval connecting h' and $-h''$. If α and β have different signs, q lies on an interval between some other $\pm h'$ and $\pm h''$. So comparison queries are linear queries that lies on the projective lines intervals spanned by $\{\pm h : h \in H\}$. In particular, if each $h \in H$ has sparsity at most k (namely, at most k nonzero coordinates) then each generalized comparison has sparsity at most $2k$.

Our main result is:

▶ **Theorem 1** (Main theorem). *Let $H \subset \mathbb{R}^d$. Then there exists a generalized comparison decision tree of depth $O(d^4 \log d \log |H|)$ that computes $\mathcal{A}_H(x)$ for every input $x \in \mathbb{R}^d$.*

Why consider generalized comparisons?

We consider generalized comparisons for a number of reasons:

- The lower bound against comparison queries in [6] was achieved by essentially scaling different elements of $H \subset \mathbb{R}^3$ with exponentially different scales. Allowing for re-scaling (which is what generalized comparisons allow to do) solves this problem.
- Generalized comparisons may be natural from a machine learning perspective, in particular in the context of active learning. A common type of queries used in practice it to give a score to an example (say 1-10), and not just label it as positive (+) or negative (-). Comparing the scores for different examples can be viewed as a "coarse" type of generalized comparisons.
- If the set of original hyperplanes H is "nice", then generalized comparisons maintain some aspects of niceness in the queries performed. As an example that was already mentioned, if all hyperplanes in H are k-sparse then generalized comparisons are $2k$-sparse. This is part of a more general line of research, studying what types of "simple queries" are sufficient to obtain efficient active learning algorithms, or equivalently efficient linear decision trees for point-location problems.

1.1 Proof outline

Our proof consists of two parts. First, we focus on the case when $H \subset \mathbb{R}^d$ is in *general position*, namely, every d vectors in it are linearly independent. Then, we extend the construction to arbitrary H. The second part is derived via standard compactness arguments; it is omitted from here and appears in the full version [8]. The technical crux lies in the first part: let $H \subseteq \mathbb{R}^d$ be in general position; we first construct a randomized generalized comparison decision tree for H, and then derandomize it. The randomized tree is simple to describe: it proceeds by steps, where in each step about d^2 elements from H are drawn, labelled, and sorted using generalized comparisons. Then, it is shown that the labels of some $1/d$-fraction of the remaining elements in H are inferred, on average. The inferred vectors are then removed from H and this step is repeated until all labels in H are inferred.

A central technical challenge lies in the analysis of a single step. It hinges on a result by Forster [4] that transforms a general-positioned H to an *isotropic-positioned* H' (see formal definition below) in a way that comparison queries on H' correspond to generalized comparison queries on H. Then, since H' is in isotropic position, it follows that a significant fraction of H' has a large margin with respect to the input x. This allows us to employ a variant of the margin-based inference analysis by [6] on H' to derive the desired inference of some $\Omega(\frac{1}{d})$-fraction of the remaining labels in each step.

1.2 Related work

The point-location problem has been studied since the 1980s, starting from the pioneering work of Meyer auf der Heide [10], Meiser [9], Cardinal et al. [2] and most recently Ezra and Sharir [3]. This last work, although not formally stated as such, solves the point-location problem for an arbitrary $H \subset \mathbb{R}^d$ by a linear decision tree whose depth is $O(d^2 \log d \log |H|)$. However, in order to do so, the linear queries used by the linear decision tree could be arbitrary, even when the original family H is very simple (say 3-sparse). This is true for all previous works, as they are all based on various geometric partitioning ideas, which may

require the use of quite generic hyperplanes. This should be compared with our results (Theorem 1). We obtain a linear decision tree of a bigger depth (by a factor of d^2), however the type of linear queries we use remain relatively simple; e.g., as discussed earlier, they are 1-dimensional and preserve sparseness.

1.3 Open problems

Our work addresses a problem raised in [7], of whether "simple queries" can be sufficient to solve the point-location problem for general hyperplanes H, without making any "niceness" assumptions on H. The solution explored here is to allow for generalized comparisons, which are a 1-dimensional set of allowed queries. An intriguing question is whether this is necessary, or whether there are some 0-dimensional gadgets that would be sufficient.

In order to formally define the problem, we need the notion of *gadgets*. A t-ary gadget in $\mathbb{R}^d$ is a function $g : (\mathbb{R}^d)^t \to \mathbb{R}^d$. Let $G = \{g_1, \ldots, g_r\}$ be a finite collection of gadgets in $\mathbb{R}^d$. Given a set of hyperplanes $H \subset \mathbb{R}^d$, a G-LDT that solves $\mathcal{A}_H(\cdot)$ is a LDT where any linear query is of the form $\text{sign}(\langle q, \cdot \rangle)$ for $q = g(h_1, \ldots, h_t)$ for some $g \in G$ and $h_1, \ldots, h_t \in H$. For example, a comparison decision tree corresponds to the gadgets $g_1(h) = h$ (label queries) and $g_2(h_1, h_2) = h_1 - h_2$ (comparison queries). A generalized comparison decision tree corresponds to the 1-dimensional (infinite) family of gadgets $\{g_\alpha(h_1, h_2) = \alpha h_1 - (1-\alpha)h_2 : \alpha \in [0, 1]\}$. It was shown in [6] that comparison decision trees are sufficient to efficiently solve the point-location problem in 2 dimensions, but not in 3 dimensions. So, the problem is already open in $\mathbb{R}^3$.

▶ **Open problem 1.** *Fix $d \geq 3$. Is there a finite set of gadgets G in $\mathbb{R}^d$, such that for every $H \subset \mathbb{R}^d$ there exists a G-LDT which computes $\mathcal{A}_H(\cdot)$, whose depth is logarithmic in $|H|$? Can one hope to get to the information theoretic lower bound, namely to $O(d \log |H|)$?*

Another open problem is whether randomized LDT can always be derandomized, without losing too much in the depth. To recall, a randomized (zero-error) LDT is a distribution over (deterministic) LDTs which each computes $\mathcal{A}_H(\cdot)$. The measure of complexity for a randomized LDT is the expected number of queries performed, for the worst-case input x. The derandomization technique we apply in this work (see Lemma 16 and its proof for details) loses a factor of d, but it is not clear whether this loss is necessary.

▶ **Open problem 2.** *Let $H \subset \mathbb{R}^d$. Assume that there exists a randomized LDT which computes $\mathcal{A}_H(\cdot)$, whose expected query complexity is at most D for any input. Does there always exist a (deterministic) LDT which computes $\mathcal{A}_H(\cdot)$, whose depth is $O(D)$?*

2 Preliminaries and some basic technical lemmas

2.1 Inferring from comparisons

Let $x, h \in \mathbb{R}^d$ and let $S \subseteq \mathbb{R}^d$.

▶ **Definition 2** (Inference)**.** We say that S infers h at x if $\text{sign}(\langle h, x \rangle)$ is determined by the linear queries $\text{sign}(\langle h', x \rangle)$ for $h' \in S$. That is, if for any point y in the set

$$\left\{ y \in \mathbb{R}^d : \text{sign}(\langle h', y \rangle) = \text{sign}(\langle h', x \rangle) \quad \forall h' \in S \right\}$$

it holds that $\text{sign}(\langle h, y \rangle) = \text{sign}(\langle h, x \rangle)$. Define

$$\mathsf{infer}(S; x) := \{h \in \mathbb{R}^d : h \text{ is inferred from } S \text{ at } x\}.$$

The notion of inference has a natural geometric perspective. Consider the partition of $\mathbb{R}^d$ induced by S. Then, S infers h at x if the cell in this partition that contains x is either disjoint from h or otherwise is contained in h (so in either case, the value of $\text{sign}(\langle h, \cdot \rangle)$ is constant on the cell).

Our algorithms and analysis are based on inferences from comparisons. Let $S - S$ denote the set $\{h' - h'' : h', h'' \in S\}$.

▶ **Definition 3** (Inference by comparisons)**.** We say that *comparisons on* S infer h at x if $S \cup (S - S)$ infers h at x. Define

$$\mathsf{InferComp}(S; x) := \mathsf{infer}\big(S \cup (S - S); x\big).$$

Thus, $\mathsf{InferComp}(S; x)$ is determined by querying $\text{sign}(\langle h', x \rangle)$ and $\text{sign}(\langle h' - h'', x \rangle)$ for all $h', h'' \in S$. Naively, this requires some $O(|S|^2)$ linear queries. However, using efficient sorting algorithm (e.g. merge-sort) achieves it with just $O(|S| \log |S|)$ comparison queries. A further improvement, when $|S| > d$, is obtained by Fredman's sorting algorithm that uses just $O(|S| + d \log |S|)$ comparison queries [5].

2.2 Vectors in isotropic position

Vectors $h_1, \ldots, h_m \in \mathbb{R}^d$ are said to be in *general position* if any d of them are linearly independent. They are said to be in *isotropic position* if for any unit vectors $v \in S^d$,

$$\frac{1}{m} \sum_{i=1}^{m} \langle h_i, v \rangle^2 = \frac{1}{d}.$$

Equivalently, if $\frac{1}{m} \sum h_i h_i^T$ is $\frac{1}{d}$ times the $d \times d$ identity matrix. An important theorem of Forster [4] (see also Barthe [1] for a more general statement) states that any set of vectors in general position can be scaled to be in isotropic position.

▶ **Theorem 4** ([4])**.** *Let $H \subset \mathbb{R}^d$ be a finite set in general position. Then there exists an invertible linear transformation T such that the set*

$$H' := \left\{ \frac{Th}{\|Th\|_2} : h \in H \right\}$$

is in isotropic position. We refer to such a T as a Forster transformation *for H.*

We will also need a relaxed notion of isotropic position. Given vectors $h_1, \ldots, h_m \in \mathbb{R}^d$ and some $0 < c < 1$, we say that the vectors are in *c-approximate isotropic position*, if for all unit vectors $v \in S^d$ it holds that

$$\frac{1}{m} \sum_{i=1}^{m} \langle h_i, v \rangle^2 \geq \frac{c}{d}.$$

We note that this condition is easy to test algorithmically, as it is equivalent to the statement that the smallest eigenvalue of the positive semi-definite $d \times d$ matrix $\frac{1}{m} \sum_{i=1}^{m} h_i h_i^T$ is at least $\frac{c}{d}$.

We summarize it in the following lemma, which follows from basic real linear algebra.

▶ **Claim 5.** *Let $h_1, \ldots, h_m \in \mathbb{R}^d$ be unit vectors. Then the following are equivalent.*

- *$h_1, \ldots, h_m$ are in c-approximate isotropic position.*
- *$\lambda_1 \left(\frac{1}{m} \sum_{i=1}^{m} h_i h_i^T \right) \geq c/d$,*

where $\lambda_1(M)$ denotes the minimal eigenvalue of a positive semidefinite matrix M.

We will need the following basic claims. The first claim shows that a set of unit vectors in an approximate isotropic position has many vectors with non-negligible inner product with any unit vector.

▶ **Claim 6.** *Let $h_1, \ldots, h_m \in \mathbb{R}^d$ be unit vectors in a c-approximate isotropic position, and let $x \in \mathbb{R}^d$ be a unit vector. Then, at least a $\frac{c}{2d}$-fraction of the h_i's satisfy $|\langle h_i, x\rangle| > \sqrt{\frac{c}{2d}}$.*

Proof. Assume otherwise. It follows that

$$\frac{1}{m}\sum_{i=1}^{m}|\langle h, x_i\rangle|^2 \leq \frac{c}{2d}\cdot 1 + \left(1-\frac{c}{2d}\right)\frac{c}{2d} < \frac{c}{2d}+\frac{c}{2d} = \frac{c}{d}.$$

This contradicts the assumption that the h_i's are in c-approximate isotropic position. ◀

The second claim shows that a random subset of a set of unit vectors in an approximate isotropic position is also in approximate isotropic position, with good probability.

▶ **Claim 7.** *Let $h_1, \ldots, h_m$ be unit vectors in c-approximate isotropic position. Let $i_1, \ldots, i_k \in [m]$ be independently and uniformly sampled. Then for any $\delta > 0$, the vectors $h_{i_1}, \ldots, h_{i_k}$ are in $((1-\delta)c)$-approximate isotropic position with probability at least*

$$1 - d\cdot\left[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right]^{ck/d}.$$

Proof. This is an immediate corollary of Matrix Chernoff bounds [11]. By Claim 5 the above event is equivalent to that $\lambda_1\left(\frac{1}{k}\sum_{i=1}^{k} h_i h_i^T\right) \geq (1-\delta)\frac{c}{d}$. By assumption, $\lambda_1\left(\frac{1}{m}\sum_{i=1}^{m} h_i h_i^T\right) \geq \frac{c}{d}$. Now, by the Matrix Chernoff bound, for any $\delta \in [0,1]$ it holds that

$$\Pr\left[\lambda_1\left(\frac{1}{k}\sum_{i=1}^{k} h_i h_i^T\right) \leq (1-\delta)\cdot\frac{c}{d}\right] \leq d\cdot\left[\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right]^{ck/d}. \qquad ◀$$

We will use two instantiations of Claim 7: (i) $c \geq 3/4$, and $(1-\delta)c = 1/2$, and (ii) $c = 1$ and $(1-\delta)c = 3/4$. In both cases the bound simplifies to

$$1 - d\cdot\left(\frac{99}{100}\right)^{k/d}. \tag{1}$$

3 Proof of main theorem

Let $H \subset \mathbb{R}^d$. Theorem 1 is proved in three steps.

1. First, we assume that H is in general position. In this case, we construct a randomized generalized comparison LDT which computes $\mathcal{A}_H(\cdot)$, whose expected depth is $O(d^3 \log d \log |H|)$ for any input. This is achieved in Section 3.1, see Lemma 8.
2. Next, we derandomize the construction. This gives for any H in general position a (deterministic) generalized comparison LDT which computes $\mathcal{A}_H(\cdot)$, whose depth is $O(d^4 \log d \log |H|)$. This is achieved in Section 3.2, see Lemma 16.
3. Finally, we handle an arbitrary H (not necessarily in general position), and construct by a compactness argument a generalized comparisons LDT of depth $O(d^4 \log d \log |H|)$ which computes $\mathcal{A}_H(\cdot)$. This step is omitted from this exposition and appears in the full version [8].

3.1 A randomized LDT for H in general position

In this section we construct a randomized generalized comparison LDT for H in general position. Here, by a randomized LDT we mean a distribution over (deterministic) LDT which compute $\mathcal{A}_H(\cdot)$. The corresponding complexity measure is the expected number of queries it makes, for the worst-case input x.

▶ **Lemma 8.** *Let $H \subseteq \mathbb{R}^d$ be a finite set in general position. Then there exists a randomized LDT that computes $\mathcal{A}_H(\cdot)$, which makes $O\left(d^3 \log d \log|H|\right)$ generalized comparison queries on expectation, for any input.*

The proof of Lemma 8 is based on a variant of the margin-based analysis of the inference dimension with respect to comparison queries as in [6] (The analysis in [6] assumed that all vectors have large margin, where here we need to work under the weaker assumption that only a noticeable fraction of the vectors have large margin). The crux of the proof relies on scaling every $h \in H$ by a carefully chosen scalar α_h such that drawing a sufficiently large random subset of H, and sorting the values $\langle \alpha_h h, x\rangle$ using comparison queries (which correspond to generalized comparisons on the h's) allows to infer, on average, at least $\Omega(1/d)$ of the labels of H. The scalars α_h are derived via Forster's theorem (Theorem 4). More specifically, $\alpha_h = \frac{1}{\|Th\|_2}$, where T is a Forster transformation for H.

Randomized generalized-comparisons tree for H in general position

Let $H \subseteq \mathbb{R}^d$ in general position.

Input: $x \in \mathbb{R}^d$, given by oracle access for $\text{sign}(\langle \cdot, x\rangle)$
Output: $\mathcal{A}_H(x) = (\text{sign}(\langle h, x\rangle))_{h\in H}$
(1) Initialize: $H_0 = H$, $i = 0$, $v(h) =?$ for all $h \in H$. Set $k = \Theta(d^2 \log(d))$.
(2) Repeat while $|H_i| \geq k$:
  (2.1) Let T_i be the Forster transformation for H_i. Define $H_i' = \left\{\frac{h}{\|T_i h\|_2} : h \in H_i\right\}$.
  (2.2) Sample uniformly $S_i \subset H_i'$ of size $|S_i| = k$.
  (2.3) Query $\text{sign}(\langle h, x\rangle)$ for $h \in S_i$ (using label queries).
  (2.4) Sort $\langle h, x\rangle$ and $\langle -h, x\rangle$ for $h \in S_i$ (using generalized comparison queries).
  (2.5) For all $h \in H_i$, check if $h \in \mathsf{InferComp}(\pm S_i; x)$, and in case it is, set $v(h) \in \{-, 0, +\}$ to be the inferred value of h.
  (2.6) Remove all $h \in H_i$ for which $\text{sign}(\langle h, x\rangle)$ was inferred, set H_{i+1} to be the resulting set and go to step (2).
(3) Query $\text{sign}(\langle h, x\rangle)$ for all $h \in H_i$, and set $v(h)$ accordingly.
(4) Return v as the value of $\mathcal{A}_H(x)$.

In order to understand the intuition behind the main iteration (2) of the algorithm, define $x' = (T_i^{-1})^{\mathrm{T}} x$ and for each $h \in H_i$ let $h' = \frac{T_i h}{\|T_i h\|}$. Then $\text{sign}(\langle h, x\rangle) = \text{sign}(\langle h', x'\rangle)$, and so it suffices to infer the sign for many $h' \in H_i$ with respect to x'. The main benefit is that we may assume in the analysis that the set of vectors H_i' is in isotropic position; and reduce the analysis to that of using (standard) comparisons on H_i' and x'. These then translate to performing generalized comparison queries on H_i and the original input x. The following lemma captures the analysis of the main iteration of the algorithm. Below, we denote by $\pm S := S \cup (-S)$.

▶ **Lemma 9.** *Let $x \in \mathbb{R}^d$, let $H \subseteq \mathbb{R}^d$ be a finite set of unit vectors in c-approximate isotropic position with $c \geq 3/4$, and let $S \subset H$ be a uniformly chosen subset of size $k = \Omega\left(d^2 \log d\right)$. Then*

$$\mathbb{E}_S\left[\left|\mathsf{InferComp}(\pm S; x) \cap H\right|\right] \geq \frac{|H|}{40d}.$$

Let us first argue how Lemma 8 follows from Lemma 9, and then proceed to prove Lemma 9.

Proof of Lemma 8 given Lemma 9. By Lemma 9, in each iteration (2) of the algorithm, we infer on expectation at least $\Omega(1/d)$ fraction of the $h \in H'_i$ with respect to $x' = T_i^{-1}x$. By the discussion above, this is the same as inferring an $\Omega(1/d)$ fraction of the $h_i \in H_i$ with respect to x. So, the total expected number of iterations needed is $O(d \log |H|)$. Next, we calculate the number of linear queries performed at each iteration. The number of label queries is $O(k)$ and the number of comparison queries on H'_i (which translate to generalized comparison queries on H_i) is $O(k \log k)$ if we use merge-sort, and can be improved to $O(k + d \log k)$ by using Fredman's sorting algorithm [5]. So, in each iteration we perform $O(d^2 \log d)$ queries, and the expected number of iterations is $O(d \log |H|)$. So the expected total number of queries by the algorithm is $O(d^3 \log d \log |H|)$. ◀

From now on, we focus on proving Lemma 9. To this end, we assume from now that $H \subset \mathbb{R}^d$ is in c-isotropic position for $c \geq 3/4$. Note that h is inferred from comparisons on $\pm S$ if and only if $-h$ is, and that replacing an element of S with its negation does not affect $\pm S$. Therefore, negating elements of H does not change the expected number of elements inferred from comparisons on $\pm S$. Therefore, we may assume in the analysis that $\langle h, x \rangle \geq 0$ for all $h \in H$. Under this assumption, we will show that

$$\mathbb{E}_S\left[\left|\mathsf{InferComp}(S; x) \cap H\right|\right] \geq \frac{|H|}{40d}.$$

It is convenient to analyze the following procedure for sampling S:

- Sample $h_1, \ldots h_{k+1}$ random points in H, and $r \in [k+1]$ uniformly at random.
- Set $S = \{h_j : j \in [k+1] \setminus \{r\}\}$.

We will analyze the probability that comparisons on S infer h_r at x. Our proof relies on the following observation.

▶ **Observation 10.** *The probability, according to the above process, that $h_r \in \mathsf{InferComp}(S; x)$ is equal to the expected fraction of $h \in H$ whose label is inferred. That is,*

$$\Pr\left[h_r \in \mathsf{InferComp}(S; x)\right] = \mathbb{E}\left[\frac{\left|\mathsf{InferComp}(S; x) \cap H\right|}{|H|}\right].$$

Thus, it suffices to show that $\Pr\left[h_r \in \mathsf{InferComp}(S; x)\right] \geq 1/40d$. This is achieved by the next two propositions as follows. Proposition 11 shows that S is in a $(1/2)$-approximate isotropic position with probability at least $1/2$, and Proposition 12 shows that whenever S is in $(1/2)$-approximate isotropic position then $h_r \in \mathsf{InferComp}(S; x)$ with probability at least $1/20d$. Combining these two propositions together yields that $\Pr\left[h_r \in \mathsf{InferComp}(S; x)\right] \geq 1/40d$ and finishes the proof of Lemma 9.

▶ **Proposition 11.** *Let $H \subset \mathbb{R}^d$ be a set of unit vectors in c-approximate isotropic position for $c \geq 3/4$. Let $S \subset H$ be a uniformly sampled subset of size $|S| \geq \Omega(d \log d)$. Then S is in $(1/2)$-approximate isotropic position with probability at least $1/2$.*

Proof. The proof follows from Claim 7 by plugging $k = \Omega(d \log d)$ in Equation 1 and calculating that the bound on the right hand side becomes at least $1/2$. ◀

▶ **Proposition 12.** *Let $x \in \mathbb{R}^d$, $S \subset \mathbb{R}^d$ be in (1/2)-approximate isotropic position, where $|S| \geq \Omega\left(d^2 \log d\right)$. Let $h \in S$ be sampled uniformly. Then*

$$\Pr_{h \in S}\left[h \in \mathsf{InferComp}\left(S \setminus \{h\}; x\right)\right] \geq \frac{1}{20d}.$$

Proof. We may assume that x is a unit vector, namely $\|x\|_2 = 1$. Let $s = |S|$ and assume that $S = \{h_1, \ldots, h_s\}$ with

$$\langle h_1, x\rangle \geq \langle h_2, x\rangle \geq \ldots \geq \langle h_s, x\rangle \geq 0.$$

Set $\varepsilon = \frac{1}{2\sqrt{d}}$. As S is in $(1/2)$-approximate isotropic position, Claim 6 gives that $\langle h_i, x\rangle \geq \varepsilon$ for at least $|S|/4d$ many $h_i \in S$. Set $t = |S|/8d$ and define

$$T = \{h_1, \ldots, h_t\},$$

where by out assumption $\langle h_t, x\rangle \geq \varepsilon$. Note that in this case, we can compute T from comparison queries on S. We will show that

$$\Pr_{h \in T}\left[h \in \mathsf{InferComp}\left(S \setminus \{h\}; x\right)\right] \geq \frac{1}{2},$$

from which the proposition follows. This in turn follows by the following two claims, whose proof we present shortly.

▶ **Claim 13.** *Let $h_a \in T$. Assume that there exists a non-negative linear combination v of $\{h_i - h_{i+1} : i = 1, \ldots, a-2\}$ such that*

$$\|h_a - (h_1 + v)\|_2 \leq \varepsilon/4.$$

Then $h_a \in \mathsf{InferComp}\left(S \setminus \{h_a\}; x\right)$.

▶ **Claim 14.** *The assumption in Claim 13 holds for at least half the vectors in T.*

Clearly, Claim 13 and Claim 14 together imply that for at least half of $h_a \in T$, it holds that $h_a \in \mathsf{InferComp}\left(S \setminus \{h_a\}; x\right)$. This concludes the proof of the proposition. ◀

Next we prove Claim 13 and Claim 14.

Proof of Claim 13. Let $S' = S \setminus \{h_a\}$ and $T' = T \setminus \{h_a\}$. As S is in $(1/2)$-approximate isotropic position then S' is in c-approximate isotropic position for $c = 1/2 - d/|S|$. In particular, as $|S| \geq 4d$ we have $c \geq 1/4$. By applying comparison queries to S' we can sort $\{\langle h_i, x\rangle : h_i \in S'\}$. Then T' can be computed as the set of the $t-1$ elements with the largest inner product. Claim 6 applied to S' then implies that $\langle h_i, x\rangle \geq \varepsilon/2$ for all $h_i \in T'$. Crucially, we can deduce this just from the comparison queries on S', together with our initial assumption that S is in $(1/2)$-approximate isotropic position. Thus we deduced from our queries that:

- $\langle h_1, x\rangle \geq \varepsilon/2$.
- $\langle v, x\rangle \geq 0$.

In addition, from our assumption it follows that $|\langle h_a - (h_1 + v), x\rangle| \leq \varepsilon/4$. These together infer that $\langle h_a, x\rangle > 0$. ◀

The proof of Claim 14 follows from the applying the following claim iteratively. We note that this claim appears in [6] implicitly, but we repeat it here for clarity.

▶ **Claim 15.** *Let $h_1, \ldots, h_t \in \mathbb{R}^d$ be unit vectors. For any $\varepsilon > 0$, if $t \geq 16d\ln(2d/\varepsilon)$ then there exist $a \in [t]$ and $\alpha_1, \ldots, \alpha_{a-2} \in \{0,1,2\}$ such that*

$$h_a = h_1 + \sum_{j=1}^{i-2} \alpha_j (h_{j+1} - h_j) + e,$$

where $\|e\|_2 \leq \varepsilon$.

In order to derive Claim 14 from Claim 15, we assume that $|T| \geq 32d\ln((2d)/(\varepsilon/4)) = \Omega(d\log d)$. Then we can apply Claim 15 iteratively $|T|/2$ times with parameter $\varepsilon/4$, at each step identify the required h_a, remove it from T and continue. Next we prove Claim 15.

Proof of Claim 15. Let $\mathbb{B} := \{h \in \mathbb{R}^d : \|h\|_2 \leq 1\}$ denote the Euclidean ball of radius 1, and let C denote the convex hull of $\{h_2 - h_1, \ldots, h_t - h_{t-1}\}$. Observe that $C \subset 2\mathbb{B}$, as each h_i is a unit vector. For $\beta \in \{0,1\}^{t-1}$ define

$$h_\beta = \sum \beta_j (h_{j+1} - h_j).$$

We claim that having $t \geq 16d\ln(2d/\varepsilon)$ guarantees that there exist distinct β', β'' for which

$$h_{\beta'} - h_{\beta''} \in \frac{\varepsilon}{4}(C - C).$$

This follows by a packing argument: if not, then the sets $h_\beta + \frac{\varepsilon}{4}C$ for $\beta \in \{0,1\}^{t-1}$ are mutually disjoint. Each has volume $(\varepsilon/4)^d \mathrm{vol}(C)$, and they are all contained in tC which has volume $t^d \mathrm{vol}(C)$. As the number of distinct β is 2^{t-1} we obtain that $2^{t-1}(\varepsilon/4)^d \leq t^d$, which contradicts our assumption on t.

Let $i \in [t]$ be maximal such that $\beta'_{i-1} \neq \beta''_{i-1}$. We may assume without loss of generality that $\beta'_{i-1} = 0, \beta''_{i-1} = 1$, as otherwise we can swap the roles of β' and β''. Thus we have

$$\sum_{j=1}^{i-1} (\beta'_j - \beta''_j)(h_{j+1} - h_j) \in \frac{\varepsilon}{4}(C - C) \subset \varepsilon\mathbb{B}.$$

Adding $h_i - h_1 = \sum_{j=1}^{i-1}(h_{j+1} - h_j)$ to both sides gives

$$\sum_{j=1}^{i-1} (\beta'_j - \beta''_j + 1)(h_{j+1} - h_j) \in h_i - h_1 + \varepsilon\mathbb{B},$$

which is equivalent to

$$h_i - h_1 \in \sum_{j=1}^{i-1} (\beta'_j - \beta''_j + 1)(h_{j+1} - h_j) + \varepsilon\mathbb{B}.$$

The claim follows by setting $\alpha_j = \beta'_j - \beta''_j + 1$ and noting that by our construction $\alpha_{i-1} = 0$, and hence the sum terminates at $i - 2$. ◀

3.2 A deterministic LDT for H in general position

In this section, we derandomize the algorithm from the previous section.

▶ **Lemma 16.** *Let $H \subseteq \mathbb{R}^d$ be a finite set in general position. Then there exists an LDT that computes $\mathcal{A}_H(\cdot)$ with $O\left(d^4 \log d \log|H|\right)$ generalized comparison queries.*

Note that the this bound is worse by a factor of d than the one in Lemma 8. In Open Question 2 we ask whether this loss is necessary, or whether it can be avoided by a different derandomization technique.

Lemma 16 follows by derandomizing the algorithm from Lemma 8. Recall that Lemma 8 boils down to showing that $h \in \mathsf{InferComp}(S_i; x)$ for an $\Omega(1/d)$ fraction of $h \in H_i$ on average. In other words, for every input vector x, *most of the subsets* $S_i \subseteq H_i'$ of size $\Omega(d^2 \log d)$ allow to infer from comparisons the labels of some $\Omega(1/d)$-fraction of the points in H_i. We derandomize this step by showing that *there exists a universal set* $S_i \subseteq H_i'$ of size $O(d^3 \log d)$ that allows to infer the labels of some $\Omega(1/d)$-fraction of the points in H_i, with respect to *any* x. This is achieved by the next lemma.

▶ **Lemma 17.** *Let $H \subseteq \mathbb{R}^d$ be a set of unit vectors in isotropic position. Then there exists $S \subseteq H$ of size $O(d^3 \log d)$ such that*

$$\left(\forall x \in \mathbb{R}^d\right) : |\mathsf{InferComp}(S; x) \cap H| \geq \frac{|H|}{100d}.$$

Proof. We use a variant of the double-sampling argument due to [12] to show that a random $S \subseteq H$ of size $s = O(d^3 \log d)$ satisfies the requirements. Let $S = \{h_1, \ldots, h_s\}$ be a random (multi-)subset of size s, and let $E = E(S)$ denote the event

$$E(S) := \left[\exists x \in \mathbb{R}^d : |\mathsf{InferComp}(S; x) \cap H| < |H|/100d\right].$$

Our goal is showing that $\Pr[E] < 1$. To this end we introduce an auxiliary event F. Let $t = \Theta(d^2 \log d)$, and let $T = \{h_1, \ldots, h_t\} \subseteq S$ be a subsample of S, where each h_i is drawn uniformly from S and independently of the others. Define $F = F(S, T)$ to be the event

$$F(S,T) := \Big[\exists x \in \mathbb{R}^d : |\mathsf{InferComp}(T; x) \cap H| < |H|/100d \text{ and } |\mathsf{InferComp}(T; x) \cap S| \geq |S|/50d\Big].$$

The following claims conclude the proof of Lemma 17.

▶ **Claim 18.** *If $\Pr[E] \geq 9/10$ then $\Pr[F] \geq 1/200d$.*

▶ **Claim 19.** $\Pr[F] \leq 1/250d$.

This concludes the proof, as it shows that $\Pr[E] < 9/10$. We next move to prove Claim 18 and Claim 19.

Proof of Claim 18. Assume that $\Pr[E] \geq 9/10$. Define another auxiliary event $G = G(S)$ as

$$G(S) := [S \text{ is in } (3/4)\text{-approximate isotropic position}].$$

Applying Claim 7 by plugging $m \geq 100d \ln(10d)$ in Equation 1 gives that $\Pr[G] \geq 9/10$, which implies that $\Pr[E \wedge G] \geq 8/10$. Next, we analyze $\Pr[F|E \wedge G]$.

To this end, fix S such that both $E(S)$ and $G(S)$ hold. That is: S is in $(3/4)$-approximate isotropic position, and there exists $x = x(S) \in \mathbb{R}^d$ such that $|\mathsf{InferComp}(S; x) \cap H| < |H|/100d$.

If we now sample $T \subset S$, in order for $F(S,T)$ to hold, we need that (i) $|\mathsf{InferComp}(T;x) \cap H| < |H|/100d$, which holds with probability one, as $|\mathsf{InferComp}(S;x) \cap H| < |H|/100d$; and (ii) that $|\mathsf{InferComp}(T;x) \cap S| \geq |S|/50d$. So, we analyze this event next.

Applying Lemma 9 to the subsample T with respect to S gives that

$$\mathbb{E}_T\left[|\mathsf{InferComp}(T;x) \cap S|\right] \geq |S|/40d.$$

This then implies that

$$\Pr\left[|\mathsf{InferComp}(T;x) \cap S| \geq |S|/100d\right] \geq 1/100d.$$

To conclude: we proved under the assumptions of the lemma that $\Pr_S[E(S) \wedge G(S)] \geq 8/10$; and that for every S which satisfies $E(S) \wedge G(S)$ it holds that $\Pr_T[F(S,T)|S] \geq 1/100d$. Together these give that $\Pr[F(S,T)] \geq 1/200d$. ◀

Proof of Claim 19. We can model the choice of (S,T) as first sampling $T \subset H$ of size t, and then sampling $S \setminus T \subset H$ of size $s - t$. We will prove the following (stronger) statement: for any choice of T,

$$\Pr\left[F(S,T)|T\right] < 1/250d.$$

So from now on, fix T and consider the random choice of $T' = S \setminus T$. We want to show that:

$$\Pr_{T'}\Big[(\exists x \in \mathbb{R}^d) : |\mathsf{InferComp}(T;x) \cap H| < |H|/100d \text{ and } |\mathsf{InferComp}(T;x) \cap S| \geq |S|/50d.\Big] \leq 1/250d.$$

We would like to prove this statement by applying a union bound over all $x \in \mathbb{R}^d$. However, $\mathbb{R}^d$ is an infinite set and therefore a naive union seems problematic. To this end we introduce a suitable equivalence relation that is based on the following observation.

▶ **Observation 20.** $\mathsf{InferComp}(T;x)$ *is determined by sign*$(\langle h, x\rangle)$ *for* $h \in T \cup (T-T)$.

We thus define an equivalence relation on $\mathbb{R}^d$ where $x \sim y$ if and only if $\text{sign}(\langle h, x\rangle) = \text{sign}(\langle h, y\rangle)$ for all $h \in T \cup (T-T)$. Let C be a set of representatives for this relation. Thus, it suffices to show that

$$\Pr_{T'}\Big[(\exists x \in C) : |\mathsf{InferComp}(T;x) \cap H| < |H|/100d \text{ and } |\mathsf{InferComp}(T;x) \cap S| \geq |S|/50d.\Big] \leq 1/250d.$$

Since C is finite, a union bound is now applicable. Sepcifically, it is enough to show that

$$(\forall x \in C) : \Pr_{T'}\Big[\, |\mathsf{InferComp}(T;x) \cap H| < |H|/100d \text{ and } |\mathsf{InferComp}(T;x) \cap S| \geq |S|/50d.\Big] \leq \frac{1}{250d|C|}.$$

Now, (a variant of) Sauer's Lemma (see e.g. Lemma 2.1 in [7]) implies that

$$|C| \leq \left(2e \cdot |T \cup (T-T)|\right)^d \leq \left(2e \cdot t^2\right)^d \leq (20t)^{2d}. \tag{2}$$

Fix $x \in C$. If $|\mathsf{InferComp}(T;x) \cap H| \geq \frac{|H|}{100d}$ then we are done (note that $\mathsf{InferComp}(T;x)$ is fixed since it depends only on T and x and not on T'). So, we may assume that $|\mathsf{InferComp}(T;x) \cap H| < \frac{|H|}{100d}$. Then we need to bound

$$\Pr\left[|\mathsf{InferComp}(T;x) \cap S| \geq \frac{|S|}{50d}\right] \leq \Pr\left[|\mathsf{InferComp}(T;x) \cap T'| \geq \frac{|T'|}{75d}\right],$$

where the inequality follows if $t \leq \frac{s}{150d}$, which can be satisfied since $t = \Theta(d^2 \log d)$ and $s = \Theta(d^3 \log d)$. To bound this probability we use the Chernoff bound: let $p = \frac{|\mathsf{InferComp}(T;x) \cap H|}{|H|}$; note that $|\mathsf{InferComp}(T;x) \cap T'|$ is distributed like $\mathsf{Bin}(s-t,p)$. By assumption, $p \leq \frac{1}{100d}$, and therefore:

$$\begin{aligned}\Pr\left[|\mathsf{InferComp}(T;x) \cap T'| \geq \frac{|T'|}{75d}\right] &\leq \exp\left(-\frac{(1/3)^2 \cdot (t/100d)}{3}\right) \\ &\leq \frac{1}{250d \cdot (20t)^{2d}} \leq \frac{1}{250d \cdot |C|},\end{aligned}$$

where the second inequality follows because $t = \Theta(d^2 \ln(d))$ with a large enough constant, and the last inequality follows by Equation 3.2. ◀

◀

References

1. Franck Barthe. On a reverse form of the Brascamp-Lieb inequality. *Inventiones mathematicae*, 134(2):335–361, 1998.
2. Jean Cardinal, John Iacono, and Aurélien Ooms. Solving k-sum using few linear queries. *arXiv preprint arXiv:1512.06678*, 2015.
3. Esther Ezra and Micha Sharir. A nearly quadratic bound for the decision tree complexity of k-sum. In *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, pages 41:1–41:15, 2017.
4. Jürgen Forster. A linear lower bound on the unbounded error probabilistic communication complexity. *J. Comput. Syst. Sci.*, 65(4):612–625, 2002. doi:10.1016/S0022-0000(02)00019-3.
5. Michael L Fredman. How good is the information theory bound in sorting? *Theoretical Computer Science*, 1(4):355–361, 1976.
6. Daniel Kane, Shachar Lovett, Shay Moran, and Jiapeng Zhang. Active classification with comparison queries. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*. IEEE, 2017.
7. Daniel M. Kane, Shachar Lovett, and Shay Moran. Near-optimal linear decision trees for k-sum and related problems. *CoRR*, abs/1705.01720, 2017. arXiv:1705.01720.
8. Daniel M. Kane, Shachar Lovett, and Shay Moran. Generalized comparison trees for point-location problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 2018.
9. Stefan Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993.
10. Friedhelm Meyer auf der Heide. A polynomial linear search algorithm for the n-dimensional knapsack problem. *Journal of the ACM (JACM)*, 31(3):668–676, 1984.
11. Joel A. Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of Computational Mathematics*, 12(4):389–434, 2012. doi:10.1007/s10208-011-9099-z.
12. VN Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971.

Stabilizing Weighted Graphs

Zhuan Khye Koh
Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada
zkkoh@uwaterloo.ca

Laura Sanità
Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Canada
lsanita@uwaterloo.ca

Abstract

An edge-weighted graph $G = (V, E)$ is called *stable* if the value of a maximum-weight matching equals the value of a maximum-weight *fractional* matching. Stable graphs play an important role in some interesting game theory problems, such as *network bargaining* games and *cooperative matching* games, because they characterize instances which admit stable outcomes. Motivated by this, in the last few years many researchers have investigated the algorithmic problem of turning a given graph into a stable one, via edge- and vertex-removal operations. However, all the algorithmic results developed in the literature so far only hold for *unweighted* instances, i.e., assuming unit weights on the edges of G.

We give the first polynomial-time algorithm to find a minimum cardinality subset of vertices whose removal from G yields a stable graph, for any weighted graph G. The algorithm is combinatorial and exploits new structural properties of basic fractional matchings, which are of independent interest. In particular, one of the main ingredients of our result is the development of a polynomial-time algorithm to compute a basic maximum-weight fractional matching with minimum number of *odd cycles* in its support. This generalizes a fundamental and classical result on unweighted matchings given by Balas more than 30 years ago, which we expect to prove useful beyond this particular application.

In contrast, we show that the problem of finding a minimum cardinality subset of edges whose removal from a weighted graph G yields a stable graph, does not admit any constant-factor approximation algorithm, unless $P = NP$. In this setting, we develop an $O(\Delta)$-approximation algorithm for the problem, where Δ is the maximum degree of a node in G.

2012 ACM Subject Classification Mathematics of computing → Matchings and factors, Mathematics of computing → Approximation algorithms, Mathematics of computing → Graph algorithms, Theory of computation → Discrete optimization, Theory of computation → Algorithmic game theory, Theory of computation → Network games

Keywords and phrases combinatorial optimization, network bargaining, cooperative game

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.83

Related Version A full version of this paper is available at `https://arxiv.org/abs/1709.01982`.

Funding This work was supported by the NSERC Discovery Grant Program and an Early Researcher Award by the Province of Ontario.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 83; pp. 83:1–83:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Several interesting game theory problems are defined on networks, where the vertices represent players and the edges model the way players can interact with each other. In many such games, the structure of the underlying graph that describes the interactions among players is essential in determining the existence of stable outcomes for the corresponding games, i.e., outcomes where players have no incentive to deviate. Popular examples are *cooperative matching* games, introduced by Shapley and Shubik [17], and *network bargaining* games, defined by Kleinberg and Tardos [13], both extensively studied in the game theory community. Instances of such games are described by a graph $G = (V, E)$ with edge weights $w \in \mathbb{R}^E_{\geq 0}$, where V represents a set of players, and the value of a *maximum-weight matching*, denoted as $\nu(G)$, is the total value that the players could get by interacting with each other.

An important role in such games is played by so-called *stable* graphs. An edge-weighted graph $G = (V, E)$ is called stable if the value $\nu(G)$ of a maximum-weight matching equals the value of a maximum-weight *fractional* matching, denoted as $\nu_f(G)$. Formally, $\nu_f(G)$ is given by the optimal value of the standard linear programming relaxation of the matching problem, defined as

$$\nu_f(G) := \max \left\{ w^\top x : x(\delta(v)) \leq 1 \; \forall v \in V, x \geq 0 \right\} \tag{P}$$

Here x is a vector in $\mathbb{R}^E$, $\delta(v)$ denotes the set of edges incident to the node v, and for a set $F \subseteq E$, $x(F) = \sum_{e \in F} x_e$. Feasible solutions of the above LP are called *fractional matchings*.

The relation that interplays between stable graphs and network games is as follows. In cooperative matching games [17], the goal is to find an allocation of the value $\nu(G)$ among the vertices, given as a vector $y \in \mathbb{R}^V_{\geq 0}$, such that no subset $S \subseteq V$ has an incentive to form a *coalition* to deviate. This condition is formally defined by the constraints $\sum_{v \in S} y_v \geq \nu(G[S]) \; \forall S \subseteq V$, where $G[S]$ denotes the subgraph induced by S, and an allocation y that satisfies the above set of constraints is called *stable*. Deng et al. [8] proved that a stable allocation exists if and only if the graph describing the game is a stable graph. This is an easy consequence from LP duality. If y is a stable allocation, then y is a feasible solution to the dual of (P) and has value $\nu(G)$, showing that $\nu_f(G) = \nu(G)$. Conversely, if $\nu_f(G) = \nu(G)$, then an optimal dual solution yields a stable allocation of $\nu(G)$.

In network bargaining games [13], each edge e represents a deal of value w_e. A player can enter in a deal with at most one neighbor, and when a deal is made, the players have to agree on how to split the value of the deal between them. An outcome of the game is given by a pair (M, y), where M is a matching of G and stands for the set of deals made by the players, and $y \in \mathbb{R}^V_{\geq 0}$ is an allocation vector representing how the deal values have been split. Kleinberg and Tardos have defined a notion of *stable* outcome for such games, as well as a notion of *balanced* outcome, that are outcomes where players have no incentive to deviate, and in addition the deal values are "fairly" split among players. They proved that a balanced outcome exists if and only if a stable outcome exists, and this happens if and only if the graph G describing the game is stable.

Motivated by the above connection, in the last few years many researchers have investigated the algorithmic problem of turning a given graph into a stable one, by performing a minimum number of modifications on the input graph [6, 1, 10, 7, 14, 4, 5]. Two natural operations which have a nice network game interpretation, are vertex-deletion and edge-deletion. They correspond to *blocking players* and *blocking deals*, respectively, in order to achieve stability in the corresponding games. Formally, a subset of vertices $S \subseteq V$ is called a *vertex-stabilizer* if the graph $G \setminus S := G[V \setminus S]$ is stable. Similarly, a subset of edges $F \subseteq E$

is called an *edge-stabilizer* if the graph $G \setminus F := (V, E \setminus F)$ is stable. The corresponding optimization problems, which are the focus of this paper, are:

Minimum Vertex-stabilizer: *Given an edge-weighted graph $G = (V, E)$, find a minimum-cardinality vertex-stabilizer.*
Minimum Edge-stabilizer: *Given an edge-weighted graph $G = (V, E)$, find a minimum-cardinality edge-stabilizer.*

The above problems have been studied quite intensively in the last few years on unweighted graphs. In particular, Bock et al. [6] have showed that finding a minimum-cardinality edge-stabilizer is hard to approximate within a factor of $(2-\varepsilon)$, assuming Unique Game Conjecture (UGC) [11]. On the positive side, they have given an approximation algorithm for the edge-stabilizer problem, whose approximation factor depends on the sparsity of the input graph G. In other work, Ahmadian et al. [1] and Ito et al. [10] have shown independently that finding a minimum-cardinality vertex-stabilizer is a polynomial-time solvable problem. These (exact and approximate) algorithmic results, developed for unweighted instances, do not easily generalize when dealing with arbitrary edge-weights, since they heavily rely on the structure of maximum matchings in unweighted graphs. In fact, unweighted instances of the above problems exhibit a very nice property, as shown in [6, 1]: the removal of any inclusion-wise minimal edge-stabilizer (resp. vertex-stabilizer) from a graph G *does not decrease* the cardinality of a maximum matching in the resulting graph. This property ensures that there is at least one maximum-cardinality matching that survives in the modified graph, and this insight can be successfully exploited when designing (exact and approximate) algorithms. Unfortunately, it is not difficult to realize that this crucial property does not hold anymore when dealing with edge-weighted graphs, and in fact, the development of algorithmic results for weighted graphs requires substantial new ideas.

Our results and techniques

Vertex-stabilizers. We give the first polynomial-time algorithm to find a minimum-cardinality vertex-stabilizer S, in any weighted graph G. Our algorithm also ensures that $\nu(G \setminus S) \geq \frac{2}{3}\nu(G)$, i.e., the value of a maximum-weight matching is preserved up to a factor of $\frac{2}{3}$, and we show that this factor is tight in general. Specifically, as previously mentioned, a minimum-cardinality vertex-stabilizer for a weighted graph might decrease the value of a maximum-weight matching in the resulting graph. From a network bargaining perspective, this means we are decreasing the total value which the players are able to get, which is of course undesirable. However, we can show this is inevitable, since deciding whether there exists *any* vertex-stabilizer S that preserves the value of a maximum-weight matching (i.e., such that $\nu(G \setminus S) = \nu(G)$) is an NP-complete problem. Furthermore, we give an example of a graph G where *any* vertex-stabilizer S decreases the value of a maximum-weight matching by a factor of essentially $\frac{1}{3}$, i.e. $\nu(G \setminus S) \leq \left(\frac{2}{3} + \varepsilon\right) \nu(G)$ (for an arbitrary small $\varepsilon > 0$). This shows that the bounds of our algorithm are essentially best possible: the algorithm finds a vertex-stabilizer S whose cardinality is the *smallest* possible, and preserves the value of a maximum-weight matching up to a factor of $\frac{2}{3}$, that is the tightest factor that holds for all instances.

The above result is based on two main ingredients. The first one is giving a lower bound on the cardinality of a minimum vertex-stabilizer, which generalizes the lower bound used in the unweighted setting, and is based on the structure of optimal basic solutions of (P). In particular, it was shown in [1] that a lower bound on the cardinality of a vertex-stabilizer for unweighted graphs is given by the minimum number of *odd-cycles* in the support of an optimal

basic solution to (P). We show that this lower bound holds also for weighted graphs, though this generalization is not obvious (in fact, as we will show later, the same generalization does *not* hold for edge-stabilizers). Consequently, our proof is much more involved, and requires different ideas. The second main ingredient is giving a polynomial-time algorithm for computing an optimal basic solution to (P) with the smallest number of odd-cycles in its support, which is of independent interest, as highlighted in the next paragraph.

Computing maximum fractional matchings with minimum cycle support. The fractional matching polytope given by (P) has been extensively studied in the literature, and characterizing instances for which a maximum fractional matching equals an integral one is a natural graph theory question (see [6, 1]). It is well-known that basic solutions of (P) are half-integral, and the support of a basic solution is a disjoint union of a matching (given by 1-valued entries) and a set of odd-cycles (given by half-valued entries). Balas [2] gave a polynomial-time algorithm to compute a basic maximum fractional matching in an unweighted graph, with minimum number of odd-cycles in its support. This is a classical result on matching theory, which has been known for more than 30 years. In this paper, we generalize this result to arbitrary weighted instances, exploiting structural properties of basic fractional matchings. Our algorithm is based on combinatorial techniques, and we expect that this result will prove useful beyond this particular application.

Edge-stabilizers. When dealing with edge-removal operations, the stabilizer problem becomes harder, already in the unweighted setting. It is shown in [6] that finding a minimum edge-stabilizer is as hard as vertex cover, and whether the problem admits a constant factor approximation algorithm is an interesting open question. We here show that the answer to this question is negative for weighted graphs, since we prove that the minimum edge-stabilizer problem for a weighted graph G does not admit any constant-factor approximation algorithm, unless $P = NP$. From an approximation point of view, we show that the algorithm we developed for the vertex-stabilizer problem translates into a $O(\Delta)$-approximation algorithm for the edge-stabilizer problem, where Δ is the maximum degree of a node in G.

Once again, the analysis relies on proving a lower bound on the cardinality of a minimum edge-stabilizer. It was shown in [6] that a lower-bound on the cardinality of a minimum edge-stabilizer for unweighted graphs is again given by the minimum number of odd-cycles in the support of an optimal solution to (P) (called $\gamma(G)$). Interestingly, we show that, differently from the vertex-stabilizer setting, here this lower bound does not generalize, and $\gamma(G)$ *is not* a lower bound on the cardinality of an edge-stabilizer for arbitrary weighted graphs. However, we are able to show that $\lceil \gamma(G)/2 \rceil$ *is* a lower bound on the cardinality of a minimum edge-stabilizer, and this is enough for our approximation purposes.

Additional results. Lastly, we also generalize a result given in [1] on finding a minimum vertex-stabilizer which avoids a fixed maximum matching M, on unweighted graphs. We prove that if M is a maximum-weight matching of a weighted graph G, then finding a minimum vertex-stabilizer that is element-disjoint from M is a polynomial-time solvable problem. Otherwise, if M is not a maximum-weight matching, the problem is at least as hard as vertex cover. We supplement this result with a 2-approximation algorithm for this case, that is best possible assuming UGC [12].

Related work. Biró et al. [4] were the first to consider the edge-stabilizer problem in weighted graphs, and they showed NP-hardness for this case. Stabilizing a graph via different operations on the input graph (other than removing edges/vertices) has also been studied. In

particular, Ito et al. [10] have given polynomial-time algorithms to stabilize an unweighted graph by adding edges and by adding vertices. Chandrasekaran et al. [7] have recently studied the problem of stabilizing unweighted graphs by fractionally increasing edge weights. Ahmadian et al. [1] have also studied the vertex-stabilizer problem on unweighted graphs, but in the more-general setting where there are (non-uniform) costs for removing vertices, and gave approximation algorithms for this case.

Biró et al. [5] and Könemann et al. [14] studied a variant of the problem where the goal is to compute a minimum-cardinality set of *blocking pairs*, that are edges whose removal from the graph yield the existence of a fractional vertex cover of size at most $\nu(G)$ (but note that the resulting graph might not be stable). Mishra et al. [15] studied the problem of converting a graph into a *König-Egerváry graph*, via vertex-deletion and edge-deletion operations. A König-Egerváry graph is a graph where the size of a maximum matching equals the size of an (integral) minimum vertex cover. They gave an $O(\log n \log\log n)$-approximation algorithm for the vertex-removal setting in unweighted graphs, and showed constant-factor hardness of approximation (assuming UGC) for both the minimum vertex-removal and edge-removal problem.

Paper Organization. In Section 2, we give some preliminaries and discuss notation. In Section 3, we give a polynomial-time algorithm to compute an optimal basic solution to (P) with minimum number of odd cycles in its support. This algorithm will be crucially used in Section 4, where we give our results on vertex-stabilizers. The sections on edge-stabilizers and additional results can be found in the full version of this paper. All missing proofs also appear in the full version.

2 Preliminaries and notation

A key concept that we will use is LP duality. The dual of (P) is given by

$$\tau_f(G) := \min\left\{\mathbb{1}^\top y : y_u + y_v \geq w_{uv}\ \forall uv \in E, y \geq 0\right\}. \tag{D}$$

As feasible solutions to (P) are called fractional matchings, we call feasible solutions to (D) *fractional w-vertex covers*. In fact, (D) is the standard LP-relaxation of the problem of finding a minimum w-vertex cover, obtained by adding integrality constraints on (D). We also call basic feasible solutions to (P) as *basic fractional matchings*. An application of duality theory yields the following relationship $\nu(G) \leq \nu_f(G) = \tau_f(G)$. Recall that a graph G is *stable* if $\nu(G) = \nu_f(G) = \tau_f(G)$.

For a vector $x \in \mathbb{R}^E$ and any subset $F \subseteq E$, we denote $x_{-F} \in \mathbb{R}^{E-F}$ as the subvector obtained by dropping the entries corresponding to F. For any multisubset $F \subseteq E$, we define $x(F) := \sum_{e\in F} x_e$. Note that an element may be accounted for multiple times in the sum if it appears more than once in F. We denote $\mathrm{supp}(x) := \{e \in E : x_e \neq 0\}$ as the support of x. For any positive integer k, $[k]$ represents the set $\{1, 2, \ldots, k\}$.

Given an undirected graph G, we denote by n the number of vertices and by m the number of edges. For a matching M in G, a path is called *M-alternating* if its edges alternately belong to M and $E \setminus M$. We say that an M-alternating path is *valid* if it starts with an M-exposed vertex or an edge in M, and ends with an M-exposed vertex or an edge in M. For edge weights $w \in \mathbb{R}^m_+$, a valid M-alternating path P is called *M-augmenting* if $w(P \setminus M) > w(P \cap M)$. We will need the following classical result on the structure of basic fractional matchings:

▶ **Theorem 1** ([3]). *A fractional matching x in $G = (V, E)$ is basic if and only if $x_e = \{0, \frac{1}{2}, 1\}$ for all $e \in E$ and the edges e having $x_e = \frac{1}{2}$ induce vertex-disjoint odd cycles in G.*

Let $\hat{x}$ be a basic fractional matching in G. We partition the support of $\hat{x}$ into two parts. Define

$$\mathcal{C}(\hat{x}) := \{C_1, \ldots, C_q\} \quad \text{and} \quad M(\hat{x}) := \{e \in E : \hat{x}_e = 1\}$$

as the set of odd cycles such that $\hat{x}_e = \frac{1}{2}$ for all $e \in E(C_i)$ and the set of matched edges in $\hat{x}$ respectively. For ease of notation, we use $V(\mathcal{C}(\hat{x})) = \cup_{C \in \mathcal{C}(\hat{x})} V(C)$ and $E(\mathcal{C}(\hat{x})) = \cup_{C \in \mathcal{C}(\hat{x})} E(C)$ to denote the vertex set and edge set of $\mathcal{C}(\hat{x})$ respectively. We define two operations on the entries of $\hat{x}$ associated with certain edge sets of G:

▶ **Definition 2.** By *complementing* on $E' \subseteq E$, we mean replacing $\hat{x}_e$ by $\bar{x}_e = 1 - \hat{x}_e$ for all $e \in E'$.

▶ **Definition 3.** By *alternate rounding* on $C \in \mathcal{C}(\hat{x})$ at v where $C = \{e_1, \ldots, e_{2k+1}\}$ and $v = e_1 \cap e_{2k+1}$, we mean replacing $\hat{x}_e$ by $\bar{x}_e = 0$ for all $e \in \{e_1, e_3, \ldots, e_{2k+1}\}$ and $\bar{x}_e = 1$ for all $e \in \{e_2, e_4, \ldots, e_{2k}\}$. When v is clear from the context, we just say alternate rounding on C.

Let $\mathcal{X}$ be the set of basic maximum-weight fractional matchings in G. Define $\gamma(G) := \min_{\hat{x} \in \mathcal{X}} |\mathcal{C}(\hat{x})|$. Note that G is stable if and only if $\gamma(G) = 0$.

3 Maximum fractional matching with minimum support

In this section, we give a polynomial-time algorithm to compute a basic maximum-weight fractional matching $\hat{x}$ for a weighted graph G with minimum number of odd cycles in its support, i.e., satisfying $|\mathcal{C}(\hat{x})| = \gamma(G)$. This algorithm will be used as a subroutine by our vertex-stabilizer algorithm, which we will develop in Section 4.

Our first step is to characterize basic maximum-weight fractional matchings which have more than $\gamma(G)$ odd cycles. Balas [2] considered this problem on unweighted graphs, and gave the following characterization:

▶ **Theorem 4** ([2]). *Let $\hat{x}$ be a basic maximum fractional matching in an unweighted graph G. If $|\mathcal{C}(\hat{x})| > \gamma(G)$, then there exists an $M(\hat{x})$-alternating path which connects two odd cycles $C_i, C_j \in \mathcal{C}(\hat{x})$. Furthermore, alternate rounding on the odd cycles and complementing on the path produces a basic maximum fractional matching $\bar{x}$ such that $\mathcal{C}(\bar{x}) \subset \mathcal{C}(\hat{x})$.*

We generalize this to weighted graphs. Before stating the theorem, we need to introduce the concept of *connector* (see Figure 1 for some examples):

▶ **Definition 5.** Let C be a cycle and $S_0, S_1, \ldots, S_k$ be a partition of $V(C)$ such that $|S_0|$ is even and $k \geq 2$, where S_0 is allowed to be empty and $S_1, \ldots, S_k$ are non-empty. Let M be a perfect matching on the vertex set S_0. We call the graph $C \cup M$ a *connector*. Each S_i is called a *terminal set* for $i \geq 1$. An edge $e \in M$ is called a *chord* if $e \notin E(C)$.

Connectors are useful because of the following property:

▶ **Lemma 6.** *Let $C \cup M$ be a connector. For every terminal set S_i, there exists an M-alternating path in the connector from a vertex $v \in S_i$ to a vertex $u \in S_j$, for some $j \notin \{0, i\}$.*

Let y be a minimum fractional w-vertex cover in G. We say that an edge uv is *tight* if $y_u + y_v = w_{uv}$. Similarly, we say that a path is tight if all of its edges are tight.

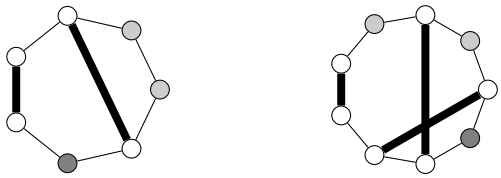


Figure 1 Two examples of connectors. Bold edges indicate M. Vertices of the same color belong to the same terminal set. White vertices are the ones in S_0.

▶ **Theorem 7.** *Let $\hat{x}$ be a basic maximum-weight fractional matching and y be a minimum fractional w-vertex cover in G. If $|\mathcal{C}(\hat{x})| > \gamma(G)$, then there exists*

(i) *a vertex $v \in V(C_i)$ for some odd cycle $C_i \in \mathcal{C}(\hat{x})$ such that $y_v = 0$; or*

(ii) *a tight $M(\hat{x})$-alternating path P which connects two odd cycles $C_i, C_j \in \mathcal{C}(\hat{x})$; or*

(iii) *a tight and valid $M(\hat{x})$-alternating path P which connects an odd cycle $C_i \in \mathcal{C}(\hat{x})$ and a vertex $v \notin V(\mathcal{C}(\hat{x}))$ such that $y_v = 0$.*

Furthermore, alternate rounding on the odd cycles and complementing on the path produces a basic maximum-weight fractional matching $\bar{x}$ such that $\mathcal{C}(\bar{x}) \subset \mathcal{C}(\hat{x})$.

Proof. We will start by proving the second part of the theorem, namely that alternate rounding and complementing produces a basic maximum-weight fractional matching with lesser odd cycles. For Case (i), let $\bar{x}$ be the basic fractional matching obtained by alternate rounding on C_i at v. Since $y_v = 0$, both $\bar{x}$ and y satisfy complementary slackness. Hence, $\bar{x}$ is optimal to (P) and $\mathcal{C}(\bar{x}) = \mathcal{C}(\hat{x}) \setminus C_i$. For Case (ii), denote $u = V(P) \cap V(C_i)$ and $v = V(P) \cap V(C_j)$ as the endpoints of P. Let $\bar{x}$ be the basic fractional matching obtained by alternate rounding on C_i, C_j at u, v respectively and complementing on P. Note that u and v are exposed after the alternate rounding, and covered after complementing. Since $\bar{x}$ and y satisfy complementary slackness, $\bar{x}$ is optimal to (P) and $\mathcal{C}(\bar{x}) = \mathcal{C}(\hat{x}) \setminus \{C_i, C_j\}$. For Case (iii), denote $u = V(P) \cap V(C_i)$ and $v \notin V(\mathcal{C}(\hat{x}))$ as the endpoints of P. Let $\bar{x}$ be the basic fractional matching obtained by alternate rounding on C_i at u and complementing on P. Since $y_v = 0$, both $\bar{x}$ and y satisfy complementary slackness. Thus, $\bar{x}$ is optimal to (P) and $\mathcal{C}(\bar{x}) = \mathcal{C}(\hat{x}) \setminus C_i$.

Next, we prove the first part of the theorem. We may assume $y_v > 0$ for every vertex $v \in V(\mathcal{C}(\hat{x}))$. Let x^* be a basic maximum-weight fractional matching in G such that $|\mathcal{C}(x^*)| = \gamma(G)$. Define $N(\hat{x}) := M(\hat{x}) \setminus E(\mathcal{C}(x^*))$ and $N(x^*) := M(x^*) \setminus E(\mathcal{C}(\hat{x}))$. Consider the following subgraph

$$J = (V, N(\hat{x}) \triangle N(x^*)).$$

Since $N(\hat{x})$ and $N(x^*)$ are matchings in G, J is made up of vertex-disjoint paths and cycles of G. For each such path or cycle, its edges alternately belong to $N(\hat{x})$ or $N(x^*)$. Moreover, its intermediate vertices are disjoint from $\mathcal{C}(\hat{x})$ and $\mathcal{C}(x^*)$. Since $\hat{x}$ and x^* are maximum-weight fractional matchings in G, every path in J is tight by complementary slackness. If there exists a path in J which connects two odd cycles from $\mathcal{C}(\hat{x})$, then we are done. If there exists a path in J which connects an odd cycle from $\mathcal{C}(\hat{x})$ and a vertex $v \notin V(\mathcal{C}(\hat{x}) \cup \mathcal{C}(x^*))$, then $y_v = 0$ because v is either exposed by $M(\hat{x})$ or $M(x^*)$. Hence, we are also done. So we may assume every path in J belongs to one of the following three categories:

(a) Vertex disjoint from $\mathcal{C}(\hat{x})$ and $\mathcal{C}(x^*)$.

(b) Starts and ends at the same cycle of $\mathcal{C}(\hat{x}) \cup \mathcal{C}(x^*)$.

(c) Connects an odd cycle from $\mathcal{C}(\hat{x})$ and an odd cycle from $\mathcal{C}(x^*)$.

Note that by the second part of the theorem, there is no path in J which connects two odd cycles from $\mathcal{C}(x^*)$ or an odd cycle from $\mathcal{C}(x^*)$ and a vertex $v \notin V(\mathcal{C}(\hat{x}) \cup \mathcal{C}(x^*))$. We say that two odd cycles C_i and C_j are *adjacent* if $V(C_i) \cap V(C_j) \neq \emptyset$ or if they are connected by a path in J.

▶ **Claim 1.** *Every cycle in $\mathcal{C}(\hat{x})$ is adjacent to a cycle in $\mathcal{C}(x^*)$.*

Proof. Let C be an odd cycle in $\mathcal{C}(\hat{x})$. For every vertex $v \in V(C)$, since we assumed $y_v > 0$, by complementary slackness it is either $M(x^*)$-covered or belongs to $V(\mathcal{C}(x^*))$. If $v \in V(\mathcal{C}(x^*))$, then we are done. So we may assume that every vertex in C is $M(x^*)$-covered. Let $uv \in M(x^*)$ where $u \in V(C)$ and $v \notin V(C)$. Observe that uv is the first edge of a path in J, so it either ends at an odd cycle in $\mathcal{C}(x^*)$ or C. Since C has an odd number of vertices, by the pigeonhole principle there exists a path in J which connects C and an odd cycle in $\mathcal{C}(x^*)$. ◀

Recall that we assumed no two cycles in $\mathcal{C}(\hat{x})$ are adjacent. We also know that no two cycles in $\mathcal{C}(x^*)$ are adjacent. Since $|\mathcal{C}(\hat{x})| > |\mathcal{C}(x^*)|$, by the previous claim there exists an odd cycle in $\mathcal{C}(x^*)$ which is adjacent to at least two odd cycles in $\mathcal{C}(\hat{x})$. Let $C^* \in \mathcal{C}(x^*)$ be adjacent to $C_1, \ldots, C_k \in \mathcal{C}(\hat{x})$ for some $k \geq 2$. For every $i \in [k]$, define

$$S_i := \{v \in V(C^*) : v \in V(C_i) \text{ or } \exists \text{ a path in } J \text{ from } v \text{ to } C_i\}$$

and $S_0 := V(C^*) \setminus \cup_{i=1}^{k} S_i$. Note that $y_v > 0$ for every vertex $v \in V(C^*)$. Hence, by complementary slackness every vertex in S_0 is $M(\hat{x})$-covered. Let $v \in S_0$. It is either matched to another vertex in S_0 or is an endpoint of a path in J whose other endpoint is also a vertex in S_0. Hence, $|S_0|$ is even. Moreover, $S_i \neq \emptyset$ for all $i \geq 1$, and the sets $S_0, \ldots, S_k$ partition $V(C^*)$. Let $\mathcal{P}$ be the set of paths in J that start and end at C^*, and consider the subgraph $C^* \cup \mathcal{P}$. We claim that there exists an $M(\hat{x})$-alternating path from S_i to S_j in $C^* \cup \mathcal{P}$ where $i \neq j$ and $i, j \neq 0$. Since every path in $\mathcal{P}$ starts and ends with an edge in $M(\hat{x})$, we can perform the following reduction: contract every path in $\mathcal{P}$ into a single edge in $M(\hat{x})$. It is easy to see that an $M(\hat{x})$-alternating path from S_i to S_j in $C^* \cup \mathcal{P}$ corresponds to an $M(\hat{x})$-alternating path from S_i to S_j in the reduced graph. Then, observe that the reduced graph along with the matching $M(\hat{x})$ forms a connector. By Lemma 6, there exists an $M(\hat{x})$-alternating path P from S_i to S_j in $C^* \cup \mathcal{P}$.

Let $v_i \in S_i$ and $v_j \in S_j$ be the endpoints of P. Let P_i and P_j be the paths in J connecting v_i to C_i and v_j to C_j respectively. If $v_i \in V(C_i)$, set $P_i = \emptyset$. Similarly if $v_j \in V(C_j)$, set $P_j = \emptyset$. Then, $P_i \cup P \cup P_j$ forms a tight $M(\hat{x})$-alternating path which connects C_i and C_j. ◀

Given a basic maximum-weight fractional matching $\hat{x}$ in G, we would like to reduce the number of odd cycles in $\mathcal{C}(\hat{x})$ to $\gamma(G)$. One way to accomplish this is to search for the structures described in Theorem 7. Fix a minimum fractional w-vertex cover y in G. Let G' be the unweighted graph obtained by applying the following operations to G (see Figure 2):

1. Delete all non-tight edges.
2. Add a vertex z.
3. For every vertex $v \in V$ where $\hat{x}(\delta(v)) = 1$ and $y_v = 0$, add the edge vz.
4. For every vertex $v \in V$ where $\hat{x}(\delta(v)) = 0$ and $y_v = 0$, add the vertex v' and the edges $vv', v'z$.
5. Shrink every odd cycle $C_i \in \mathcal{C}(\hat{x})$ into a pseudonode i.

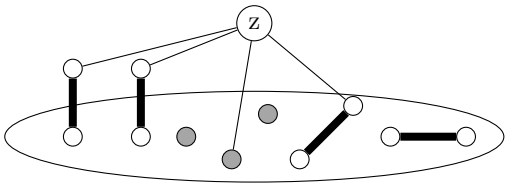


Figure 2 The auxiliary graph G' and the matching M'. Vertices in the ellipse are from the original graph G. Gray vertices represent pseudonodes.

Note that none of the edges in $M(\hat{x})$ and $\mathcal{C}(\hat{x})$ were deleted because they are tight. Consider the edge set $M' := M(\hat{x}) \cup \{vv' : v \in V\}$. It is easy to see that M' is a matching in G'. The significance of the auxiliary graph G' is given by the following lemma:

▶ **Lemma 8.** *M' is a maximum matching in G' if and only if $|\mathcal{C}(\hat{x})| = \gamma(G)$.*

Thus, searching for the structures in Theorem 7 is equivalent to searching for an M'-augmenting path in G'. This immediately gives us the following algorithm.

Algorithm 1: Minimize number of odd cycles.

```
1  Compute a basic maximum-weight fractional matching x̂ in G
2  Compute a minimum fractional w-vertex cover y in G
3  Construct G′ and M′
4  while ∃ an M′-exposed pseudonode r in G′ do
5      Grow an M′-alternating tree T rooted at r using Edmonds' algorithm [9]
6      if an M′-augmenting r-s path P′ is found in G′ then
7          Let P be the corresponding tight M(x̂)-alternating path in G
8          if s is a pseudonode then
9              Alternate round on C_r, C_s and complement on P
10         else
11             Alternate round on C_r and complement on P
12         Update G′ and M′
13     else
14         G′ ← G′ \ V(T)
15 return x̂
```

After an M'-augmenting path P' is found, let $\bar{x}$ denote the new basic maximum-weight fractional matching in G obtained by alternate rounding and complementing $\hat{x}$. We can update G' as follows. If s is a pseudonode, we unshrink C_r and C_s in G' because $\mathcal{C}(\bar{x}) = \mathcal{C}(\hat{x}) \setminus \{C_r, C_s\}$. Otherwise, $s = z$ and we only unshrink C_r. Then, there are two cases. In the first case, we have $vz \in E(P')$ for some $v \in V$. Observe that $\hat{x}(\delta(v)) = 1$ but $\bar{x}(\delta(v)) = 0$. Hence we replace the edge vz with edges $vv', v'z$. In the second case, we have $v'z \in E(P')$ for some $v \in V$. This implies $\hat{x}(\delta(v)) = 0$ but $\bar{x}(\delta(v)) = 1$. So we replace edges $vv', v'z$ with the edge vz.

▶ **Theorem 9.** *Algorithm 1 computes a basic maximum-weight fractional matching with $\gamma(G)$ odd cycles in polynomial time.*

We remark here that in Algorithm 1, we can avoid solving linear programs to obtain $\hat{x}$ and y in Steps 1 and 2. They can be computed using a simple duplication technique by Nemhauser and Trotter [16], which involves solving the problem on a suitable bipartite graph.

4 Computing vertex-stabilizers

The goal of this section is to prove the following theorem:

▶ **Theorem 10.** *There exists a polynomial-time algorithm that computes a minimum vertex-stabilizer S for a weighted graph G. Moreover, $\nu(G \setminus S) \geq \frac{2}{3}\nu(G)$.*

Let us start with discussing a lower bound on the size of a minimum vertex-stabilizer.

Lower bound. We will here prove that $\gamma(G)$ is a lower bound on the number of vertices to remove in order to stabilize a graph. Recall that a graph is stable if and only if $\gamma(G) = 0$. One strategy to achieve this is by showing that $\gamma(G)$ drops by at most 1 when a vertex is deleted (Lemma 12). We first develop a couple of claims.

▶ **Claim 2.** *Let $\hat{x}$ be a basic maximum-weight fractional matching and y be a minimum fractional w-vertex cover in G. Pick a vertex s from any odd cycle $C \in \mathcal{C}(\hat{x})$. If $\bar{x}$ is the fractional matching obtained by alternate rounding on C at s, then $\bar{x}_{-\delta(s)}$ is a basic maximum-weight fractional matching and y_{-s} is a minimum fractional w-vertex cover in $G \setminus s$.*

The following operation allows us to switch between fractional matchings on a set of edges:

▶ **Definition 11.** Let x and x' be fractional matchings in G. By *switching* on $E' \subseteq E$ from x to x', we mean replacing x_e by x'_e for all $e \in E'$.

Switching does not necessarily yield a feasible fractional matching. Hence, we will only use it on the components of a specific subgraph of G:

▶ **Claim 3.** *Given two basic fractional matchings x and x', let H be the subgraph of G induced by* $\operatorname{supp}(x + x')$*. For any component K in H, switching on $E(K)$ from x to x' yields a basic fractional matching in G.*

▶ **Lemma 12.** *For every vertex $v \in V$, $\gamma(G \setminus v) \geq \gamma(G) - 1$.*

Proof. Let x^* be a basic maximum-weight fractional matching in G such that $|\mathcal{C}(x^*)| = \gamma(G)$. Let y be a minimum fractional w-vertex cover in G. For the purpose of contradiction, suppose there exists a vertex $u \in V$ such that $\gamma(G \setminus u) < \gamma(G) - 1$. There are two cases:

Case 1: $u \in V(C)$ for some odd cycle $C \in \mathcal{C}(x^*)$. Let $\bar{x}$ be the fractional matching obtained from x^* by alternate rounding on C at u. By Claim 2, we know that $\bar{x}_{-\delta(u)}$ is a basic maximum-weight fractional matching and y_{-u} is a minimum fractional w-vertex cover in $G \setminus u$. We first give a proof sketch for this case. If $\bar{x}_{-\delta(u)}$ is not an optimal basic solution yielding $\gamma(G \setminus u)$ odd cycles, then one of the structures given by Theorem 7 must exist. This same structure would be a structure corresponding to the basic solution x^*, but this yields a contradiction since x^* is an optimal basic solution with $\gamma(G)$ odd cycles.

For notational convenience, we can use $\mathcal{C}(\bar{x})$ and $M(\bar{x})$ to refer to the odd cycles and matched edges of $\bar{x}_{-\delta(u)}$ respectively because $\mathcal{C}(\bar{x}) = \mathcal{C}\left(\bar{x}_{-\delta(u)}\right)$ and $M(\bar{x}) = M\left(\bar{x}_{-\delta(u)}\right)$. Since $|\mathcal{C}(\bar{x})| = |\mathcal{C}(x^*)| - 1 = \gamma(G) - 1 > \gamma(G \setminus u)$, Theorem 7 tells us that $G \setminus u$ contains one of the following structures. The first structure is a vertex $v \in V(C_i)$ for some odd cycle $C_i \in \mathcal{C}(\bar{x})$ such that $y_v = 0$. However, since $C_i \in \mathcal{C}(x^*)$, by Theorem 7 we arrive at the contradiction $|\mathcal{C}(x^*)| > \gamma(G)$. The second structure is a tight and valid $M(\bar{x})$-alternating path P which connects two odd cycles $C_i, C_j \in \mathcal{C}(\bar{x})$, or an odd cycle $C_i \in \mathcal{C}(\bar{x})$ and a vertex

$v \notin V(\mathcal{C}(\bar{x}))$ such that $y_v = 0$. Note that $C_i, C_j \in \mathcal{C}(x^*)$. If $V(P) \cap V(C) = \emptyset$, then P is also a tight and valid $M(x^*)$-alternating path in G which connects C_i and C_j, or C_i and v. So, let $s = V(C_i) \cap V(P)$ and t denote the first vertex of C encountered while traversing along P from s. Then, the s-t subpath of P is a tight $M(x^*)$-alternating path which connects $C_i, C \in \mathcal{C}(x^*)$. We again obtain the contradiction $|\mathcal{C}(x^*)| > \gamma(G)$ by Theorem 7.

Case 2: $u \notin V(\mathcal{C}(x^*))$. If u is $M(x^*)$-exposed, then $\nu_f(G \setminus u) = \nu_f(G)$ and $\gamma(G \setminus u) = \gamma(G)$. So we may assume u is $M(x^*)$-covered. Let $\hat{x}$ be a basic maximum-weight fractional matching in $G \setminus u$ such that $|\mathcal{C}(\hat{x})| < \gamma(G) - 1$. Define $N(\hat{x}) := M(\hat{x}) \setminus E(\mathcal{C}(x^*))$ and $N(x^*) := M(x^*) \setminus E(\mathcal{C}(\hat{x}))$. Consider the subgraph $J = (V, N(x^*) \triangle N(\hat{x}))$. Note that u is covered by $N(x^*)$ and exposed by $N(\hat{x})$. Let P be the component in J which contains u. We know that P is a path with u as an endpoint. Let v be the other endpoint of P. There are 3 subcases, but before jumping into them, we first give an overview of how we arrive at a contradiction in each subcase. We show that one can move from x^* to a new solution $\tilde{x}$ such that:

- **(i)** $\tilde{x}$ is a basic maximum-weight fractional matching for a subgraph G' obtained by deleting at most 1 vertex from a cycle of $\mathcal{C}(x^*)$; and
- **(ii)** $|\mathcal{C}(\tilde{x})| < \gamma(G')$.

Clearly, both of the above properties cannot hold, so this yields a contradiction.

Subcase 2.1: $v \in C$ for some odd cycle $C \in \mathcal{C}(x^*)$. In this subcase, the path P has even length. Let $\bar{x}$ be the fractional matching obtained from x^* by alternate rounding on C at v. By Claim 2, $\bar{x}_{-\delta(v)}$ is a basic maximum-weight fractional matching in $G \setminus v$. Let H be the subgraph of G induced by $\text{supp}(\hat{x} + \bar{x})$. Note that $\hat{x}_e + \bar{x}_e = 0$ for every edge $e \notin E(P)$ which is incident to a vertex in P. Thus, P is a component in H. Since $|\mathcal{C}(\bar{x})| = \gamma(G) - 1 > |\mathcal{C}(\hat{x})|$, there exists a component K in H which has more odd cycles from $\mathcal{C}(\bar{x})$ than $\mathcal{C}(\hat{x})$. Switching on K from $\bar{x}_{-\delta(v)}$ to $\hat{x}$ yields a basic fractional matching in $G \setminus v$ with less than $\gamma(G) - 1$ odd cycles. To yield a contradiction to Case 1, it is left to show that it is maximum-weight. This is because we are deleting a vertex v from an odd cycle of $\mathcal{C}(x^*)$, but $\gamma(G \setminus v)$ decreases by more than 1. Now, since $\hat{x}$ and $\bar{x}_{-\delta(v)}$ are maximum-weight fractional matchings in $G \setminus u$ and $G \setminus v$ respectively, we have $\sum_{e \in E(K)} w_e \hat{x}_e = \sum_{e \in E(K)} w_e \bar{x}_e$ because $u, v \notin V(K)$. Thus, the resulting matching is indeed maximum-weight in $G \setminus v$.

Subcase 2.2: $v \in C$ for some odd cycle $C \in \mathcal{C}(\hat{x})$. In this subcase, the path P has odd length. Let $\bar{x}$ be the fractional matching obtained from $\hat{x}$ by alternate rounding on C at v. By Claim 2, $\bar{x}_{-\delta(v)}$ is a basic maximum-weight fractional matching in $G \setminus \{u, v\}$. Let H be the subgraph of G induced by $\text{supp}(x^* + \bar{x})$. Note that $x^*_e + \bar{x}_e = 0$ for every edge $e \notin E(P)$ incident to a vertex in P. Thus, P is a component in H. Since $|\mathcal{C}(\bar{x})| = |\mathcal{C}(\hat{x})| - 1 < \gamma(G) - 2 < |\mathcal{C}(x^*)|$, there exists a component K in H which has more odd cycles from $\mathcal{C}(x^*)$ than $\mathcal{C}(\bar{x})$. Switching on K from x^* to $\bar{x}$ yields a basic fractional matching in G with less than $\gamma(G)$ odd cycles. To yield a contradiction, it is left to show that it is maximum-weight. Since x^* and $\bar{x}_{-\delta(v)}$ are maximum-weight fractional matchings in G and $G \setminus \{u, v\}$ respectively, we have $\sum_{e \in E(K)} w_e x^*_e = \sum_{e \in E(K)} w_e \bar{x}_e$ because $u, v \notin V(K)$. Thus, the resulting basic fractional matching is maximum-weight in G.

Subcase 2.3: $v \notin V(\mathcal{C}(x^*) \cup \mathcal{C}(\hat{x}))$. Let H be the subgraph of G induced by $\text{supp}(x^* + \hat{x})$. Note that $x^*_e + \hat{x}_e = 0$ for every edge $e \notin E(P)$ which is incident to a vertex in P. Thus, the path P is a component in H. Since $|\mathcal{C}(x^*)| > \gamma(G) - 1 > |\mathcal{C}(\hat{x})|$, there exists a component K in H which has more odd cycles from $\mathcal{C}(x^*)$ than $\mathcal{C}(\hat{x})$. Switching on K from x^* to $\hat{x}$ yields a basic fractional matching in G with less than $\gamma(G)$ odd cycles. To yield a contradiction, it is left to show that it is maximum-weight. Since x^* and $\hat{x}$ are maximum-weight fractional

matchings in G and $G \setminus u$ respectively, we have $\sum_{e \in E(K)} w_e x^*_e = \sum_{e \in E(K)} w_e \hat{x}_e$ because $u \notin V(K)$. Hence, the resulting basic fractional matching is maximum-weight in G. ◀

As a corollary to the above lemma, we obtain the claimed lower bound.

▶ **Lemma 13.** *For every vertex-stabilizer S of G, $|S| \geq \gamma(G)$.*

The algorithm. The algorithm we use to stabilize a graph is very simple: it computes a basic maximum-weight fractional matching $\hat{x}$ in G with $\gamma(G)$ odd cycles (this can be done using Algorithm 1) and a minimum fractional w-vertex cover y in G, and then removes *one* vertex from every cycle in $\mathcal{C}(\hat{x})$, namely, the vertex with the least y-value in the cycle. Algorithm 2 formalizes this.

Algorithm 2: Minimum vertex-stabilizer

```
Initialize S ← ∅
Compute a minimum fractional w-vertex cover y in G
Compute a basic maximum-weight fractional matching x̂ in G with γ(G) odd cycles
Let 𝒞(x̂) = {C_1, C_2, ..., C_γ(G)}
for i = 1 to γ(G) do
    Let v_i = arg min_{v ∈ V(C_i)} y_v
    S ← S + v_i
return S
```

We are now ready to prove the main theorem stated at the beginning of the section, Theorem 10.

Proof of Theorem 10. Let $S = \{v_1, v_2, \ldots, v_{\gamma(G)}\}$ be the set of vertices returned by the algorithm. Let $\bar{x}$ be the vector obtained from $\hat{x}$ by alternate rounding on C_i at v_i for all i respectively. By Lemma 2, $\bar{x}_{-\bigcup_{i=1}^{\gamma(G)} \delta(v_i)}$ is a basic maximum-weight fractional matching in $G \setminus S$. Note that it is also a maximum-weight integral matching in $G \setminus S$. Thus, $\nu(G \setminus S) = \nu_f(G \setminus S)$ and $G \setminus S$ is stable. Moreover, S is minimum by Lemma 13. It is left to show that $\nu(G \setminus S) \geq \frac{2}{3}\nu(G)$. For every odd cycle $C_i \in \mathcal{C}(\hat{x})$, we have

$$y_{v_i} \leq \frac{y(V(C_i))}{|V(C_i)|} \leq \frac{y(V(C_i))}{3}$$

because v_i has the smallest fractional w-vertex cover in C_i. From Lemma 2, we also know that y_{-S} is a minimum fractional w-vertex cover in $G \setminus S$. Then,

$$\nu(G \setminus S) = \tau_f(G \setminus S) = \mathbb{1}^\top y - \sum_{i=1}^{\gamma(G)} y_{v_i} \geq \mathbb{1}^\top y - \frac{1}{3}\sum_{i=1}^{\gamma(G)} y(C_i) \geq \mathbb{1}^\top y - \frac{1}{3}\mathbb{1}^\top y = \frac{2}{3}\tau_f(G) \geq \frac{2}{3}\nu(G) \blacktriangleleft$$

Note that removing any single vertex from each cycle of $\mathcal{C}(\hat{x})$ yields a minimum-cardinality vertex stabilizer. The reason we chose the vertex with the smallest y_v is to preserve the value of the original maximum-weight matching by a factor of $\frac{2}{3}$.

Tightness of the matching bound. A natural question is whether it is possible to design an algorithm that always returns a vertex-stabilizer S satisfying $\nu(G \setminus S) \geq \alpha\nu(G)$, for some $\alpha > \frac{2}{3}$. We report an example in the full version of this paper showing that, in general, this is not possible since the bound of $\frac{2}{3}$ can be asymptotically tight.

Another natural question is whether one can at least distinguish if, for a specific instance, there exists a vertex-stabilizer S such that $\nu(G \setminus S) = \nu(G)$. Once again, we show that the answer is negative. Specifically, let us call a vertex-stabilizer S *weight-preserving* if

$\nu(G \setminus S) = \nu(G)$. We show that finding such a vertex-stabilizer is hard in general. The proof is based on a reduction from the independent set problem, similar to the one given by Biró et al. [4].

▶ **Theorem 14.** *Deciding whether a graph has a weight-preserving vertex-stabilizer is NP-complete.*

References

1 Sara Ahmadian, Hamideh Hosseinzadeh, and Laura Sanità. Stabilizing network bargaining games by blocking players. In *Proceedings of the 18th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 164–177, 2016.

2 Egon Balas. Integer and fractional matchings. *North-Holland Mathematics Studies*, 59:1–13, 1981.

3 Michel Balinski. On maximum matching, minimum covering and their connections. In *Proceedings of the Princeton Symposium on Mathematical Programming*, pages 303–312, 1970.

4 Péter Biró, Matthijs Bomhoff, Petr A. Golovach, Walter Kern, and Daniël Paulusma. Solutions for the stable roommates problem with payments. *Theor. Comput. Sci.*, 540:53–61, 2014.

5 Péter Biró, Walter Kern, and Daniël Paulusma. Computing solutions for matching games. *Int. J. Game Theory*, 41(1):75–90, 2012.

6 Adrian Bock, Karthekeyan Chandrasekaran, Jochen Könemann, Britta Peis, and Laura Sanità. Finding small stabilizers for unstable graphs. *Math. Program.*, 154(1-2):173–196, 2015.

7 Karthekeyan Chandrasekaran, Corinna Gottschalk, Jochen Könemann, Britta Peis, Daniel Schmand, and Andreas Wierz. Additive stabilizers for unstable graphs. *arXiv e-prints*, Aug 2016. `arXiv:1608.06797`.

8 Xiaotie Deng, Toshihide Ibaraki, and Hiroshi Nagamochi. Algorithmic aspects of the core of combinatorial optimization games. *Math. Oper. Res.*, 24(3):751–766, 1999.

9 Jack Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17:449–467, 1965.

10 Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto. Efficient stabilization of cooperative matching games. *Theor. Comput. Sci.*, 677:69–82, 2017.

11 Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 767–775, 2002.

12 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

13 Jon M. Kleinberg and Éva Tardos. Balanced outcomes in social exchange networks. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 295–304, 2008.

14 Jochen Könemann, Kate Larson, and David Steiner. Network bargaining: Using approximate blocking sets to stabilize unstable instances. *Theory Comput. Syst.*, 57(3):655–672, 2015.

15 Sounaka Mishra, Venkatesh Raman, Saket Saurabh, Somnath Sikdar, and C. R. Subramanian. The complexity of könig subgraph problems and above-guarantee vertex cover. *Algorithmica*, 61(4):857–881, 2011.

16 George L. Nemhauser and Leslie E. Trotter Jr. Vertex packings: Structural properties and algorithms. *Math. Program.*, 8(1):232–248, 1975.

17 Lloyd S. Shapley and Martin Shubik. The assignment game I: The core". *International Journal of Game Theory*, 1(1):111–130, 1971.

Spectrally Robust Graph Isomorphism

Alexandra Kolla[1]
Department of Computer Science, University of Colorado at Boulder
alexandra.kolla@colorado.edu

Ioannis Koutis[2]
Department of Computer Science, New Jersey Institute of Technology
ioannis.koutis@njit.edu

Vivek Madan
Department of Computer Science, University of Illinois, Urbana-Champaign
vmadan2@illinois.edu

Ali Kemal Sinop
TOBB University of Economics and Technology, Ankara, Turkey
asinop@gmail.com

Abstract

We initiate the study of spectral generalizations of the graph isomorphism problem.

(a) The *Spectral Graph Dominance (SGD)* problem:
On input of two graphs G and H does there exist a permutation π such that $G \preceq \pi(H)$?

(b) The *Spectrally Robust Graph Isomorphism (κ-SRGI)* problem:
On input of two graphs G and H, find the smallest number κ over all permutations π such that $\pi(H) \preceq G \preceq \kappa c \pi(H)$ for some c. SRGI is a natural formulation of the network alignment problem that has various applications, most notably in computational biology.

$G \preceq cH$ means that for all vectors x we have $x^T L_G x \leq c x^T L_H x$, where L_G is the Laplacian G.

We prove NP-hardness for SGD. We also present a κ^3-approximation algorithm for SRGI for the case when both G and H are bounded-degree trees. The algorithm runs in polynomial time when κ is a constant.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Network Alignment, Graph Isomorphism, Graph Similarity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.84

Related Version A full version of the paper is available at [12], `https://arxiv.org/abs/1805.00181`.

1 Introduction

Network alignment, a problem loosely defined as the comparison of graphs under permutations, has a very long history of applications in disparate fields [5]. Notably, alignment of protein and other biological networks are among the most recent and popular applications [18, 6]. There are several heuristic algorithms for the problem; naturally some of them are based on generalizations of the graph isomorphism problem, mostly including variants of the robust

[1] Supported by NSF CAREER grant CCF-1452923.
[2] Supported by NSF CAREER grant CCF-1319376.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 84; pp. 84:1–84:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

graph isomorphism problem which asks for a permutation that minimizes the number of 'mismatched' edges [16].

Robust graph isomorphism may not be always an appropriate problem for applications where one wants to certify the 'functional' equivalence of two graphs. Consider for example the case when G and H are two random constant-degree expanders. While they can be arguably functionally equivalent (e.g. as information dispersers), all permutations will incur a large number of edge mismatches, deeming the two graphs very unsimilar. Functional equivalence is of course an application-dependent notion. In the case of protein networks, it is understood that proteins act as electron carriers [9]. Thus it is reasonable to model them as electrical resistive networks that are algebraically captured by graph Laplacian matrices [4]. Going back to the graph isomorphism problem, we note the simple fact that the Laplacian matrices of two isomorphic graphs share the same eigenvalues, with the corresponding eigenspaces being identical up to the isomorphism. We can them aim for a **spectrally robust** version of graph isomorphism (SRGI) which allows for similar eigenvalues and approximately aligned eigenspaces, up to a permutation.

In lieu of using directly the eigenvalues and eigenspaces to define SRGI, we will rely on the much cleaner notion of spectral graph similarity, which underlies spectral sparsification of graphs, a notion that has been proven extremely fruitful in algorithm design [2, 14]. More concretely, let us introduce the precise notion of similarity we will be using.

▶ **Definition 1** (dominance)**.** We say that graph G dominates graph H ($G \preceq H$), when for all vectors x, we have $x^T L_G x \leq x^T L_H x$, where L_G is the standard Laplacian matrix for G.

▶ **Definition 2** (κ-similarity)**.** We say that graphs G and H are κ-similar, when when there exist numbers β and γ, such that $\kappa = \gamma/\beta$ and $\beta H \preceq G \leq \gamma H$.[3]

We are now ready to introduce our main problem.

Spectrally Robust Graph Isomorphism (κ-SRGI): Given two graphs G, H, does there exist a permutation π on $V(G)$ such that G and $\pi(H)$ are κ-similar?

It can been shown that this definition does imply approximately equal eigenvalues and aligned eigenspaces [13], thus testing for κ-similarity under permutations is indeed a spectrally robust version of graph isomorphism. Going back to our example with the two random expanders, it is well-understood that G and $\pi(H)$ will be κ-similar for a constant κ and for all permutations π, which is what we intuitively expect.

We view spectrally robust graph isomorphism as an interesting theoretical problem due to its close relationship with other fundamental algorithmic questions. In particular, it can be easily seen that κ-SRGI is equivalent to the graph isomorphism problem when $\kappa = 1$. As we will discuss in more detail, SRGI can also be viewed as a natural generalization of the minimum distortion problem [10]. Up to our knowledge, the spectral-similarity approach to network alignment has been mentioned earlier only in [20]. In view of the vast number of works on GI ([11, 7, 21, 19, 3, 15, 1] to mention a few) as well as the works on the robust graph isomorphism problem [16] and the minimum distortion problem [10], we find it surprising that SRGI has not received a wider attention.

The goal of this work is to prove some initial results on SRGI and stimulate further research. Towards that end, we provide the first algorithm for this problem, for the case when both graphs are trees.

[3] Graphs are weighted and cG is graph G with its edge weights multiplied by c.

▶ **Theorem 3.** *Given two κ-similar trees G and H of maximum degree d, there exists an algorithm running in time $O(n^{O(k^2 d)})$ which finds a mapping certifying that the they are at most κ^4-similar.*

The algorithm for trees is already highly involved, which gives grounds for speculating that the problem is NP-hard. We give evidence that this may be indeed true by turning our attention to the one-sided version of the problem.

Spectral Graph Dominance (SGD): Given two graphs G, H, does there exist a permutation π such that G dominates $\pi(H)$?

Given two graphs G and H that have the same eigenvalues, it is not hard to prove that if G and H are not isomorphic, then G cannot dominate H (and vice-versa). Combining this with the fact that isomorphic graphs have the same eigenvalues, we infer that SGD is at least graph isomorphism-hard. The second contribution of this work is the following theorem.

▶ **Theorem 4.** *The* Spectral Graph Dominance *problem is NP-hard.*

Theorem 4 is proved in Section 2. We can actually prove a slightly stronger theorem that restricts one of the input graphs to be a tree.

1.1 Related Work

The Robust Graph Isomorphism problem (RGI) asks for a permutation that minimizes the number of mismatched edges. O'Donnell *et al.* [16] gave a constant factor hardness for RGI . The Minimum Distortion problem (MD) views graphs as distance metrics, using the shortest path metric. The goal is to find a mapping between the two metrics so as to minimize the maximum distortion. The connection between SRGI and MD stems from the observation that if two tree graphs G and H are κ-similar up to a permutation π, then the distortion between the induced graph distances of G and $\pi(H)$ is at most κ. For the MD problem, Kenyon *et al.* [10] gave an algorithm which finds a solution with distortion at most α (provided that it exists) in time $\text{poly}(n)\exp(d^{O(\alpha^3)})$, for a tree of degree at most g and an arbitrary weighted graph. They also prove that this problem is NP-hard to approximate within a constant factor.

The term 'spectral alignment' has been used before in [6] in the context of spectral relaxation of the graph matching function. The algorithm in [18] is more spectral 'in spirit' because it uses directly the spectral of the normalized Laplacians of several subgraphs to construct complicated 'graph signatures' that are then compared for similarity. There is no underlying objective function that drives the computation of these signatures, but we imagine that the proposed algorithm or some variant of it, may be a reasonably good practical candidate for SRGI. The work by Tsourakakis [20] proposes an algorithm that searches for the optimal permutation via a sequence of transpositions; however the running time of the algorithm does not have any non-trivial sub-exponential upper bound.

2 Graph Dominance

Preliminaries. Given a weighted graph $G = (V, E, w)$ we denote by E_G its edges. The Laplacian L_G of G is the matrix defined by $L(i,j) = -w_{ij}$ and $L(i,i) = \sum_{i \neq j} w_{ij}$. The quadratic form $\mathcal{R}(G, x)$ of G is the function defined as:

$$\mathcal{R}(G, x) = x^T L_G x = \sum_{i,j} w_{ij}(x_i - x_j)^2. \tag{1}$$

Let G^∞ be the infinite graph with vertex set equal to all points on the plane with integer coordinates. There is an edge between two points of G^∞ if they have Euclidean distance one. A *cubic subgrid* is a finite subgraph of G^∞ such that all of its nodes have degree at most 3.

The main ingredient of the proof is the following theorem.

▶ **Theorem 5.** *Let G be a cubic subgrid and C be the cycle graph, both on n vertices. There exists a permutation π such that $\pi(C) \preceq G$ if and only if G contains a Hamiltonian cycle.*

Proof. If G contains a Hamiltonian cycle $\pi(C)$, then equation 1 directly implies that $\pi(C) \preceq G$. To prove the converse assume that G does not contain a Hamiltonian cycle and let H be a permutation of C such that $|E_G \cap E_H|$ is maximized. We prove a number of claims and lemmas.

▶ **Claim 1.** *Let G', H' be the graphs obtained by deleting the common edges between G and H respectively. Then, $\mathcal{R}(G,x) < \mathcal{R}(H,x)$ if and only if $\mathcal{R}(G',x) < \mathcal{R}(H',x)$.*

Proof. Let F be the graph induced by the edges shared by G and H. By equation 1 we have $\mathcal{R}(G,x) = \mathcal{R}(G',x) + \mathcal{R}(F,x)$ and $\mathcal{R}(H,x) = \mathcal{R}(H',x) + \mathcal{R}(F,x)$. The claim follows. ◀

▶ **Claim 2.** *Let v be a vertex with $deg_{G'}(v) = 1$, $deg_{H'}(v) = 0$ and let G'' be the graph obtained from G' after deleting the edge incident to v, and set $H'' = H'$. Then, there exists a vector x s.t. $\mathcal{R}(H',x) > \mathcal{R}(G',x)$ iff there exists a vector y s.t. $\mathcal{R}(H'',y) > \mathcal{R}(G'',y)$.*

Proof. Let x be a vector such that $\mathcal{R}(H',x) > \mathcal{R}(G',x)$. Since G'' is a subgraph of G', we have $\mathcal{R}(G'',x) \leq \mathcal{R}(G',x) < \mathcal{R}(H',x) = \mathcal{R}(H'',x)$, and we can take $y = x$. For the converse, assume that there is a vector y such that $\mathcal{R}(H'',y) > \mathcal{R}(G'',y)$. Let (v,w) be the edge incident to v in G'. We define a vector x as follows: $x_u = y_u$ for all $u \neq v$ and $x_v = y_w$. Since, $deg_{H''}(v) = 0$, we have $\mathcal{R}(H'',y) = \mathcal{R}(H'',x)$. On the other hand, G' and G'' agree on all the edges except (v,w). Hence, $\mathcal{R}(G',x) = \mathcal{R}(G'',x) + (x_v - x_w)^2 = \mathcal{R}(G'',x)$. The two vectors x and y agree on all the entries except at v, and the degree of v in G'' is zero. Hence, $\mathcal{R}(G'',x) = \mathcal{R}(G'',y)$. Combining all the inequalities, we get:
$\mathcal{R}(G',x) = \mathcal{R}(G'',x) = \mathcal{R}(G'',x) < \mathcal{R}(H'',y) = \mathcal{R}(H'',x) = \mathcal{R}(H',x)$. ◀

▶ **Claim 3.** *Let G' and H' be the graphs obtained by deleting the shared edges between G and H as in Claim 1. Let $\tilde{G}$ and $\tilde{H}$ be the graphs obtained starting from G' and H' and repeatedly applying the edge deletion operation of Claim 2. Then, for any vertex u, $deg_{\tilde{G}}(u) \leq deg_{\tilde{H}}(u) + 1$.*

Proof. Since G is a cubic subgrid graph and H is a cycle, $deg_G(u) \leq 3$, $deg_H(u) = 2$, for all vertices u. Deleting edges shared between G and H decreases the degree of any given vertex by the same amount in G and H. Moreover, at any given step, we only delete edges from G'. Hence, $deg_{\tilde{G}}(u) \leq deg_{\tilde{H}}(u) + 1$. ◀

▶ **Claim 4.** *Let G' and H' be the graphs obtained by deleting the shared edges between G and H as in Claim 1. If there exists a vertex v such that $deg_{G'}(v) = 1, deg_{H'}(v) \geq 1$. Then, there exists a vector x such that $\mathcal{R}(H',x) > \mathcal{R}(G',x)$.*

Proof. Let the edge incident to v in G' be (v,w) and an edge incident to v in H' be (v,u). Since, H' and G' do not share any edge, we have $u \neq w$. Let $x \in \mathbb{R}^n$ be a vector defined as follows: $x_v = 0, x_w = \frac{1}{2}$ and $x_t = 1$ otherwise. We have $\mathcal{R}(H',x) > (x_v - x_u)^2 = 1$, and

$$\mathcal{R}(G',x) = (x_v - x_w)^2 + \sum_{(w,a)\in E'_G, a\neq v} (x_w - x_a)^2$$

Vertex w has at most two neighbors other than v in G, since $deg_{G'}(v) \leq 3$ and for any such neighbor a, we have $(x_a - x_w)^2 = (\frac{1}{2} - 1)^2 = 1/4$. Hence $\mathcal{R}(G', x) \leq 3/4 < 1 \leq \mathcal{R}(H', x)$. ◀

Let G' and H' be the graphs obtained by deleting the shared edges between G and H as in Claim 1. Claims 3 and 4 allow us to assume without loss of generality that there is no degree one vertex in G' and for all vertices u, $deg_{G'}(u) \leq deg_{H'}(u) + 1$. For convenience, we will refer to the edges of G' as **black edges** and edges of H' as **blue edges**.

▶ **Lemma 6.** *If there exist five vertices u, v, w_1, w_2, w_3 such that*
- *$(u, w_1), (w_1, w_2), (w_2, w_3)$ are black edges and $(v, w_1), (v, w_2)$ are **not** black edges.*
- *(u, v) is a blue edge and (u, w_2) is **not** a blue edge.*

Then, there exists a vector x such that $\mathcal{R}(H', x) > \mathcal{R}(G', x)$.

Proof. Let x be the vector with $x_u = 0$, x_v=2, $x_{w_1} = \frac{1}{3}$ and $x_{w_2} = \frac{2}{3}$, and $x_t = 1$ otherwise. We have

$$\begin{aligned}
\mathcal{R}(H', x) &\geq (x_u - x_v)^2 + \sum_{\substack{(u,a)\in E_{H'} \\ a \neq v}} (x_u - x_a)^2 + \sum_{\substack{(v,b)\in E_{H'} \\ b \neq u}} (x_v - x_b)^2 \\
&= 4 + (deg_{H'}(u) - 1) \cdot (0 - 1)^2 + (deg_{H'}(v) - 1) \cdot (2 - 1)^2 \\
&= deg_{H'}(u) + deg_{H'}(v) + 2 \geq deg_{G'}(u) + deg_{G'}(v) \qquad \text{(Claim 3)}
\end{aligned}$$

and

$$\begin{aligned}
\mathcal{R}(G', x) &= (x_u - x_{w_1})^2 + (x_{w_1} - x_{w_2})^2 + (x_{w_2} - x_{w_3})^2 + \sum_{\substack{(u,a)\in E_{G'} \\ a \neq w_1}} (x_u - x_a)^2 \\
&\quad + \sum_{\substack{(w_1,c)\in E_{G'} \\ c \neq w_2, u}} (x_{w_1} - x_c)^2 + \sum_{(v,b)\in E_{G'}} (x_v - x_b)^2 + \sum_{\substack{(w_2,d)\in E_{G'} \\ d \neq w_1, w_3}} (x_{w_2} - x_d)^2.
\end{aligned}$$

We observe that **(i)** The first three terms are equal to $\frac{1}{9}$. **(ii)** There is at most one edge (w_1, c) for $c \neq w_2, u$. Also, since w_1 is not incident to v, we have $x_c = 1$. Thus the fifth term is at most $\frac{4}{9}$. **(iii)** There is at most one edge (w_2, d) for $d \neq w_1, w_3$. In addition, G' is a subgrid, so there is no cycle of length 3 and w_2 is not incident to u. Also, w_2 is not incident to v, by assumption. So, it must be that $x_d = 1$ and the last term is at most equal to $\frac{1}{9}$. **(iv)** Since G' and H' do not share an edge, u is not connected to v. By assumption u is also not incident to w_2. So, it must be that $x_a = 1$ and the fourth term is equal to $deg_{G'}(u) - 1$. **(v)** Vertex v is not connected to u, w_1, w_2. Thus it must be $x_b = 1$ and the sixth term is equal to $deg_{G'}(v)$. Collecting the terms gives $\mathcal{R}(G, x) \leq deg_{G'}(u) + deg_{G'}(v) - \frac{1}{9}$ and the Lemma follows. ◀

▶ **Lemma 7.** *If there exist four different vertices u, v, w_1, w_2 such that*
- *w_1 has only two black adjacent edges (u, w_1) and (w_1, w_2)*
- *(u, v) is a blue edge.*

Then, there exists a vector x such that $\mathcal{R}(H', x) > \mathcal{R}(G', x)$.

Proof. Let x be a vector with $x_u = 0, x_v = 2, x_{w_1} = \frac{1}{2}$ and $x_t = 1$ otherwise. We have

$$\begin{aligned}
\mathcal{R}(H', x) &\geq (x_u - x_v)^2 + \sum_{\substack{(u,a)\in E_{H'} \\ a \neq v}} (x_u - x_a)^2 + \sum_{\substack{(v,b)\in E_{H'} \\ b \neq u}} (x_v - x_b)^2 \\
&= 4 + (deg_{H'}(u) - 1) \cdot (0 - 1)^2 + (deg_{H'}(v) - 1) \cdot (2 - 1)^2 \qquad \text{(no shared edges)} \\
&= deg_{H'}(u) + deg_{H'}(v) + 2 \geq deg_{G'}(u) + deg_{G'}(v) \qquad \text{(Claim 3)}
\end{aligned}$$

and

$$\mathcal{R}(G',x) = (x_u - x_{w_1})^2 + (x_{w_1} - x_{w_2})^2 + \sum_{\substack{(u,a)\in E_{G'} \\ a\neq w_1}} (x_u - x_a)^2 + \sum_{(v,b)\in E_{G'}} (x_v - x_b)^2$$

Since G' and H' do not share an edge, $(x_u - x_a)^2$ and $(x_v - x_b)^2$ terms are $(0-1)^2$ and $(2-1)^2$ respectively. We have

$$\mathcal{R}(G',x) = \frac{1}{4} + \frac{1}{4} + (deg_{G'}(u) - 1)\cdot 1 + deg_{G'}(v)\cdot 1 = deg_{G'}(u) + deg_{G'}(v) - \frac{1}{2}.$$

The Lemma follows. ◀

▶ **Lemma 8.** *If there exists a degree three vertex in G', then there exists a vector x such that $\mathcal{R}(H',x) > \mathcal{R}(G',x)$.*

Proof. Since $deg_{G'}(u) = 3, deg_{H'}(u) \geq 2$ by claim 3. Consider the underlying grid of which G is a subgraph. Pick the edge $(u,v) \in E_{H'}$ which is either not axis-parallel or is axis-parallel and v is at distance at least 2 in the grid. Since u has degree 3 in G', there exists a neighbor w_1 of u in G' such that any path from w_1 to v in G' has length at least 3.

If w_1 has degree 2 in G', then we set w_2 to be the neighbor of w_1 other than u. It is then straightforward to check that u, v, w_1, w_2 satisfy the condition of Lemma 7. Hence, there exists x such that $\mathcal{R}(H',x) > \mathcal{R}(G',x)$.

If w_1 is not of degree 2 in G', it must have degree 3 since there are no degree 1 vertices in G'. Let w_2 be the neighbor of w_1 other than u such that $(u,w_2) \notin E'_H$. Such a neighbor exists since u has at most one neighbor in H' other than v and v is not incident to w_1 due to the fact that any path of length from w_1 to v has length at least 3. Let w_3 be the neighbor of w_2 other than w_1. Such a neighbor must exist since there is no vertex of degree 1 in G'. Now, we prove that these vertices satisfy the condition of Lemma 6. By construction $(u,w_1),(w_1,w_2),(w_2,w_3)$ are black eges. Any path from w_1 to v in G' has length at least 3 which implies that $(w_1,v),(w_2,v)$ are not black edges. Also, by construction, (u,v) is a blue edge and (u,w_2) is not a blue edge. Hence, by lemma 6 there exists x such that $\mathcal{R}(H',x) > \mathcal{R}(G',x)$. ◀

▶ **Lemma 9.** *If there exists a set S of vertices such that no edges leave S in G', but at least one edge leaves S in H' then then there exists a vector $x \in \mathbb{R}^n$ such that $\mathcal{R}(H',x) > \mathcal{R}(G',x)$.*

Proof. Let x be defined as follows: $x_u = 1$ for $u \in S$ and $x_u = 0$ for $u \notin S$. The $\mathcal{R}(G',x)$ is equal to the number of edges leaving S in G', and similarly for H'. The lemma follows. ◀

▶ **Lemma 10.** *If there exists a cycle of length more than 4 in G' such that all vertices of cycle have degree 2 in G', then there exists $x \in \mathbb{R}^n$ such that $\mathcal{R}(H',x) > \mathcal{R}(G',x)$.*

Proof. Let C be the set of vertices in the cycle. For any vertex v in C, $deg_{G'}(v) = 2$. By claim 3, $deg_{H'}(v) \geq 1$. If there is no blue edge connecting two vertices of C in G', then there are at least $|C|$ edges going out of C in H' and no edge going out of C in G'. Then, by lemma 9, there exists x such that $\mathcal{R}(H',x) > \mathcal{R}(G',x)$.

In the complementary case, suppose there is an edge $(a,b) \in E_{H'}$ such that $a, b \in C$. Let the two paths from a to b on C be $P_1 = a, w_1, \ldots, w_{k_1}, b$ and $P_2 = a, v_1, \ldots, v_{k_2}, b$. Since G' and H' do not share any edge, $\min(k_1,k_2) \geq 1$. And since the cycle has length at least 5, $\max(k_1,k_2) \geq 2$. Let a vector x be defined as follows: $x_a = 0, x_b = 1, x_{w_i} = \frac{i}{k_1+1}, x_{v_i} = \frac{i}{k_2+1}$ and for $u \notin C$, set $x_u = 0$. We have

$$\begin{aligned}\mathcal{R}(G',x) &= \sum_{(u,v)\in P_1}(x_u-x_v)^2+\sum_{(u,v)\in P_2}(x_u-x_v)^2\\ &= (k_1+1)\cdot\frac{1}{(k_1+1)^2}+(k_2+1)\cdot\frac{1}{(k_2+1)^2}\leq\frac{5}{6}.\end{aligned}$$

The last inequality holds because $\min(k_1,k_2)\geq 1, \max(k_1,k_2)\geq 2$.
On the other hand $\mathcal{R}(H',x)\geq(x_a-x_b)^2=1$ and the lemma follows. ◀

▶ **Lemma 11.** *If G' contains a set of disjoint cycles of length 4 and H' edges only have endpoints on the same cycle, then there is a cycle $\tilde{H}$ such that $|E_{\tilde{H}}\cap E_G|=|E_H\cap E_G|+2$.*

Proof. Consider a cycle C of length 4 in G' such that there is no blue edge between a vertex in the cycle and a vertex not in the cycle. Since $deg_{G'}(v)\geq 2$ for all vertices in the cycle, $deg_{H'}(v)\geq 1$ by claim 3. Since G' and H' do not share any edge and the cycle has length 4, we must have $deg_{H'}(v)=1$ for all $v\in C$. And for vertices not in the length 4 cycles, $deg_{H'}(v)=0$. Let the edges of H be $F_1\cup F_2$ where F_1 are the edges shared between G and H and F_2 are the diagonal edges in the disjoint cycles of length 4. Let C be one such cycle in G' with vertices v_1,v_2,v_3,v_4 in this order and $(v_1,v_3)\in F_2,(v_2,v_4)\in F_2$. Let $H_1=(V,F_1\cup F_2\backslash\{(v_1,v_3),(v_2,v_4)\}\cup\{(v_1,v_2),(v_3,v_4)\}), H_2=(V,F_1\cup F_2\backslash\{(v_1,v_3),(v_2,v_4)\}\cup\{(v_1,v_4),(v_2,v_3)\})$. Then, one of the H_1 or H_2 is a cycle of length n. We let $\tilde{H}$ be that cycle. ◀

Finishing the proof: Recall that we have assumed that G does not contain a Hamiltonian cycle and let H is a permutation of C such that $|E_G\cap E_H|$ is maximized. To show that G does **not** dominate H, we need to construct a vector x such that $\mathcal{R}(H,x)>\mathcal{R}(G,x)$.

Starting with G and H, we form two graphs G' and H' as follows: (i) delete from G and H all common edges, (ii) iteratively and greedily delete all vertices such that $deg_G(u)=1, deg_H(u)=0$. Then Claims 1 and 2 show that it suffices to find a vector x such that $\mathcal{R}(H',x)>\mathcal{R}(G',x)$.

If there is still a vertex with $deg_{G'}(u)=1$, then $deg_{H'}(u)$ must be at least 1 and hence, by Claim 4, there exists a vector x such that $\mathcal{R}(H',x)>\mathcal{R}(G',x)$. Also, if there is a vertex u with degree 3 in G', then by lemma 8 there exists x such that $\mathcal{R}(H',x)>\mathcal{R}(G',x)$.

If there are no degree 1 or degree 3 vertices G', then G' must be a collection of isolated vertices and cycles. If there is a vertex v such that $deg_{G'}(v)=0, deg_{H'}(v)\geq 1$, then by setting $S=\{v\}$, lemma 9 implies that there exists a vector x such that $\mathcal{R}(H',x)>\mathcal{R}(G',x)$.

So, if none of the above cases occurs, then G' is a collection of disjoint cycles and H' edges are only incident to vertices of the cycles. If there is a cycle of length at least 5, then by lemma 10 there exists x such that $\mathcal{R}(H',x)>\mathcal{R}(G',x)$. Otherwise, if there is at least one blue edge with end points on two different cycles of length 4, then by setting S to be the vertex set of the cycle of length 4 Lemma 9 implies that there exists x such that $\mathcal{R}(H',x)>\mathcal{R}(G',x)$.

So, either G' and H' are empty or G' consists of a collection of disjoint cycles of length 4 such that blue edges have end points in the same cycle. In the first case, G trivially contains a Hamiltonian cycle since H' is empty. This is a contradiction to the assumption that G does not contain a Hamiltonian cycle. In the second case G Lemma 11 contradicts our assumption about the maximality of $|E_G\cap E_H|$. ◀

Proof. (Theorem 4) The problem of detecting if a cubic subgrid contains a Hamiltonian cycle is NP-complete [17]. Hence Theorem 5 is a direct reduction, and the theorem follows. ◀

3 Spectrally Robust Graph Isomorphism on Trees

This section outlines a proof for Theorem 3. We first introduce necessary notation.

▶ **Definition 12.** The support $\sigma(G, H)$ of G by H is the smallest number γ such that γH dominates G.

▶ **Definition 13.** The condition $\kappa(G, H)$ of a pair of graphs G and H is the smallest number κ such that G and H are κ-similar. We have $\kappa(G, H) = \sigma(G, H)\sigma(H, G)$.

We denote by $d_G(u, v)$ the distance between u and v in G using the shortest path metric. For $S \subseteq V$, we denote by $\delta_G(S)$ the set of of edges crossing the cut $(S, V - S)$ in G.

We now briefly review well-known facts [8]; a more detailed version of this paragraph along with proofs can be found in the appendix of the full version of the paper [12]. Given two trees G and H there is an obvious way to embed the edges of G into H: each edge (u, v) is routed over the unique path between vertices (u, v) in H. The *dilation* of the embedding is defined by: $\mathbf{d} = \max_{(u,v)\in E_G} d_H(u, v)$. The congestion c_e of an edge $e \in E_H$ is the number of G-edges that are routed over e. The *congestion* $\mathbf{c}$ of the embedding is then defined as $\max_{e\in E_H} c_e$. An upper bound of κ on the condition number implies the same upper bound on both c and d. On the other hand, the product $\mathbf{cd}$ is an upper bound on $\sigma(G, H)$, which is at most a quadratic over-estimation of $\sigma(G, H)$.

Our algorithm finds a mapping that controls both the dilation and the congestion of the embeddings from G to H and vice versa, thus obtaining a quadratic approximation to the condition number as a corollary.

▶ Remark. To simplify the presentation and the proof, we assume uniform upper bounds on the congestion and the dilation of both embeddings (G to H and H to G), rather than handling them separately. This formally proves a κ^3-approximation to κ-similarity. A κ-approximation algorithm may be possible with a refined argument that tracks the two embeddings separately.

Formally, our result can be stated as follows:

▶ **Theorem 14.** *Suppose G and H are two trees for which there exists a bijective mapping $\pi : V(G) \to V(H)$ satisfying the following properties:*

- *For all $(u, v) \in E(G)$, $d_H(\pi(u), \pi(v)) \leq \ell$*
- *For all $(u, v) \in E(H)$, $d_G(\pi^{-1}(u), \pi^{-1}(v)) \leq \ell$*
- *For $S \subset V(G)$ such that $|\delta_G(S)| = 1$, $|\delta_H(\pi(S))| \leq k$.*
- *For $S \subset V(H)$ such that $|\delta_H(S)| = 1$, $|\delta_G(\pi^{-1}(S))| \leq k$.*

Then, there exists an algorithm to find such a mapping in time $n^{O(k^2 d)}$ where d is the maximum degree of a vertex in G or H.

Our main result, theorem 3, follows immediately as a corollary from the fact that $\max\{k, l\} \leq \sigma(H, G) \leq kl$ (which is proved in the full version of the paper) and using the fact that $\max\{\sigma(G, H), \sigma(H, G)\} \leq \kappa(G, H) \leq \sigma(G, H)\sigma(H, G)$.

▶ **Corollary 15.** *Given two tree graphs G and H with condition number κ and maximum degree d, there exists an algorithm running in time $n^{O(\kappa^2 d)}$ which finds a mapping certifying that condition number is at most κ^4.*

The algorithm uses dynamic programming; it proceeds by recursively finding mappings for different subtrees and merging them. The challenge is to find partial mappings of subtrees which also map their boundaries in such a way that enables different mappings to

be appropriately merged. Notice that it is not enough to consider just the boundary vertices of the subtrees and their images. Instead, we need to additionally consider the boundary edges of those vertex sets, which correspond to cuts induced on the graph.

Definitions and Lemmas. To proceed with the proof, we introduce some definitions. We fix k and ℓ to be defined as in Theorem 14. Also, we fix an arbitrary ordering L on the edges of H. Without loss of generality it is convenient to root the trees such that we always map the two roots to each other. Let r_G be the root of tree G and r_H be the root of tree H.

Suppose that u is a vertex in G and T_u^G is the subtree rooted at u in G. If T_u^G is mapped to the set T in H, then its boundary includes the vertex u, the edge incident to u, the boundary vertices of T, and the cuts induced by edges going out of T. Hence, in addition to considering the mapping of boundary vertices, we need to consider the mapping of sets T' such that $\delta_H(T') = \{e\}$ where $e \in \delta_H(T)$. This notion is formalized in the following two definitions.

▶ **Definition 16.** Let Γ be the set of tuples $(u, T, v, u_1, \ldots, u_x, S_1, \ldots, S_x)$ satisfying the following properties:

- $u, u_1, \ldots, u_x \in V(G), v \in V(H), T \subset V(H), S_1, \ldots, S_x \subset V(G)$;
- $r_G \notin S_1, \ldots, S_x, r_H \notin T$;
- $u, u_1, \ldots, u_x \neq r_G, v \neq r_H$;
- $|\delta_H(T)| = x \leq k$ and $\forall j \in [1, x], |\delta_G(S_j)| \leq k$.

For $\alpha \in \Gamma$, we use the indicator variable z_α to denote if there is a mapping π which realizes α and preserves the distances and cuts for edges in T_u^G and T. A permutation π realizing α is formally defined below. Intuitively, this mapping maps the subtree rooted at u in G to the set T in H, vertex u to vertex v. It also maps $u_1, \ldots, u_x$ to the vertex boundary of the set T, and maps sets $S_1, \ldots, S_x$ to the cuts induced by the boundary edges of T. The formal definition of z_α is as follows:

▶ **Definition 17.** For $\alpha = (u, T, v, u_1, \ldots, u_x, S_1, \ldots, S_x) \in \Gamma$, let $\delta_H(T) = \{e_1, \ldots, e_x\}$ be such that for $i < j$, e_i is ordered before e_j in ordering L. Let $v_j = e_j \cap T$, T_u^G be the vertex set in the subtree rooted at u in G and for $e \in E(G)$, let T_e^G be the vertex set in the subtree under edge e. Formally speaking $T_e^G \subset V(G)$ such that $\delta_G(T_e^G) = \{e\}$ and $r_G \notin T_e^G$ (T_e^H is similarly defined). We define $z_\alpha = 1$ if there exists a mapping $\pi : V(G) \to V(H)$ such that:

1. $\pi(T_u^G) = T, \pi(u) = v, \forall j \in [1, x], \pi(u_j) = v_j, \pi(S_j) = T_{e_j}^H$, $\pi(r_G) = r_H$.
2. $\forall (u, v) \in E[G[T_u^G]], d_H(\pi(u), \pi(v)) \leq \ell$
3. $\forall (u, v) \in E[H[T]], d_G(\pi^{-1}(u), \pi^{-1}(v)) \leq \ell$
4. $\forall e \in E[G[T_u^G]]$, $|\delta_H(\pi(T_e^G)| \leq k$.
5. $\forall e \in E[H[T]]$, $|\delta_G(\pi^{-1}(T_e^H)| \leq k$.

We refer to such a mapping π as a certificate of $z_\alpha = 1$. Moreover, we define $z_{\alpha,\pi} = 1$ if π is a certificate of $z_\alpha = 1$ and 0 otherwise.

▶ **Claim 5.** *There exists a poly(n) time algorithm which given $\alpha \in \Gamma, \pi : V(G) \to V(H)$, outputs the value of $z_{\alpha,\pi}$.*

Our goal is to design an algorithm which computes z_α for every $\alpha \in \Gamma$. However, for our algorithm to run in polynomial time, we need Γ to not be exponentially large.

▶ **Lemma 18.** $|\Gamma| \leq n^{O(k^2)}$.

Proof. Let $\alpha = (u, T, v, u_1, \ldots, u_x, S_1, \ldots, S_x)$. We prove the lemma by bounding the number of choices for each parameter.

- The number of choices of u is upper bounded by n.
- Since $|\delta_H(T)| = x$, the number of choices $\delta_H(T)$ is upper bounded by $\binom{m}{x}$ where m is the number of edges. By substituting $m = n - 1$, we get that the number of different $\delta_H(T)$, i.e. the number of different T's, is upper bounded by $\binom{n-1}{x}$.
- The number of different v and u_j is at most n for each $j \in [1, x]$.
- Similarly to the argument for T, the number of different S_j with $|\delta_H(S_j)| \leq k$ is at most $\sum_{t=1}^{k} \binom{n-1}{t}$.

For $x \in [1, k]$, the number of different tuples α in Γ with $|\delta_H(T)| = x$ is at most:

$$n \cdot \binom{n-1}{x} n \cdot n^x \cdot \left[\sum_{t=1}^{k} \binom{n-1}{t}\right]^x = n^{O(k \cdot x)}.$$

Since $x \leq k$, this gives us an upper bound of $n^{O(k^2)}$ on $|\Gamma|$. ◀

Suppose π is the optimal mapping from G to H which yields a mapping with cut distortion k and distance distortion ℓ and also certifies $z_\alpha = 1$ for some α. Our recursive algorithm does not necessarily obtain the same certificate as π for $z_\alpha = 1$. So, before we show how to compute z_α, we examine certain properties of z_α. In particular, we start by proving that if both π and γ certify $z_\alpha = 1$ so that $z_{\alpha,\pi} = z_{\alpha,\gamma} = 1$, then they not only match on the boundary vertices but also on the cuts induced by boundary edges.

▶ **Lemma 19.** *For $\alpha = (u, T, v, u_1, \ldots, u_x, S_1, \ldots, S_x)$, let π and γ be two mappings such that $z_{\alpha,\pi} = z_{\alpha,\gamma} = 1$. Then:*

1. $\pi(u) = \gamma(u)$.
2. $\pi(T_u^G) = \gamma(T_u^G)$.
3. *For every boundary vertex w of T (in T with an incident edge in $\delta_H(T)$), $\pi^{-1}(w) = \gamma^{-1}(w)$. Equivalently, $\pi(u_j) = \gamma(u_j)$ for $j \in [1, x]$.*
4. *For every edge $e \in \delta_H(T)$, $\pi^{-1}(T_e^H) = \gamma^{-1}(T_e^H)$.*
5. *For every connected component C in $H \setminus \delta_H(T)$, $\pi^{-1}(C) = \gamma^{-1}(C)$.*

Proof. Items 1-4. follow directly from the definition of $z_{\alpha,\pi}$. Consider a connected component C in $H \setminus \delta_H(T)$. Let $\delta_H(C) = \{e_{i_1}, \ldots, e_{i_t}\}$. Without loss of generality, assume that e_{i_1} is the edge closest to the root r_H. Then:

$$\gamma^{-1}(C) = \gamma^{-1}(T_{e_{i_1}}) \setminus \cup_{j=2}^{t} \gamma^{-1}(T_{e_{i_j}}).$$

Item 3 implies that $\gamma^{-1}(T_{e_{i_j}}) = \pi^{-1}(T_{e_{i_j}})$ for $j \in [1, t]$ thus proving $\pi^{-1}(C) = \gamma^{-1}(C)$. ◀

The next lemma is somewhat like a converse of the previous lemma. It shows that if we have a mapping π such that $z_{\alpha,\pi} = 1$ and another mapping γ such that γ matches with π on the subtree and the boundary vertices and edges, then $z_{\alpha,\gamma} = 1$ as well.

▶ **Lemma 20.** *Let $\alpha = (u, T, v, u_1, \ldots, u_x, S_1, \ldots, S_x)$ and $\pi : V(G) \to V(H)$ be such that $z_{\alpha,\pi} = 1$. Let $\gamma : V(G) \to V(H)$ be such that*

1. $\gamma(w) = \pi(w)$ *for* $w \in T_u^G$
2. $\gamma(u_j) = \pi(u_j)$ *for* $j \in [1, x]$
3. $\gamma(S_j) = \pi(S_j)$ *for* $j \in [1, x]$ *(π and γ may not be identical on every element of S_j)*

Then, $z_{\alpha,\gamma} = 1$.

Proof. Follows immediately from definition 17. ◀

Next we show how to change the optimal mapping such that it agrees with the mapping found by our algorithm on the subtree and is still optimal. Following lemma formalizes this statement:

▶ **Lemma 21.** *Let π be a mapping such that $z_{\alpha,\pi} = z_{\alpha_1,\pi} = 1$ where*

$$\alpha = (a, T, b, u_1, \ldots, u_x, S_1, \ldots, S_x) \in \Gamma \text{ and } \alpha_1 = (a^1, T^1, b^1, u_1^1, \ldots, u_{x_1}^1, S_1^1, \ldots, S_{x_1}^1) \in \Gamma$$

such that a_1 is a child of a in G. Suppose, γ_1 is another mapping such that $z_{\alpha_1,\gamma_1} = 1$. Let ζ be defined as follows: $\zeta(u) = \gamma_1(u)$ for $u \in T_{a^1}^G$ and $\zeta(u) = \pi(u)$ otherwise. Then, $z_{\alpha,\zeta} = 1$.

Proof. Items 1-4 of lemma 17 can be easily verified for $z_{\alpha,\zeta}$. Here, we prove that the following property holds: $\forall e \in E[H[T]], |\delta_G(\zeta^{-1}(T_e^H))| \leq k$.

There are three possible cases, that we consider separately.

1. $e = (b, b_1)$, $e \in \delta_H(T^1)$.
   By definition, $\zeta^{-1}(T_e^H) = \gamma_1^{-1}(T_e^H)$. Since, $z_{\alpha_1,\gamma_1} = 1$, we get $|\zeta^{-1}(T_e^H)| = |\gamma_1^{-1}(T_e^H)| \leq k$.
2. $e \in E[H[T \setminus T^1]]$.
   By definition, $\zeta^{-1}(T_e^H) = \pi^{-1}(T_e^H)$. Since, $z_{\alpha,\pi} = 1$,, we get $|\zeta^{-1}(T_e^H)| = |\pi^{-1}(T_e^H)| \leq k$.
3. $e \in E[H[T^1]]$.
   By definition, $\zeta^{-1}(T_e^H) = \gamma_1^{-1}(T_e^H)$ and since, $z_{\alpha_1,\gamma_1} = 1$, we get $|\zeta^{-1}(T_e^H)| \leq k$. ◀

The above lemmas show that even if we find mappings for subtrees which are different from the optimal mappings, they can still be merged with the optimal mappings. Hence, we may just find any of the mappings for each α and then recursively combine mappings. The following lemma states the result and shows how to make it constructive:

▶ **Lemma 22.** *Let $a \in V(G)$ be a vertex in G with children $a^1, \ldots, a^t$. Let*

$$\alpha = (a, T, b, u_1, \ldots, u_x, S_1, \ldots, S_x) \text{ and}$$
$$\alpha_1 = (a^1, T^1, b^1, u_1^1, \ldots, u_{x_1}^1, S_1^1, \ldots, S_{x_1}^1), \ldots, \alpha_t = (a^t, T^t, b^t, u_1^t, \ldots, u_{x_t}^t, S_1^t, \ldots, S_{x_t}^t) \in \Gamma.$$

Let $\pi : V(G) \to V(H)$ be a mapping such that $z_{\alpha,\pi} = z_{\alpha_1,\pi} = \cdots = z_{\alpha_j,\pi} = 1$ for all $j \in [1, t]$. If for each $j \in [1, t]$ there exists a mapping $\gamma_j : V(G) \to V(H)$ with $z_{\alpha_j,\gamma_j} = 1$, then there exists $\pi' : V(G) \to V(H)$ such that $z_{\alpha,\pi'} = 1$ and $\pi'(w) = \gamma_j(w)$ for $w \in T_{a_j}^G$ where $j \in [1, t]$. Moreover, given $\{\gamma_j \mid j \in [1, t]\}$, such π' can be found in time poly(n).

Proof. Let $\zeta : V(G) \to V(H)$ such that $\zeta(w) = \gamma_j(w)$ for $w \in T_{a_j}^G$ where $j \in [1, t]$ and $\zeta(w) = \pi(w)$ otherwise. By lemma 21, $z_{\alpha,\zeta} = 1$.

Construction of π': Let $\pi'(w) = \gamma_j(w)$ for $w \in T_{a_j}^G, j \in [1, t]$ and $\pi'(a) = b$. For $w \notin T_a^G$, define π' such that $\pi'(S_j) = T_j$ for $j \in [1, x]$. Setting $\pi = \zeta, \gamma = \pi'$ in lemma 20, we get $z_{\alpha,\pi'} = 1$. Easy to see that π' is constructed in polynomial time. ◀

Lemma 22 suggests that we can recursively compute z_α. Namely, we can show the following:

▶ **Lemma 23.** *There exists an algorithm with running time poly$(n, |\Gamma|^d)$ which calculates z_α for each $\alpha \in \Gamma$. Additionally if $z_\alpha = 1$, it also computes π_α such that $z_{\alpha,\pi_\alpha} = 1$.*

Proof. Consider $\alpha = (a, T, b, u_1, \ldots, u_x, S_1, \ldots, S_x) \in \Gamma$ with $z_\alpha = 1$ and $\pi : V(G) \to V(H)$ be the mapping such that $z_{\alpha,\pi} = 1$. Let the children of a be $a^1, \ldots, a^t$.

▶ **Claim 6.** *For $j \in [1, t], \exists \alpha_j = (a^j, T^j, b^j, u_j^1, \ldots, u_{x_j}^1, S_1^1, \ldots, S_{x_j}^1) \in \Gamma$ such that $z_{\alpha_j,\pi} = 1$.*

To construct a mapping π' such that $z_{\alpha,\pi'} = 1$, we guess $\alpha_1, \ldots, \alpha_t$ and use lemma 22 to construct such a mapping. It requires mapping γ_j such that $z_{\alpha_j,\gamma_j} = 1$ which can be assumed to be constructed recursively. The number of choices of $\alpha_1, \ldots, \alpha_t$ is upper bounded by $|\Gamma|^t$ which is upper bounded by $|\Gamma|^d$, as the degree of any vertex is at most d. For any such choice, algorithm in lemma 22 runs in time $poly(n)$. Hence, computing z_α takes $|\Gamma|^d poly(n)$ time for each α and $|\Gamma|^{d+1} poly(n) = poly(n, |\Gamma|^d)$ time for all $\alpha \in \Gamma$.

If $z_\alpha = 0$, then for any of the mappings π' considered above has $z_{\alpha,\pi'} = 0$. This can be checked in $poly(n)$ time for each π' (proposition 5). ◀

Proof. (of theorem 14) Let π be a mapping $\pi : V(G) \to V(H)$ such that:

(a) For all $(u, v) \in E(G)$, $d_H(\pi(u), \pi(v)) \leq \ell$

(b) For all $(u, v) \in E(H)$, $d_G(\pi^{-1}(u), \pi^{-1}(v)) \leq \ell$

(c) For $S \subset V(G)$ s.t. $|\delta_G(S)| = 1$, $|\delta_H(\pi(S))| \leq k$.

(d) For $S \subset V(H)$ s.t. $|\delta_H(S)| = 1$, $|\delta_G(\pi^{-1}(S))| \leq k$.

First, we start by guessing the roots of G and H and define Γ. Then, using lemma 23, we can calculate z_α for $\alpha \in \Gamma$. It does not give us a mapping π' satisfying the conditions above since for $\alpha = (u, T, v, u_1, \ldots, u_x, S_1, \ldots, S_x) \in \Gamma$, we have $u \neq r_G$. However, a proof almost identical to that of lemma 23 works here as well. Assume r_G has children $a_1, \ldots, a_t$. Then there exists $\alpha_j = (a^j, T^j, b^j, u^1_j, \ldots, u^1_{x_j}, S^1_1, \ldots, S^1_{x_j}) \in \Gamma$ such that $z_{\alpha_j,\pi} = 1$. Then, similarly to the proof of lemma 23 we can guess $\alpha_j, j \in [1, t]$ and compute π' in time $poly(n) \cdot |\Gamma|^d$, which satisfies the conditions described above. ◀

4 Final Remarks

From an algebraic standpoint, the problems we considered in this work have natural generalizations to pairs of positive definite matrices (A, B), and the corresponding eigenvalue problem $Ax = \lambda P^T BPx$. SGD generalizes to minimizing the maximum eigenvalue, and SGRI generalizes to finding the permutation P that minimizes the condition number $\kappa(A, B)$. But the problem appears to be much harder in some sense: one can construct 'pathological' examples of A and B with just two distinct eigenspaces that are nearly identical, but different enough to cause unbounded condition numbers due to the eigenvalue gap. This makes implausible the existence of non-trivial subexponential time algorithms for the general case.

On the other hand, besides their potential for applications, Laplacians seem to offer an interesting mathematical ground with a wealth of open problems. In this paper we presented the first algorithmic result, for unweighted trees. The algorithm is admittedly complicated, but it can at least be viewed as an indication of algorithmic potential, as we are not aware of any fact that would preclude a similar approximation for general graphs. To make such algorithmic progress, we would likely have to give up on the combinatorial interpretations of the condition number, and use deeper spectral properties of Laplacians.

References

1 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 684–697, 2016.

2 Joshua Batson, Daniel A. Spielman, Nikhil Srivastava, and Shang-Hua Teng. Spectral sparsification of graphs: Theory and algorithms. *Commun. ACM*, 56(8):87–94, 2013. doi:10.1145/2492007.2492029.

3 Adolfo Piperno Brendan D. Mckay. Faster isomorphism testing of strongly regular graphs. *Journal of Symbolic Computation archive*, 60:94–112, 2014.

4 P. G. Doyle and J. L. Snell. Random Walks and Electric Networks. *ArXiv Mathematics e-prints*, 2000. `arXiv:math/0001057`.

5 Frank Emmert-Streib, Matthias Dehmer, and Yongtang Shi. Fifty years of graph matching, network alignment and network comparison. *Inf. Sci.*, 346(C):180–197, 2016. `doi:10.1016/j.ins.2016.01.074`.

6 Soheil Feizi, Gerald Quon, Mariana Recamonde Mendoza, Muriel Médard, Manolis Kellis, and Ali Jadbabaie. Spectral alignment of networks. *CoRR*, abs/1602.04181, 2016. `arXiv:1602.04181`.

7 S. Fortin. The graph isomorphism problem. *Technical Report 96-20*, 1996.

8 Stephen Guattery and Gary L. Miller. Graph embeddings and laplacian eigenvalues. *SIAM J. Matrix Analysis Applications*, 21(3):703–723, 2000.

9 Israel Hanukoglu. Electron transfer proteins of cytochrome p450 systems. In E. Edward Bittar, editor, *Physiological Functions of Cytochrome P450 in Relation to Structure and Regulation*, volume 14 of *Advances in Molecular and Cell Biology*, pages 29–56. Elsevier, 1996. `doi:10.1016/S1569-2558(08)60339-2`.

10 Claire Kenyon, Yuval Rabani, and Alistair Sinclair. Low distortion maps between point sets. In *SIAM J. Comput*, volume 39(4), page 1617–1636, 2004.

11 J. Kobler, U. Schoning, and J. Toran. The graph isomorphism problem: Its structural complexity. *Progress in Theoretical Computer Science*, 1993.

12 Alexandra Kolla, Ioannis Koutis, Vivek Madan, and Ali Kemal Sinop. Spectrally robust graph isomorphism. *CoRR*, abs/1805.00181, 2018. URL: `https://arxiv.org/abs/1805.00181`.

13 Ioannis Koutis. *Combinatorial and algebraic tools for optimal multilevel algorithms*. PhD thesis, Carnegie Mellon University, Pittsburgh, May 2007. CMU CS Tech Report CMU-CS-07-131.

14 Ioannis Koutis, Gary L. Miller, and Richard Peng. A fast solver for a class of linear systems. *Commun. ACM*, 55(10):99–107, 2012. `doi:10.1145/2347736.2347759`.

15 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982.

16 Ryan O'Donnell, John Wright, Chenggang Wu, and Yuan Zhou. Hardness of robust graph isomorphism, lasserre gaps, and asymmetry of random graphs. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1659–1677, 2014.

17 Christos H Papadimitriou and Umesh V Vazirani. On two geometric problems related to the travelling salesman problem. *Journal of Algorithms*, 5(2):231–246, 1984.

18 Robert Patro and Carl Kingsford. Global network alignment using multiscale spectral signatures. *Bioinformatics*, 28(23):3105–3114, 2012. `doi:10.1093/bioinformatics/bts592`.

19 Daniel A. Spielman. Faster isomorphism testing of strongly regular graphs. *Proc. 28th ACM STOC*, 215:576–584, 1995.

20 Charalampos E. Tsourakakis. Toward quantifying vertex similarity in networks. *Internet Mathematics*, 10(3-4):263–286, 2014. `doi:10.1080/15427951.2013.836581`.

21 Regina I. Tyshkevich Viktor N. Zemlyachenko, Nikolai M. Korneenko. Graph isomorphism problem. *Zapiski Nauchnykh Seminarov LOMI*, 215:83–158, 1982.

A Parameterized Strongly Polynomial Algorithm for Block Structured Integer Programs

Martin Koutecký[1]
Technion – Israel Institute of Technology, Haifa, Israel, and
Charles University, Prague, Czech Republic
koutecky@technion.ac.il
https://orcid.org/0000-0002-7846-0053

Asaf Levin[2]
Technion – Israel Institute of Technology, Haifa, Israel
levinas@ie.technion.ac.il

Shmuel Onn[3]
Technion – Israel Institute of Technology, Haifa, Israel
onn@ie.technion.ac.il

Abstract

The theory of n-fold integer programming has been recently emerging as an important tool in parameterized complexity. The input to an n-fold integer program (IP) consists of parameter A, dimension n, and numerical data of binary encoding length L. It was known for some time that such programs can be solved in polynomial time using $O(n^{g(A)}L)$ arithmetic operations where g is an exponential function of the parameter. In 2013 it was shown that it can be solved in fixed-parameter tractable time using $O(f(A)n^3L)$ arithmetic operations for a single-exponential function f. This, and a faster algorithm for a special case of *combinatorial* n-fold IP, have led to several very recent breakthroughs in the parameterized complexity of scheduling, stringology, and computational social choice. In 2015 it was shown that it can be solved in strongly polynomial time using $O(n^{g(A)})$ arithmetic operations.

Here we establish a result which subsumes all three of the above results by showing that n-fold IP can be solved in strongly polynomial fixed-parameter tractable time using $O(f(A)n^6 \log n)$ arithmetic operations. In fact, our results are much more general, briefly outlined as follows.

- There is a strongly polynomial algorithm for integer linear programming (ILP) whenever a so-called Graver-best oracle is realizable for it.
- Graver-best oracles for the large classes of multi-stage stochastic and tree-fold ILPs can be realized in fixed-parameter tractable time. Together with the previous oracle algorithm, this newly shows two large classes of ILP to be strongly polynomial; in contrast, only few classes of ILP were previously known to be strongly polynomial.
- We show that ILP is fixed-parameter tractable parameterized by the largest coefficient $\|A\|_\infty$ and the primal or dual treedepth of A, and that this parameterization cannot be relaxed, signifying substantial progress in understanding the parameterized complexity of ILP.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases integer programming, parameterized complexity, Graver basis, n-fold integer programming

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.85

[1] Supported by a Technion postdoctoral fellowship and the project 17-09142S of GA ČR.
[2] Supported by a grant from the GIF, the German-Israeli Foundation for Scientific Research and Development (grant number I-1366-407.6/2016).
[3] Supported by the Dresner chair.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 85; pp. 85:1–85:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version A full version of the paper is available at `https://arxiv.org/abs/1802.05859`.

1 Introduction

In this article we consider the general linear integer programming (ILP) problem in standard form,

$$\min\{\mathbf{wx} \mid A\mathbf{x} = \mathbf{b}\,,\ \mathbf{l} \le \mathbf{x} \le \mathbf{u}\,,\ \mathbf{x} \in \mathbb{Z}^n\}\,. \tag{ILP}$$

with A an integer $m \times n$ matrix, $\mathbf{b} \in \mathbb{Z}^m$, $\mathbf{w} \in \mathbb{Z}^n$, $\mathbf{l}, \mathbf{u} \in (\mathbb{Z} \cup \{\pm\infty\})^n$. It is well known to be strongly NP-hard, which motivates the search for tractable special cases.

The first important special case is ILP in fixed dimension. In the '80s it was shown by Lenstra and Kannan [17, 20] that (ILP) can be solved in time $n^{O(n)}L$, where L is the length of the binary encoding of the input. Secondly, it is known that if the matrix A is totally unimodular (all subdeterminants between -1 and 1), all vertices of the feasible region are integral and thus applying any polynomial algorithm for linear programming (LP) suffices. Later, Veselov and Chirkov [25] have shown that the more general class of bimodular ILP is also polynomial-time solvable. Other results exploit certain structural properties of A. These include the large classes of n-fold [13], tree-fold [4], 2-stage and multi-stage stochastic [3], and 4-block n-fold [12] ILPs, as well as algorithms for ILPs with bounded treewidth [11], treedepth [10] and fracture number [7] of certain graphs related to the matrix A.

A fundamental question regarding problems involving large numbers is whether there exists an algorithm whose number of arithmetic operations does not depend on the length of the numbers involved; if this number is polynomial, this is a *strongly polynomial algorithm* [24]. For example, the ellipsoid method or the interior-point method which solve LP take time which does depend on the encoding length, and the existence of a strongly polynomial algorithm for LP remains a major open problem. So far, the only strongly polynomial ILP algorithms we are aware of exist for totally unimodular ILP [14], bimodular ILP [2], so-called binet ILP [1], and n-fold IP with constant block dimensions [6]. All remaining results, such as Lenstra's famous algorithm or the fixed-parameter tractable algorithm for n-fold IP which has recently led to several breakthroughs [4, 16, 18, 19], are not strongly polynomial.

1.1 Our Contributions

To clearly state our results we introduce the following terminology. The input to a problem will be partitioned into three parts (α, β, γ), where α is the *parametric input*, β is the *arithmetic input*, and γ is the *numeric input*. A *strongly fixed-parameter tractable* (FPT) *algorithm* for the problem is one that solves it using $f(\alpha)\text{poly}(\beta)$ arithmetic operations and $g(\alpha)\text{poly}(\beta, \gamma)$ time, where f, g are some computable functions. If such an algorithm exists, we say that the problem is *strongly fixed-parameter tractable* (FPT) *parameterized by* α. Thus, such an algorithm both demonstrates that the problem is FPT *parameterized by* α because it runs in FPT time $g(\alpha)\text{poly}(\beta, \gamma)$, and provides a strongly polynomial algorithm for each fixed α. Having multiple parameters $\alpha_1, \ldots, \alpha_k$ simultaneously is understood as taking the *aggregate parameter* $\alpha = \alpha_1 + \cdots + \alpha_k$. If the algorithm involves oracles then the oracle queries are also counted as arithmetic operations and the answers to oracle queries should be polynomial in (β, γ). Each part of the input may have several entities, which may be presented in unary or binary, where $\langle e \rangle$ denotes the encoding length of an entity e presented in binary. For the parametric input the distinction between unary and binary is irrelevant.

ILP abounds in natural parameters: the dimension n, number of inequalities m, largest coefficient $\|A\|_\infty$, largest right-hand side $\|\mathbf{b}\|_\infty$, various structural parameters of A, etc. Here, we are interested in algorithms which are both strongly polynomial *and* FPT.

Recently it was shown that, if we have access to the so-called *Graver basis* of A, the problem (ILP) is polynomial time solvable even for various nonlinear objective functions [5, 21]. We show that all of these results can be extended to be strongly polynomial with only $\langle A \rangle$ as the arithmetic input.

▶ **Theorem 1.** *The problem* (ILP) *with arithmetic input* $\langle A \rangle$ *and numeric input* $\langle \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u} \rangle$*, endowed with a Graver-best oracle for* A*, is solvable by a strongly polynomial oracle algorithm.*

The existence of Graver-best oracles is thus of prime interest. We show such oracles for the wide classes of multi-stage stochastic and tree-fold ILPs; for precise definitions of these classes cf. Section 3.1.2. See Table 1 for a summary of improvements over the current state of the art.

▶ **Theorem 2.** *Multi-stage stochastic ILP with blocks* $B_1, \ldots, B_\tau$*,* $B_i \in \mathbb{Z}^{l \times n_i}$*, is strongly* FPT *parameterized by* $l + n_1 + \cdots + n_\tau$ *and* $\|A\|_\infty$*.*

▶ **Theorem 3.** *Tree-fold ILP with blocks* $A_1, \ldots, A_\tau$*,* $A_i \in \mathbb{Z}^{r_i \times t}$*, is strongly* FPT *parameterized by* $r_1 + \cdots + r_\tau$ *and* $\|A\|_\infty$*.*

This improves on the algorithm for tree-fold ILP [4] not only by making it strongly FPT, but also by leaving the block length t out of the parameter. Similarly, the following algorithm for the special case of n-fold ILP greatly improves both on the previous results of Hemmecke et al. [13] and Knop et al. [18] and is the currently fastest algorithm for this problem:

▶ **Theorem 4.** *n-fold ILP with blocks* $A_1 \in \mathbb{Z}^{r \times t}$ *and* $A_2 \in \mathbb{Z}^{s \times t}$ *can be solved in time* $a^{O(r^2 s + r s^2)}(nt)^6 \log(nt) + \mathcal{L}(\langle A \rangle)$*, where* $\mathcal{L}(\langle A \rangle)$ *is the runtime of a strongly polynomial LP algorithm.*

Next, we turn our attention to structural parameters of the constraint matrix A. We focus on two graphs which can be associated with A:

- the *primal graph* $G_P(A)$, which has a vertex for each column and two vertices are connected if there exists a row such that both columns are non-zero, and,
- the *dual graph* $G_D(A) = G_P(A^\intercal)$, which is the above with rows and columns swapped.

Two standard parameters of structural sparsity are the *treewidth* (measuring the "tree-likeness" of a graph) and the more restrictive *treedepth* (measuring its "star-likeness"). We denote the treewidth of $G_P(A)$ and $G_D(A)$ by $\mathrm{tw}_P(A)$ and $\mathrm{tw}_D(A)$; for treedepth we have $\mathrm{td}_P(A)$ and $\mathrm{td}_D(A)$. Note that bounded treedepth implies bounded treewidth but not vice versa.

We show that ILP parameterized by $\mathrm{td}_P(A) + \|A\|_\infty$ and $\mathrm{td}_D(A) + \|A\|_\infty$ can be reduced to the previously mentioned classes, respectively, implying (ILP) with these parameters is strongly FPT.

▶ **Theorem 5.** (ILP) *is strongly* FPT *parameterized by* $\mathrm{td}_P(A)$ *and* $\|A\|_\infty$*.*

This improves in two ways upon the result of Ganian and Ordyniak [10] who show that (ILP) with $\mathbf{w} \equiv \mathbf{0}$ (i.e. deciding the feasibility) is FPT parameterized by $\mathrm{td}_P(A) + \|A, \mathbf{b}\|_\infty$ [10]. First, we use the smaller parameter $\|A\|_\infty$ instead of $\|A, \mathbf{b}\|_\infty$, and second, we solve not only the feasibility but also the optimization problem. An analogous result holds for the parameter $\mathrm{td}_D(A)$, for which previously nothing was known at all.

▶ **Theorem 6.** (ILP) *is strongly* FPT *parameterized by* $\mathrm{td}_D(A)$ *and* $\|A\|_\infty$*.*

Table 1 Run time improvements implied by this paper. We denote by L the binary length of the numeric input $\mathbf{b}, \mathbf{l}, \mathbf{u}, \mathbf{w}$, i.e., $L = \langle \mathbf{b}, \mathbf{l}, \mathbf{u}, \mathbf{w} \rangle$, and consider $\langle A \rangle$ to be part of the arithmetic input. We denote by $a = \max\{2, \|A\|_\infty\}$, by r, s, t the relevant block dimensions (cf. Section 3.1), and by $\mathcal{L}(\langle A \rangle)$ the runtime of a strongly polynomial LP algorithm [24].

Type of instance	Previous best run time	Our result
n-fold ILP	$a^{O(rst+st^2)}n^3L$ [13]	$a^{O(r^2s+sr^2)}(nt)^6 \log(nt) + \mathcal{L}(\langle A \rangle)$ Thm 4
n-fold ILP	$n^{f_1(a,r,s,t)}$ [6]	
n-fold ILP	$t^{O(r)}(ar)^{r^2}n^3L$ if $A_2 = (1\ 1\ \cdots\ 1)$ [18]	$a^{O(r^2)}(nt)^6 \log(nt) + \mathcal{L}(\langle A \rangle)$ Thm 4
tree-fold ILP	$f_{\text{tf}'}(a, n_1, \ldots, n_\tau, t)n^3L$ [4]	$f_{\text{tf}}(a, n_1, \ldots, n_\tau)(nt)^3 + \mathcal{L}(\langle A \rangle)$ Thm 3
Multi-stage stochastic ILP	$f_{\text{mss}}(a, n_1, \ldots, n_\tau, l)n^3L$ [3]	$f_{\text{mss}}(a, n_1, \ldots, n_\tau, l)n^3 + \mathcal{L}(\langle A \rangle)$ Thm 2
Bounded dual treedepth	Open whether fixed-parameter tractable	$f_D(a, \text{td}_D(A)(nt)^3 + \mathcal{L}(\langle A \rangle)$ Thm 6
Bounded primal treedepth	$f_{P'}(a, \|\mathbf{b}\|_\infty, \text{td}_P(A))nL$ [10]	$f_P(a, \text{td}_P(A))n^3 + \mathcal{L}(\langle A \rangle)$ Thm 5

We emphasize that the parameterizations cannot be relaxed neither from treedepth to treewidth, nor by removing the parameter $\|A\|_\infty$: (ILP) is NP-hard already on instances with $\text{tw}_P(A) = 3$ and $\|A\|_\infty = 2$ [10, Thm 12], and it is strongly W[1]-hard parameterized by $\text{td}_P(A)$ alone [10, Thm 11]; the fact that a problem is W[1]-hard is strong evidence that it is not FPT. Similarly, deciding feasibility is NP-hard on instances with $\text{tw}_D(A) = 3$ and $\|A\|_\infty = 2$ (Lemma 18) and strongly W[1]-hard parameterized by $\text{td}_D(A)$ alone [19, Thm 5].

1.2 Interpretation of Results

We believe our approach also leads to several novel insights. First, we make it clear that the central question is finding Graver-best oracles; provided these oracles, Theorem 1 shows that tasks such as optimization and finding initial solutions can be handled under very mild assumptions. Even though we show these tasks are routine, they have been reimplemented repeatedly [4, 12, 13, 18].

Second, we show that the special classes of highly uniform block structured ILPs, namely multi-stage stochastic and tree-fold ILPs, are in some sense *universal* for all ILPs of bounded primal or dual treedepth, respectively. Specifically, we show that any ILP with bounded primal or dual treedepth can be embedded in an equivalent multi-stage stochastic or tree-fold ILP, respectively (Lemmas 25 and 26).

Third, we show that, besides bounded primal or dual treedepth, the crucial property for efficiency is the existence of augmenting steps with bounded ℓ_∞- or ℓ_1-norms, respectively (Lemmas 19 and 21). This suggests that for ILPs whose primal or dual graph is somehow "sparse" and "shallow", finding augmenting steps of bounded ℓ_∞- or ℓ_1-norm might be both sufficient for reaching the optimum and computationally efficient.

1.3 Related Work

We have already covered all relevant work regarding strongly polynomial algorithms for ILP.

Let us focus on structural parameterizations. It follows from Freuder's algorithm [9] and was reproven by Jansen and Kratsch [15] that (ILP) is FPT parameterized by $\text{tw}_P(A)$ and the largest domain $\|\mathbf{u} - \mathbf{l}\|_\infty$. Regarding the dual graph $G_D(A)$, the parameters $\text{td}_D(A)$ and $\text{tw}_D(A)$ were only recently considered by Ganian et al. [11]. They show that even deciding feasibility of (ILP) is NP-hard on instances with $\text{tw}_I(A) = 3$ ($\text{tw}_I(A)$ denotes the treewidth of the *incidence graph*; $\text{tw}_I(A) \leq \text{tw}_D(A) + 1$ always holds) and $\|A\|_\infty = 2$ [11, Theorem 12].

Furthermore, they show that (ILP) is FPT parameterized by $\mathrm{tw}_I(A)$ and parameter Γ, which is an upper bound on any prefix sum of $A\mathbf{x}$ for any feasible solution $\mathbf{x}$.

Dvořák et al [7] introduce the parameter fracture number; having a bounded *variable fracture number* $\mathfrak{p}^V(A)$ implies that deleting a few columns of A breaks it into independent blocks of small size; similarly for *constraint fracture number* $\mathfrak{p}^C(A)$ and deleting a few rows. Because bounded $\mathfrak{p}^V(A)$ implies bounded $\mathrm{td}_P(A)$ and bounded $\mathfrak{p}^C(A)$ implies bounded $\mathrm{td}_D(A)$, our results generalize theirs. The remaining case of *mixed fracture number* $\mathfrak{p}(A)$, where deleting both rows and columns is allowed, reduces to the 4-block n-fold ILP problem, which is not known to be either FPT or W[1]-hard. Because bounded $\mathfrak{p}(A)$ implies bounded $\mathrm{td}_I(A)$, ILP parameterized by $\mathrm{td}_I(A) + \|A\|_\infty$ is at least as hard as 4-block n-fold ILP, highlighting its status as an important open problem.

Organization. The paper contains three main parts. In Section 2, we provide the proof of Theorem 1, showing the existence of a strongly polynomial algorithm whenever a Graver-best oracle is provided. Then, in Section 3, we provide Graver-best oracles for multi-stage stochastic and tree-fold ILPs and discuss n-fold ILP, and prove Theorems 2, 3 and 4. Finally, in Section 4 we show how to embed any instance of bounded primal or dual treedepth into a multi-stage stochastic or tree-fold ILP without increasing the relevant parameters, proving Theorems 5 and 6. Due to space restrictions the proofs of our technical statement and other supplementary material are moved to the full version available at `https://arxiv.org/abs/1802.05859`; the statements whose proofs are presented there are marked with (*) .

2 The Graver-best Oracle Algorithm

2.1 Preliminaries

For positive integers m, n, $m \leq n$, we set $[m, n] = \{m, \ldots, n\}$ and $[n] = [1, n]$. We write vectors in boldface (e.g., $\mathbf{x}, \mathbf{y}$) and their entries in normal font (e.g., the i-th entry of $\mathbf{x}$ is x_i). If A is a matrix, A_r denotes its r-th column. For an integer $a \in \mathbb{Z}$, we denote by $\langle a \rangle = 1 + \log_2 a$ the binary encoding length of a; we extend this notation to vectors, matrices and tuples of these objects. For example, $\langle A, \mathbf{b} \rangle = \langle A \rangle + \langle \mathbf{b} \rangle$, and $\langle A \rangle = \sum_{i,j} \langle a_{ij} \rangle$. For a graph G we denote by $V(G)$ its set of vertices.

Graver bases and augmentation. Let us now introduce Graver bases and discuss how they are used for optimization. We define a partial order $\sqsubseteq$ on $\mathbb{R}^n$ as follows: for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ we write $\mathbf{x} \sqsubseteq \mathbf{y}$ and say that $\mathbf{x}$ is *conformal* to $\mathbf{y}$ if $x_i y_i \geq 0$ (that is, $\mathbf{x}$ and $\mathbf{y}$ lie in the same orthant) and $|x_i| \leq |y_i|$ for $i \in [n]$. It is well known that every subset of $\mathbb{Z}^n$ has finitely many $\sqsubseteq$-minimal elements.

▶ **Definition 7** (Graver basis)**.** The *Graver basis* of an integer $m \times n$ matrix A is the finite set $\mathcal{G}(A) \subset \mathbb{Z}^n$ of $\sqsubseteq$-minimal elements in $\{\mathbf{x} \in \mathbb{Z}^n : A\mathbf{x} = 0,\ \mathbf{x} \neq 0\}$.

We say that $\mathbf{x}$ is *feasible* for (ILP) if $A\mathbf{x} = \mathbf{b}$ and $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$. Let $\mathbf{x}$ be a feasible solution for (ILP). We call $\mathbf{g}$ a *feasible step* if $\mathbf{x} + \mathbf{g}$ is feasible for (ILP). Further, call a feasible step $\mathbf{g}$ *augmenting* if $\mathbf{w}(\mathbf{x} + \mathbf{g}) < \mathbf{w}(\mathbf{x})$. An augmenting step $\mathbf{g}$ and a *step length* $\alpha \in \mathbb{Z}$ form an $\mathbf{x}$-*feasible step pair* with respect to a feasible solution $\mathbf{x}$ if $\mathbf{l} \leq \mathbf{x} + \alpha\mathbf{g} \leq \mathbf{u}$. An augmenting step $\mathbf{h}$ is a *Graver-best step* for $\mathbf{x}$ if $\mathbf{w}(\mathbf{x} + \mathbf{h}) \leq \mathbf{w}(\mathbf{x} + \lambda\mathbf{g})$ for all $\mathbf{x}$-feasible step pairs $(\mathbf{g}, \lambda) \in \mathcal{G}(A) \times \mathbb{Z}$. The *Graver-best augmentation procedure* for (ILP) with a given feasible solution $\mathbf{x}_0$ works as follows:

1. If there is no Graver-best step for $\mathbf{x}_0$, return it as optimal.
2. If a Graver-best step $\mathbf{h}$ for $\mathbf{x}_0$ exists, set $\mathbf{x}_0 := \mathbf{x}_0 + \mathbf{h}$ and go to 1.

▶ **Proposition 8** ([21, Lemma 3.10]). *Given a feasible solution* $\mathbf{x}_0$ *for* (ILP)*, the Graver-best augmentation procedure finds an optimum of* (ILP) *in at most* $(2n-2)\log F$ *steps, where* $F = \mathbf{w}\mathbf{x}_0 - \mathbf{w}\mathbf{x}^*$ *and* $\mathbf{x}^*$ *is any minimizer of* $\mathbf{w}\mathbf{x}$*.*

▶ **Definition 9** (Graver-best oracle). A *Graver-best oracle* for an integer matrix A is one that, queried on $\mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u}$ and $\mathbf{x}$ feasible to (ILP), returns a Graver-best step $\mathbf{h}$ for $\mathbf{x}$.

2.2 The Algorithm

It follows from Proposition 8 that given a Graver-best oracle, problem (ILP) can be solved in time which is polynomial in the binary encoding length $\langle A, \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u}\rangle$ of the input. We now show that, in fact, given such an oracle, the problem admits a *strongly* polynomial algorithm. In the next theorem the input has only arithmetic and numeric parts and no parametric part.

▶ **Theorem 1.** *The problem* (ILP) *with arithmetic input* $\langle A\rangle$ *and numeric input* $\langle \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u}\rangle$*, endowed with a Graver-best oracle for* A*, is solvable by a strongly polynomial oracle algorithm.*

▶ Remark. The partition of the input to the arithmetic input $\langle A\rangle$ and the numeric input $\langle \mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u}\rangle$ is the same as in the classical results for linear programming [8, 24].

Proof. The algorithm which demonstrates the theorem consists of several steps as follows.

Step 1: Reducing $\mathbf{b}, \mathbf{l}, \mathbf{u}$. Apply the strongly polynomial algorithm of Tardos [24] to the linear programming relaxation $\min\{\mathbf{w}\mathbf{y} \mid \mathbf{y} \in \mathbb{R}^n,\ A\mathbf{y} = \mathbf{b},\ \mathbf{l} \le \mathbf{y} \le \mathbf{u}\}$; the algorithm performs $\mathcal{L}(\langle A\rangle) = \mathrm{poly}(\langle A\rangle)$ arithmetic operations. If the relaxation is infeasible then so is (ILP) and we are done. If it is unbounded then (ILP) is either infeasible or unbounded too, and in this case we set $\mathbf{w} := \mathbf{0}$ so that all solutions are optimal, and we proceed as below and terminate at the end of step 3. Suppose then that we obtain an optimal solution $\mathbf{y}^* \in \mathbb{R}^n$ to the relaxation, with round down $\lfloor \mathbf{y}^* \rfloor \in \mathbb{Z}^n$. Let $a := \max\{2, \|A\|_\infty\}$. Let $\mathcal{C}(A) \subseteq \mathcal{G}(A)$ be the set of *circuits* of A, which are those $\mathbf{c} \in \mathcal{G}(A)$ with support which is a circuit of the linear matroid of A. Let $c_\infty := \max_{\mathbf{c}\in\mathcal{C}(A)} \|\mathbf{c}\|_\infty$. We have $c_\infty \le n^{\frac{n}{2}} a^n$ [21, Lemma 3.18].

We now use the proximity results of [12, 14] which assert that either (ILP) is infeasible or it has an optimal solution $\mathbf{x}^*$ with $\|\mathbf{x}^* - \mathbf{y}^*\|_\infty \le n c_\infty$ and hence $\|\mathbf{x}^* - \lfloor \mathbf{y}^* \rfloor\|_\infty \le n^{\frac{n}{2}+1} a^n + 1$. Thus, making the variable transformation $\mathbf{x} = \mathbf{z} + \lfloor \mathbf{y}^* \rfloor$, problem (ILP) reduces to following,

$$\min\left\{\mathbf{w}(\mathbf{z} + \lfloor \mathbf{y}^* \rfloor) \mid \mathbf{z} \in \mathbb{Z}^n\,,\ A(\mathbf{z} + \lfloor \mathbf{y}^* \rfloor) = \mathbf{b}\,,\ \mathbf{l} \le \mathbf{z} + \lfloor \mathbf{y}^* \rfloor \le \mathbf{u}\,,\ \|\mathbf{z}\|_\infty \le n^{\frac{n}{2}+1} a^n + 1\right\},$$

which is equivalent to the program

$$\min\left\{\mathbf{w}\mathbf{z} \mid \mathbf{z} \in \mathbb{Z}^n\,,\ A\mathbf{z} = \bar{\mathbf{b}}\,,\ \bar{\mathbf{l}} \le \mathbf{z} \le \bar{\mathbf{u}}\right\} \tag{1}$$

where

$$\bar{\mathbf{b}} := \mathbf{b} - A\lfloor \mathbf{y}^* \rfloor,\ \ \bar{l}_i := \max\{l_i - \lfloor y_i^* \rfloor, -(n^{\frac{n}{2}+1} a^n + 1)\},\ \ \bar{u}_i := \min\{u_i - \lfloor y_i^* \rfloor, n^{\frac{n}{2}+1} a^n + 1\}\quad .$$

If some $\bar{l}_i > \bar{u}_i$ then (1) is infeasible and hence so is (ILP), so we may assume that

$$-(n^{\frac{n}{2}+1} a^n + 1) \le \bar{l}_i \le \bar{u}_i \le n^{\frac{n}{2}+1} a^n + 1, \quad \text{for all } i\ .$$

This implies that if $\mathbf{z}$ is any feasible point in (1) then $\|A\mathbf{z}\|_\infty \le na(n^{\frac{n}{2}+1} a^n + 1)$ and so we may assume that $\|\bar{\mathbf{b}}\|_\infty \le na(n^{\frac{n}{2}+1} a^n + 1)$ else there is no feasible solution. So we have

$$\|\bar{\mathbf{b}}\|_\infty, \|\bar{\mathbf{l}}\|_\infty, \|\bar{\mathbf{u}}\|_\infty \le 2^{O(n\log n)} a^{O(n)} \text{ and hence } \langle \bar{\mathbf{b}}, \bar{\mathbf{l}}, \bar{\mathbf{u}}\rangle \text{ is polynomial in } \langle A\rangle\quad .$$

Step 2: Solving the system of equations. We first search for an integer solution to the system of equations $A\mathbf{z} = \bar{\mathbf{b}}$. This can be done by computing the Hermite normal form of A, see [23], using a number of arithmetic operations polynomial in $\langle A\rangle$ and time polynomial in $\langle A, \bar{\mathbf{b}}\rangle$ which is polynomial in $\langle A\rangle$, and hence strongly polynomially in our original input. Then either we conclude that there is no integer solution to $A\mathbf{z} = \bar{\mathbf{b}}$ and hence (1) is infeasible, or we find a solution $\mathbf{z} \in \mathbb{Z}^n$ with $\langle \mathbf{z}\rangle$ polynomially bounded in $\langle A, \bar{\mathbf{b}}\rangle$ and hence also in $\langle A\rangle$.

Step 3: Finding a feasible point. Define relaxed bounds by

$$\hat{l}_i := \min\{\bar{l}_i, z_i\}, \quad \hat{u}_i := \max\{\bar{u}_i, z_i\}, \quad i \in [n] \ .$$

Now for $i \in [n]$ iterate the following. If $\bar{l}_i \le z_i \le \bar{u}_i$ then simply increment i and repeat. If $z_i < \bar{l}_i$ (and hence $\hat{l}_i = z_i$ and $\hat{u}_i = \bar{u}_i$) then consider the following auxiliary integer program,

$$\max\left\{x_i \mid \mathbf{x} \in \mathbb{Z}^n \,,\ A\mathbf{x} = \bar{\mathbf{b}} \,,\ \hat{\mathbf{l}} \le \mathbf{x} \le \hat{\mathbf{u}}\right\}. \tag{2}$$

Starting from the point $\mathbf{z}$ feasible in (2), and using the Graver-best oracle for A, we can solve program (2) using Proposition 8 in polynomial time and in a number of arithmetic operations and oracle queries which is polynomial in n and $\log F$ (recall $F = z_i^* - z_i$ for some minimizer z_i^*), which is bounded by $\log(\hat{u}_i - \hat{l}_i) = \log(\bar{u}_i - z_i)$, thus polynomial in $\langle A\rangle$.

Let $\mathbf{x}$ be an optimal solution of (2). If $x_i < \bar{l}_i$ then (1) is infeasible and we are done. Otherwise (in which case $\bar{l}_i \le x_i \le \bar{u}_i$) we update $\hat{l}_i := \bar{l}_i$ and $\mathbf{z} := \mathbf{x}$, increment i and repeat. The last case $z_i > \bar{u}_i$ is treated similarly where in (2) we minimize rather than maximize x_i.

Thus, strongly polynomially we either conclude at some iteration i that program (1) is infeasible or complete all iterations and obtain $\hat{\mathbf{l}} = \bar{\mathbf{l}}$, $\hat{\mathbf{u}} = \bar{\mathbf{u}}$, and a point $\mathbf{z}$ feasible in (1).

Step 4: Reducing $\mathbf{w}$. Let $N := 2n(n^{\frac{n}{2}+1}a^n + 1) + 1$. Now apply the strongly polynomial algorithm of Frank and Tardos [8], which on arithmetic input $n, \langle N\rangle$ and numeric input $\langle \mathbf{w}\rangle$, outputs $\bar{\mathbf{w}} \in \mathbb{Z}^n$ with $\|\bar{\mathbf{w}}\|_\infty \le 2^{O(n^3)}N^{O(n^2)}$ such that $\operatorname{sign}(\mathbf{wx}) = \operatorname{sign}(\bar{\mathbf{w}}\mathbf{x})$ for all $\mathbf{x} \in \mathbb{Z}^n$ with $\|\mathbf{x}\|_1 < N$. Since $\langle N\rangle = O(\log N) = O(n\log n + n\log a)$ is polynomial in $\langle A\rangle$, this algorithm is also strongly polynomial in our original input. Now, for every two points $\mathbf{x}, \mathbf{z}$ feasible in (1) we have $\|\mathbf{x} - \mathbf{z}\|_1 < 2n(n^{\frac{n}{2}+1}a^n + 1) + 1 = N$, so that for any two such points we have $\mathbf{wx} \le \mathbf{wz}$ if and only if $\bar{\mathbf{w}}\mathbf{x} \le \bar{\mathbf{w}}\mathbf{z}$, and therefore we can replace (1) by the equivalent program

$$\min\left\{\bar{\mathbf{w}}\mathbf{z} \ :\ \mathbf{z} \in \mathbb{Z}^n \,,\ A\mathbf{z} = \bar{\mathbf{b}} \,,\ \bar{\mathbf{l}} \le \mathbf{z} \le \bar{\mathbf{u}}\right\}, \tag{3}$$

where

$$\|\bar{\mathbf{w}}\|_\infty = 2^{O(n^3\log n)}a^{O(n^3)} \text{ and hence } \langle \bar{\mathbf{w}}, \bar{\mathbf{b}}, \bar{\mathbf{l}}, \bar{\mathbf{u}}\rangle \text{ is polynomial in } \langle A\rangle \quad .$$

Step 5: Finding an optimal solution. Starting from the point $\mathbf{z}$ which is feasible in (3), and using the Graver-best oracle for A, we can solve program (3) using again Proposition 8 in polynomial time and in a number of arithmetic operations and oracle queries which is polynomial in n and in $\log F$, which is bounded by $\log\left(n\|\bar{\mathbf{w}}\|_\infty\|\bar{\mathbf{u}} - \bar{\mathbf{l}}\|_\infty\right)$, which is polynomial in $\langle A\rangle$, and hence strongly polynomially. ◀

▶ Remark. In fact, the reduced objective $\bar{\mathbf{w}}$ in step 4 need not be constructed: already its *existence* implies that (1) is solved in the same number of iterations as (3).

3 Multi-stage Stochastic and Tree-fold ILP

In this section we prove Theorems 2 and 3. We first formalize a common construction for a Graver-best oracle: one constructs a set of relevant step lenghts Λ and then for each $\lambda \in \Lambda$ finds a λ-Graver-best step. A step with the best improvement among these is then guaranteed to be a Graver-best step. Thus, we reduce our task to constructing a Λ-Graver-best oracle.

Both algorithms for multi-stage stochastic ILP and tree-fold ILP follow the same pattern:

1. show that all elements of $\mathcal{G}(A)$ have bounded norms (ℓ_∞ and ℓ_1, respectively),
2. show that A has bounded treewidth (primal and dual, respectively),
3. apply existing algorithms for (ILP) which are FPT parameterized by $\|A\|_\infty$, $\max \|\mathbf{x}\|_\infty$ and $\max \|\mathbf{x}\|_1$, and $\mathrm{tw}_P(A)$ and $\mathrm{tw}_D(A)$, respectively.

3.1 Preliminaries

3.1.1 Relevant Step Lengths

We say that $\mathbf{h} \in \{\mathbf{x} \in \mathbb{Z}^n \mid A\mathbf{x} = \mathbf{0}\}$ is a *λ-Graver-best step* if $\lambda\mathbf{h}$ is a feasible step and $\lambda\mathbf{w}\mathbf{h} \leq \lambda\mathbf{w}\mathbf{g}$ for any $\mathbf{g} \in \mathcal{G}(A)$ such that $\lambda\mathbf{g}$ is a feasible step. We denote by $g_1(A) = \max_{\mathbf{g}\in\mathcal{G}(A)} \|\mathbf{g}\|_1$ and $g_\infty(A) = \max_{\mathbf{g}\in\mathcal{G}(A)} \|\mathbf{g}\|_\infty$. The following lemma states that provided a bound on $g_\infty(A)$, in order to find a Graver-best step, it is sufficient to find a λ-Graver-best step for all $\lambda \in \Lambda$ for some not too large set Λ.

▶ **Definition 10** (Graver-best step-lengths)**.** Let $\mathbf{x}$ be a feasible solution to (ILP). We say that $\lambda \in \mathbb{N}$ is a *Graver-best step-length for* $\mathbf{x}$ if there exists $\mathbf{g} \in \mathcal{G}(A)$ with $\mathbf{x} + \lambda\mathbf{g}$ feasible, such that $\forall \lambda' \in \mathbb{N}$ and $\forall \mathbf{g}' \in \mathcal{G}(A)$, $\mathbf{x} + \lambda'\mathbf{g}'$ is either infeasible or $\mathbf{w}(\mathbf{x} + \lambda\mathbf{g}) \leq \mathbf{w}(\mathbf{x} + \lambda'\mathbf{g}')$. We denote by $\Lambda(\mathbf{x}) \subseteq \mathbb{N}$ the set of Graver-best step-lengths for $\mathbf{x}$.

▶ **Lemma 11** (Polynomial $\Lambda \supseteq \Lambda(\mathbf{x})$)**. (*)** *Let $\mathbf{x}$ be a feasible solution to* (ILP)*, let $M \in \mathbb{N}$ be such that $g_\infty(A) \leq M$. Then it is possible to construct in time $O(Mn)$ a set $\Lambda \subseteq \mathbb{N}$ of size at most $2Mn$ such that $\Lambda(\mathbf{x}) \subseteq \Lambda$.*

With this Λ at hand, in order to realize a Graver-best oracle, it suffices to realize an oracle which finds a λ-Graver-best step for a given λ:

▶ **Definition 12** (Λ-Graver-best oracle)**.** A *Λ-Graver-best oracle* for an integer matrix A is one that, queried on $\mathbf{w}, \mathbf{b}, \mathbf{l}, \mathbf{u}$, $\mathbf{x}$ feasible to (ILP), and an integer $\lambda \in \mathbb{N}$, returns a λ-Graver-best step $\mathbf{h}$ for $\mathbf{x}$.

▶ **Lemma 13** (Λ-Graver-best oracle $\Rightarrow$ Graver-best oracle)**. (*)** *Let A be an integer matrix and let $M \in \mathbb{N}$ satisfy $g_\infty(A) \leq M$. Then a Graver-best oracle for A can be realized with $2Mn$ calls to a Λ-Graver-best oracle for A.*

3.1.2 Multi-stage Stochastic and Tree-fold Matrices

Let the *height* of a rooted tree or forest be the maximum root-to-leaf distance in it (i.e., the number of edges along the root-to-leaf path). In the following we let T be a rooted tree of height $\tau - 1 \in \mathbb{N}$ whose all leaves are at *depth* $\tau - 1$, that is, the length of every root-leaf path is exactly $\tau - 1$. For a vertex $v \in T$, let T_v be the subtree of T rooted in v and let $\ell(v)$ denote the number of leaves of T contained in T_v. Let $B_1, B_2, \ldots, B_\tau$ be a sequence of integer matrices with each B_s having $l \in \mathbb{N}$ rows and n_s columns, where $n_s \in \mathbb{N}$, $n_s \geq 1$. We shall define a *multi-stage stochastic* matrix $T^P(B_1, \ldots, B_\tau)$ inductively; the superscript P

refers to the fact that $T^P(B_1,\dots,B_\tau)$ has bounded *primal* treedepth td_P, as we will later see.

For a leaf $v \in T$, $T^P_v(B_\tau) := B_\tau$. Let $d \in \mathbb{N}$, $0 \le d \le \tau - 2$, and assume that for all vertices $v \in T$ at depth $d+1$, matrices $T^P_v(B_{\tau-d},\dots,B_\tau)$ have been defined. For $s \in \mathbb{N}$, $1 \le s \le \tau$, we set $T^P_v(B_{[s:\tau]}) = T^P_v(B_s,\dots,B_\tau)$. Let $v \in T$ be a vertex at depth d with δ children $v_1,\dots,v_\delta$. We set

$$T^P_v(B_{[\tau-d-1:\tau]}) := \begin{pmatrix} B_{\tau-d-1,\ell(v_1)} & T^P_{v_1}(B_{[\tau-d:\tau]}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ B_{\tau-d-1,\ell(v_\delta)} & 0 & \cdots & T^P_{v_\delta}(B_{[\tau-d:\tau]}) \end{pmatrix}$$

where, for $N \in \mathbb{N}$, $B_{s,N} = \begin{pmatrix} B_s \\ \vdots \\ B_s \end{pmatrix}$ consists of N copies of the matrix B_s.

The structure of a multi-stage stochastic matrix makes it natural to partition any solution of a multi-stage stochastic ILP into *bricks*. Bricks are defined inductively: for $T^P_v(B_\tau)$ there is only one brick consisting of all coordinates; for $T^P_v(B_{[s:\tau]})$ the set of bricks is composed of all bricks for all descendants of v, plus the first n_s coordinates form an additional brick.

▶ **Example 14.** For $\tau = 3$ and T with root r of degree 2 and its children u and v of degree 2 and 3, we have $T^P_u(B_2,B_3) = \left(\begin{smallmatrix} B_2 & B_3 \\ B_2 & B_3 \end{smallmatrix}\right)$, $T^P_v(B_2,B_3) = \left(\begin{smallmatrix} B_2 & B_3 & \\ B_2 & & B_3 \\ B_2 & & B_3 \end{smallmatrix}\right)$, and $T^P(B_1,B_2,B_2) = T^P_r(B_1,B_2,B_2) = \left(\begin{smallmatrix} B_1 & B_2 & B_3 & & \\ B_1 & B_2 & & B_3 & \\ B_1 & & B_2 & B_3 & \\ B_1 & & B_2 & & B_3 \end{smallmatrix}\right)$, with a total of 8 bricks.

Tree-fold matrices are essentially transposes of multi-stage stochastic ILP matrices. Let T be as before and $A_1,\dots,A_\tau$ be a sequence of integer matrices with each $A_s \in \mathbb{Z}^{r_s \times t}$, where $t \in \mathbb{N}$, $r_s \in \mathbb{N}$, $r_s \ge 1$. We shall define $T^D(A_1,\dots,A_\tau)$ inductively; the superscript D refers to the fact that $T^D(A_1,\dots,A_\tau)$ has bounded *dual* treedepth. The inductive definition is the same as before except that, for a vertex $v \in T$ at depth d with δ children $v_1,\dots,v_\delta$, we set

$$T^D_v(A_{[\tau-d-1:\tau]}) := \begin{pmatrix} A_{\tau-d-1,\ell(v_1)} & A_{\tau-d-1,\ell(v_2)} & \cdots & A_{\tau-d-1,\ell(v_\delta)} \\ T^D_{v_1}(A_{[\tau-d:\tau]}) & 0 & \cdots & 0 \\ 0 & T^D_{v_2}(A_{[\tau-d:\tau]}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & T^D_{v_\delta}(A_{[\tau-d:\tau]}) \end{pmatrix}$$

where, for $N \in \mathbb{N}$, $A_{s,N} = (A_s \cdots A_s)$ consists of N copies of the matrix A_s. A solution $\mathbf{x}$ of a tree-fold ILP is partitioned into bricks $(\mathbf{x}^1,\dots,\mathbf{x}^n)$ where n is the number of leaves of T, and each $\mathbf{x}^i$ is a t-dimensional vector.

3.1.3 Structural Parameters

We consider two graph parameters, namely *treewidth* $\mathrm{tw}(G)$ and *treedepth* $\mathrm{td}(G)$. We postpone the definition of treewidth to the full version as it is not central for us.

▶ **Definition 15** (Treedepth)**.** The *closure* $\mathrm{cl}(F)$ of a rooted forest F is the graph obtained from F by making every vertex adjacent to all of its ancestors. The *treedepth* $\mathrm{td}(G)$ of a graph G is one more than the minimum height of a forest F such that $G \subseteq \mathrm{cl}(F)$.

It is known that $\mathrm{tw}(G) \le \mathrm{td}(G)$. The treedepth $\mathrm{td}(G)$ of a graph G with a witness forest F can be computed in time $f_{\mathrm{td}}(\mathrm{td}(G)) \cdot |V(G)|$ for some computable function f_{td} [22].

▶ **Definition 16** (Primal and dual graph)**.** Given a matrix $A \in \mathbb{Z}^{m\times n}$, its *primal graph* $G_P(A) = (V, E)$ is defined as $V = [n]$ and $E = \{\{i, j\} \in \binom{[n]}{2} \mid \exists k \in [m] : A_{k,i}, A_{k,j} \neq 0\}$. In other words, its vertices are the columns of A and two vertices are connected if there is a row with non-zero entries at the corresponding columns. The *dual graph of* A is defined as $G_D(A) = G_P(A^\intercal)$, that is, the primal graph of the transpose of A.

▶ **Definition 17** (Matrix treewidth)**.** Given a matrix A, its *primal treewidth* $\mathrm{tw}_P(A)$ is defined as the treewidth of its primal graph, i.e., $\mathrm{tw}(G_P(A))$, and its *dual treewidth* $\mathrm{tw}_D(A)$ is $\mathrm{tw}(G_D(A))$. Similarly, we define the *primal* and *dual treedepth* as $\mathrm{td}_P(A) = \mathrm{td}(G_P(A))$ and $\mathrm{td}_D(A) = \mathrm{td}(G_D(A))$, respectively.

Using a proof of Ganian et al. [11, Theorem 12] we show that we cannot hope to relax the parameter $\mathrm{td}_D(A)$ to $\mathrm{tw}_D(A)$, even if $\|A\|_\infty$ was a constant.

▶ **Lemma 18. (*)** (ILP) *is* NP*-hard already when* $\mathrm{tw}_D(A) = 3$*,* $\|A\|_\infty = 2$*, and* $\mathbf{w} = \mathbf{0}$*.*

3.2 Multi-stage Stochastic ILP is strongly FPT

To prove Theorem 2, we need two ingredients: a bound on $g_\infty(A)$, and an algorithm for (ILP) with bounded $\mathrm{tw}_P(A)$ and $\max \|\mathbf{x}\|_\infty$.

▶ **Lemma 19** (Multi-stage stochastic ⇒ bounded $g_\infty(A)$)**. (*)** *Let* $A = T^P(B_1, \ldots, B_\tau)$*. Then* $g_\infty(A) \leq f_{mss\text{-}norm}(a, n_1, \ldots, n_\tau, l)$ *for some computable function* $f_{mss\text{-}norm}$*.*

▶ **Lemma 20. (*)** *Let* $X \in \mathbb{N}$*. Problem* (ILP) *with the additional constraint* $\|\mathbf{x}\|_\infty \leq X$ *can be solved in time* $(X + 1)^{O(\mathrm{tw}_P(A))} \cdot (n + m)$*.*

Proof of Theorem 2. Let $A = T^P(B_1, \ldots, B_\tau)$ be a multi-stage stochastic matrix. By Lemma 19, $g_\infty(A)$ is bounded by $M = f_{\text{mss-norm}}(a, n_1, \ldots, n_\tau, l)$. We show how to construct a Λ-Graver-best oracle. Given an integer $\lambda \in \mathbb{N}$, use Lemma 20 to solve

$$\min\{\lambda\mathbf{wh} \mid A\mathbf{h} = 0,\ \mathbf{l} \leq \mathbf{x} + \lambda\mathbf{h} \leq \mathbf{u},\ \|\mathbf{h}\|_\infty \leq M\}\ .$$

This returns a λ-Graver-best step, because any optimal solution satisfies $\lambda\mathbf{h} \leq \lambda\mathbf{g}$ for all $\mathbf{g} \in \mathcal{G}(A)$. Using a simple induction and the inductive construction of A, one gets that A has $\mathrm{tw}_P(A) \leq \mathrm{td}_P(A) \leq n_1 + \cdots + n_\tau + 1$ and thus the oracle is realized in FPT time. Lemma 13 then yields a Graver-best oracle, which, combined with Theorem 1, finishes the proof. ◀

3.3 Tree-fold ILP is strongly FPT

As before, to prove Theorem 3, we need two ingredients: a bound on $g_1(A)$, and an algorithm for (ILP) with bounded $\mathrm{tw}_D(A)$ and $\max \|\mathbf{x}\|_1$.

▶ **Lemma 21** (Tree-fold ⇒ bounded $g_1(A)$)**. (*)** *Let* $A_i \in \mathbb{Z}^{r_i\times t}$ *for* $i \in [\tau]$ *with* $a = \max\{2, \max_{i\in[\tau]} \|A_i\|_\infty\}$*,* $r = \sum_{i=1}^{\tau} r_i$*. Let* $A = T^D(A_1, \ldots, A_\tau)$*. There exists a computable function* $f_{tf\text{-}norm}(a, r_1, \ldots, r_\tau)$ *such that* $g_1(A) \leq f_{tf\text{-}norm}(a, r_1, \ldots, r_\tau)$*.*

Proof sketch. Chen and Marx [4] prove a similar result under the assumption that t is also a parameter; thus, the remaining problem are essentially duplicitous columns. However, De Loera et al. [5] show that repeating columns of any matrix A' does not increase $g_1(A')$, and thus we can take A, delete duplicitous columns, apply the result of Chen and Marx, and our Lemma follows.

▶ **Lemma 22. (*)** *Let* $X \in \mathbb{N}$*. Problem* (ILP) *with the additional constraint* $\|\mathbf{x}\|_1 \leq X$ *can be solved in time* $(aX)^{O(\mathrm{tw}_D(A))} \cdot n$*, where* $a = \max\{2, \|A\|_\infty\}$*.*

Proof sketch. Lemma 22 is proved by reformulating the nonlinear constraint $\|\mathbf{x}\|_1 \leq X$ by "splitting" each variable x_i into two non-negative variables $x_i = x_i^+ - x_i^-$, imposing the constraint $\sum_{i=1}^n (x_i^+ + x_i^-) \leq X$, and showing that this does not increase $\mathrm{tw}_D(A)$ much; then, a recent dynamic programming algorithm of Ganian et al. [11, Theorem 6] does the job.

Proof of Theorem 3. Let $A = T^D(A_1, \ldots, A_\tau)$ be a tree-fold matrix. By Lemma 21 we have that $g_1(A) \leq f_{\text{tf-norm}}(a, r_1, \ldots, r_\tau) =: M$ We show how to construct a Λ-Graver-best oracle. Given an integer $\lambda \in \mathbb{N}$, solve $\min\{\lambda \mathbf{wh} \mid A\mathbf{h} = 0, \mathbf{l} \leq \mathbf{x} + \lambda\mathbf{h} \leq \mathbf{u}, \|\mathbf{h}\|_1 \leq M\}$ using Lemma 22; clearly the result is a λ-Graver-best step. Using a simple induction and the inductive construction of A, one gets that A has $\mathrm{tw}_D(A) \leq \mathrm{td}_D(A) \leq r_1 + \cdots + r_\tau + 1$ and thus the oracle is realized in FPT time. Lemma 13 then yields a Graver-best oracle, which, combined with Theorem 1, finishes the proof. ◀

***n*-fold ILP.** A special case of tree-fold ILP is n-fold ILP, obtained by taking T to be the star with n leaves and $A = T^D(A_1, A_2)$, where $A_1 \in \mathbb{Z}^{r \times t}$ and $A_2 \in \mathbb{Z}^{s \times t}$.

Proof of Theorem 4. Before we apply Lemma 22, we need to bound $g_1(A)$. It follows from the proof of [13, Lemma 6.1] that there is a number $g(A) = \max_{\mathbf{v} \in \mathcal{G}(A_1\mathcal{G}(A_2))} \|\mathbf{v}\|_1$ such that $g_1(A) \leq g(A) \cdot g_1(A_2)$.

Let $d_2 \leq (2a+1)^s$ be the number of distinct columns of A_2. De Loera et al. [5] give a bound on $g_1(A)$ in terms of the number of distinct columns of a matrix A:

▶ **Lemma 23** ([5, Corollary 3.7.4]). *Let $A \in \mathbb{Z}^{m \times n}$ be a matrix of rank r, let d be the number of different columns in A, and let $a = \max\{2, \|A\|_\infty\}$. Then $g_1(A) \leq (d-r)(r+1)(\sqrt{ma})^m$.*

Thus, $g_1(A_2) \leq (d_2 - s)(s+1)(\sqrt{s}a)^s \leq (as)^{O(s)}$. Let G_2 be a matrix whose columns are elements of $\mathcal{G}(A_2)$. We have that $\|A_1G_2\|_\infty \leq a \cdot (as)^{O(s)} \leq (as)^{O(s)}$. Moreover, since A_1G_2 has r rows, it has at most $d_1 = \left((as)^{O(s)}\right)^r = (as)^{O(rs)}$ distinct columns. Again, by Lemma 23 we have that $g(A) = g_1(A_1G_2) \leq (d_1 - r)(r+1)(\sqrt{r}as)^{O(s)})^r \leq (ars)^{O(rs)}$. Combining, we get $g_1(A) \leq (ars)^{O(rs)} \cdot (as)^{O(s)} \leq (ars)^{O(rs)} =: M$.

We have $\mathrm{tw}_D(A) \leq r + s + 1$ and thus running the algorithm of Lemma 22 once takes time $\left((ars)^{O(rs)}\right)^{r+s} nt \leq (ars)^{O(r^2s+rs^2)}nt$ and finds the λ-Graver-best step. Lemma 13 then yields a Graver-best oracle. The reduced objective function $\bar{\mathbf{w}}$ in Step 4 of the proof of Theorem 1 satisfies $\|\bar{\mathbf{w}}\|_\infty \leq (ant)^{O((nt)^3)}$ and thus the number of calls to the Graver-best oracle is bounded by $(nt)^3 \log(nt)$, concluding the proof. ◀

4 Primal and Dual Treedepth

We prove Theorems 5 and 6 by showing that an ILP with bounded primal (dual) treedepth can be embedded into a multi-stage stochastic (tree-fold) ILP without increasing the parameters too much. The precise notion of how one ILP is embedded in another is captured as follows.

▶ **Definition 24** (Extended formulation). Let $n' \geq n$, $m' \in \mathbb{N}$, $A \in \mathbb{Z}^{m \times n}$, $\mathbf{b} \in \mathbb{Z}^m$, $\mathbf{l}, \mathbf{u} \in (\mathbb{Z} \cup \{\pm\infty\}^n$ and $A' \in \mathbb{Z}^{m' \times n'}$, $\mathbf{b}' \in \mathbb{Z}^{m'}$, $\mathbf{l}', \mathbf{u}' \in (\mathbb{Z} \cup \{\pm\infty\})^{n'}$. We say that $A'(\mathbf{x}, \mathbf{y}) = \mathbf{b}', \mathbf{l}' \leq (\mathbf{x}, \mathbf{y}) \leq \mathbf{u}'$ is an *extended formulation* of $A\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$ if $\{\mathbf{x} \mid A\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\} = \{\mathbf{x} \mid \exists \mathbf{y} : A'(\mathbf{x}, \mathbf{y}) = \mathbf{b}', \mathbf{l}' \leq (\mathbf{x}, \mathbf{y}) \leq \mathbf{u}'\}$.

We note that from here on we always assume that if $\mathrm{td}_P(A) = k$ or $\mathrm{td}_D(A) = k$, then there is a *tree* (not a forest) F of height $k-1$ such that $G_P(A) \subseteq \mathrm{cl}(F)$ or $G_D(A) \subseteq \mathrm{cl}(F)$, respectively. Otherwise $G_P(A)$ is not connected and each component corresponds to a subset of variables which defines an ILP that can be solved independently; similarly for $G_D(A)$.

4.1 Primal Treedepth

▶ **Lemma 25** (Bounded primal treedepth ⇒ multi-stage stochastic). *Let $A, \mathbf{b}, \mathbf{l}$ and $\mathbf{u}$ as in* (ILP) *be given, let $a = \max\{2, \|A\|_\infty\}$ and $\tau + 1 = \mathrm{td}_P(A)$. Then there exists $C \in \mathbb{Z}^{m' \times n'}$, $\mathbf{b}' \in \mathbb{Z}^{m'}$ and $\mathbf{l}', \mathbf{u}' \in (\mathbb{Z} \cup \{\pm\infty\})^{n'}$, $n' \leq n\tau$, $m' \leq (2a+1)^{\tau^2}$, which define an integer program $C(\mathbf{x}, \mathbf{y}) = \mathbf{b}', \mathbf{l}' \leq (\mathbf{x}, \mathbf{y}) \leq \mathbf{u}'$, which is an extended formulation of $A\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$. Moreover, there exist matrices $B_1, \ldots, B_{\tau-1} \in \mathbb{Z}^{(2a+1)^{\tau^2} \times \tau}$ and $B_\tau \in \mathbb{Z}^{(2a+1)^{\tau^2} \times \left(\tau + (2a+1)^{\tau^2}\right)}$ and a tree T such that $C = T^P(B_1, \ldots, B_\tau)$ is a multi-stage stochastic constraint matrix, and all can be computed in time $f_{P\text{-}embed}(a, \mathrm{td}_P(A)) \cdot n^2$ for some computable function $f_{P\text{-}embed}$.*

Proof. Let F be a rooted tree of height τ such that $G_P(A) \subseteq \mathrm{cl}(F)$ (recall that it can be computed in time $f_{\mathrm{td}}(\mathrm{td}_P(A)) \cdot |V(G_P(A))|$).

Step 1: Dummy columns. We make F structured by adding dummy columns. Observe that every root-leaf path is of length at most τ and thus contains at most τ branching vertices. Unless F is a path, obtain a matrix A' from A by inserting zero columns into A in order to make the path between any two branching vertices of length τ; a special case is the root which we force to be in distance $\tau - 1$ from the closest branching vertex. Set lower and upper bounds on the corresponding new variables to 0. A zero column is an isolated vertex in the primal graph and thus can be inserted to an arbitrary path of the tree F. Moreover, if any leaf is at depth less than $\tau^2 - 1$, insert zero columns in the same way to make it be at depth exacty $\tau^2 - 1$. Now there exists a rooted tree F' of height $\tau^2 - 1$ such that $G_P(A') \subseteq \mathrm{cl}(F')$, all branching vertices are in distances $0, \tau - 1, 2\tau - 1, \ldots, (\tau - 1)\tau - 1$ from the root, and all leaves are at depth exactly $\tau^2 - 1$. We call all vertices at depth 0 or $i\tau - 1$, for $i \in [\tau]$, *frets*, including the root and leaves.

Step 2: Multi-stage stochastic extended formulation. Consider a root-leaf path P in F': its vertex set $V(P)$ corresponds to a certain subset of the columns of A', with $|V(P)| \leq \tau^2$. Furthermore, any row $\mathbf{a}$ of A' with $\mathrm{supp}(\mathbf{a}) \subseteq V(P)$ can be written as a vector $(\mathbf{a}^1, \ldots, \mathbf{a}^\tau) \in \mathbb{Z}^{\tau^2}$ with $\mathbf{a}^i \in \mathbb{Z}^\tau$ for each $i \in [\tau]$, and with $\|\mathbf{a}\|_\infty \leq a$. The *bricks* $\mathbf{a}^i$ correspond to segments between frets (including the end fret, i.e., the fret farthest from the root; segments adjacent to the root also contain the root). Also, for any row $\mathbf{a}$ of A', there exists some root-leaf path P such that $\mathrm{supp}(\mathbf{a}) \subseteq V(P)$.

This inspires the following construction: let $B \in \mathbb{Z}^{(2a+1)^{\tau^2} \times \tau^2}$ be the matrix whose columns are all the possible vectors $\mathbf{a} \in \mathbb{Z}^{\tau^2}$ with $\|\mathbf{a}\|_\infty \leq \|A\|_\infty$. Let $B_i \in \mathbb{Z}^{(2a+1)^{\tau^2} \times \tau}$, for $i \in [\tau]$, be the submatrix of B formed by rows $(i-1)\tau + 1, \ldots, i\tau$ and modify the last such submatrix B_τ by putting $B_\tau := (B_\tau \mid I)$ where $I \in \mathbb{Z}^{(2a+1)^{\tau^2} \times (2a+1)^{\tau^2}}$ is the identity matrix; the variables corresponding to columns of I will play the role of slack variables. Let T be the tree of height τ obtained from F' by contracting all paths between frets.

Now, let $C = T^P(B_1, \ldots, B_\tau)$. Obtain $\tilde{F}$ from F' by appending a leaf to every leaf, and observe that $G_P(T^P(B_1, \ldots, B_\tau)) \subseteq \mathrm{cl}(\tilde{F})$; the new leaves correspond to the slack variables in B_τ. Our goal now is to construct a right hand side vector $\mathbf{b}'$ and lower and upper bounds $\mathbf{l}', \mathbf{u}'$ to enforce exactly the constraints present in $A\mathbf{x} = \mathbf{b}$. For every root-leaf path P in F' there is a corresponding root-leaf path $\tilde{P}$ in $\tilde{F}$ such that $\tilde{P}$ is P with an additional leaf. Fix a root-leaf path P in F'. For every row $\mathbf{a}$ of A' with $\mathrm{supp}(\mathbf{a}) \subseteq V(P)$ and right hand side β, there exists a unique row $\mathbf{c}$ of C with $\mathrm{supp}(\mathbf{c}) \subseteq V(\tilde{P})$ such that $\mathbf{c} = (\mathbf{a}, 1)$, and we set the right hand side of row $\mathbf{c}$ to β.

For every row $\mathbf{c}$ of C which was not considered in the previous paragraph, set the right hand side to 0 and for the slack variable of this row set the lower bound to $-\infty$ and the

upper bound to ∞. Let us remark in passing that we are not limited by using the standard equality form of ILP: transforming an instance $A\mathbf{x} \leq \mathbf{b}$ into the standard equality form by adding slack variables only possibly increases the treedepth by 1. ◀

4.2 Dual Treedepth

▶ **Lemma 26** (Bounded dual treedepth ⇒ tree-fold)**. (*)** *Let $A, \mathbf{b}, \mathbf{l}$ and $\mathbf{u}$ be as in* (ILP)*, $a = \max\{2, \|A\|_\infty\}$ and $\tau + 1 = \mathrm{td}_D(A)$. Then there exists $D \in \mathbb{Z}^{m' \times n'}$, $\mathbf{b}' \in \mathbb{Z}^{m'}$ and $\mathbf{l}', \mathbf{u}' \in (\mathbb{Z} \cup \{\pm\infty\})^{n'}$, $n' \leq nt$, $t \leq n$, $m' \leq m \cdot \tau$, which define an extended formulation of $A\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$. Moreover, there exist matrices $A_1, \ldots, A_\tau \in \mathbb{Z}^{\tau \times t}$ and a tree T such that $D = T^D(A_1, \ldots, A_\tau)$ is a tree-fold constraint matrix, and all can be computed in time $f_{D\text{-}embed}(a, \mathrm{td}_D(A)) \cdot n^2$ for some computable function $f_{D\text{-}embed}$.*

References

1. Gautam Appa, Balázs Kotnyek, Konstantinos Papalamprou, and Leonidas Pitsoulis. Optimization with binet matrices. *Operations research letters*, 35(3):345–352, 2007.
2. Stephan Artmann, Robert Weismantel, and Rico Zenklusen. A strongly polynomial algorithm for bimodular integer linear programming. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 1206–1219. ACM, 2017.
3. Matthias Aschenbrenner and Raymond Hemmecke. Finiteness theorems in stochastic integer programming. *Foundations of Computational Mathematics*, 7(2):183–227, 2007.
4. Lin Chen and Dániel Marx. Covering a tree with rooted subtrees–parameterized and approximation algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 2801–2820. SIAM, 2018.
5. Jesús A. De Loera, Raymond Hemmecke, and Matthias Köppe. *Algebraic and Geometric Ideas in the Theory of Discrete Optimization*, volume 14 of *MOS-SIAM Series on Optimization*. SIAM, 2013.
6. Jesús A. De Loera, Raymond Hemmecke, and Jon Lee. On augmentation algorithms for linear and integer-linear programming: From Edmonds–Karp to Bland and beyond. *SIAM Journal on Optimization*, 25(4):2494–2511, 2015.
7. Pavel Dvořák, Eduard Eiben, Robert Ganian, Dušan Knop, and Sebastian Ordyniak. Solving integer linear programs with a small number of global variables and constraints. *arXiv preprint arXiv:1706.06084*, 2017.
8. András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.
9. Eugene C. Freuder. Complexity of K-tree structured constraint satisfaction problems. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 4–9, 1990.
10. Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. *Artificial Intelligence*, 2018.
11. Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Going beyond primal treewidth for (M)ILP. In Satinder P. Singh and Shaul Markovitch, editors, *AAAI*, pages 815–821. AAAI Press, 2017.
12. Raymond Hemmecke, Matthias Köppe, and Robert Weismantel. Graver basis and proximity techniques for block-structured separable convex integer minimization problems. *Mathematical Programming*, 145(1-2, Ser. A):1–18, 2014.
13. Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. N-fold integer programming in cubic time. *Mathematical Programming*, pages 1–17, 2013.
14. Dorit S. Hochbaum and J. George Shanthikumar. Convex separable optimization is not much harder than linear optimization. *Journal of the ACM*, 37(4):843–862, 1990.

15 Bart M. P. Jansen and Stefan Kratsch. A structural approach to kernels for ILPs: Treewidth and total unimodularity. In Nikhil Bansal and Irene Finocchi, editors, *ESA*, volume 9294 of *Lecture Notes in Computer Science*, pages 779–791. Springer, 2015.

16 Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. Empowering the configuration-IP – new PTAS results for scheduling with setups times. *arXiv preprint arXiv:1801.06460*, 2018.

17 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

18 Dušan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial n-fold integer programming and applications. In *ESA*, volume 87 of *LIPIcs*, pages 54:1–54:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. Extended version available at `https://arxiv.org/abs/1705.08657`.

19 Dušan Knop, Martin Koutecký, and Matthias Mnich. Voting and bribing in single-exponential time. In *STACS*, volume 66 of *LIPIcs*, pages 46:1–46:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

20 Hendrik W. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

21 Shmuel Onn. Nonlinear discrete optimization. *Zurich Lectures in Advanced Mathematics, European Mathematical Society*, 2010. `http://ie.technion.ac.il/~onn/Book/NDO.pdf`.

22 Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. A faster parameterized algorithm for treedepth. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP (1)*, volume 8572 of *Lecture Notes in Computer Science*, pages 931–942. Springer, 2014.

23 Alexander Shrijver. *Theory of linear and integer programming.* Interscience Series in Discrete Mathematics and Optimization. Wiley, 1986.

24 Éva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986.

25 Sergey I. Veselov and Aleksandr J. Chirkov. Integer program with bimodular matrix. *Discrete Optimization*, 6(2):220–222, 2009.

Finer Tight Bounds for Coloring on Clique-Width

Michael Lampis
Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243
LAMSADE, 75016, Paris, France
michail.lampis@dauphine.fr

Abstract

We revisit the complexity of the classical k-Coloring problem parameterized by clique-width. This is a very well-studied problem that becomes highly intractable when the number of colors k is large. However, much less is known on its complexity for small, concrete values of k. In this paper, we completely determine the complexity of k-Coloring parameterized by clique-width for any fixed k, under the SETH. Specifically, we show that for all $k \geq 3, \epsilon > 0$, k-Coloring cannot be solved in time $O^*\left((2^k - 2 - \epsilon)^{\text{cw}}\right)$, and give an algorithm running in time $O^*\left((2^k - 2)^{\text{cw}}\right)$. Thus, if the SETH is true, $2^k - 2$ is the "correct" base of the exponent for every k.

Along the way, we also consider the complexity of k-Coloring parameterized by the related parameter modular treewidth (mtw). In this case we show that the "correct" running time, under the SETH, is $O^*\left(\binom{k}{\lfloor k/2 \rfloor}^{\text{mtw}}\right)$. If we base our results on a weaker assumption (the ETH), they imply that k-Coloring cannot be solved in time $n^{o(\text{cw})}$, even on instances with $O(\log n)$ colors.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Clique-width, SETH, Coloring

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.86

1 Introduction

Graph Coloring (from now on simply Coloring) is one of the most intensely studied problems in theoretical computer science. In this classical problem we are given a graph $G = (V, E)$ and an integer k and are asked if we can partition V into k sets inducing edge-less graphs. Coloring is a notoriously hard problem as it remains NP-hard even in very restricted cases (e.g. 4-regular planar graphs [10]) and is essentially completely inapproximable in general [11, 30]. This intractability has motivated the study of the problem in the framework of parameterized complexity, especially with respect to structural graph parameters.[1]

Treewidth is by far the most widely studied among such graph parameters, and Coloring has long been known to be FPT by treewidth. This can be seen by either invoking Courcelle's theorem [4], or by applying a straightforward dynamic programming technique which, for each bag of a tree decomposition of width tw considers all possible k^{tw} colorings. Remarkably, thanks to the work of Lokshtanov, Marx, and Saurabh [23], we know that this basic algorithm is likely to be optimal, or at least that improving it would require a major breakthrough on SAT-solving, as, for any $k \geq 3, \epsilon > 0$, the existence of a $(k-\epsilon)^{\text{tw}}$ algorithm would contradict the Strong Exponential Time Hypothesis of Impagliazzo and Paturi [18, 19]. More recently, these lower bounds were strengthened, as Jaffke and Jansen showed that a $(k-\epsilon)^w$ algorithm would contradict the SETH when w is the graph's vertex edit distance from being a path

[1] In the remainder, we assume that the reader is familiar with the basics of parameterized complexity, such as the class FPT, as given in relevant textbooks [8, 13]

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 86; pp. 86:1–86:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

[20]. The same paper showed that the trivial algorithm can, however, be improved when one considers more restrictive parameters, such as vertex cover, but still not to the point that the base of the exponent becomes a constant. These results thus paint a very clear picture of the complexity of Coloring with respect to treewidth and its restrictions.

One of the drawbacks of treewidth is that it does not cover dense graphs, even if they have a very simple structure. This has led to the introduction of clique-width, which is by now (arguably) the second most well-studied parameter. The complexity of Coloring parameterized by clique-width has also been investigated. Even though Coloring is polynomial-time solvable when clique-width is constant, the best known algorithm for this case runs in time $n^{2^{O(\mathrm{cw})}}$ [21]. Fomin et al. [14] showed that Coloring is not FPT for clique-width (under standard assumptions), and this was recently followed up by Golovach et al. [17] who showed, somewhat devastatingly, that the aforementioned algorithm is likely to be optimal, as an algorithm running in $n^{2^{o(\mathrm{cw})}}$ would contradict the ETH. The problem thus seems to become significantly harder for clique-width, and this has, in part, motivated the study of alternative dense graph parameters, such as split-matching width [28], modular-width [15], and twin cover [16], all of which make Coloring FPT.

Contribution. Although the results mentioned above demonstrate a clear jump in the complexity of Coloring when moving from treewidth to clique-width, we observe that they leave open a significant hole: all the aforementioned hardness results for clique-width ([14, 17]) only apply to the case where k is large (polynomially related to the size of the graph). It is not hard to see that the problem becomes significantly easier if both cw and k are assumed to have moderate values; indeed Coloring is FPT when parameterized by $\mathrm{cw}+k$ [21]. Since the case where k is relatively small is arguably the most interesting scenario for most applications, we are strongly motivated to take a closer look at the complexity of Coloring parameterized by clique-width, in order to obtain a more fine-grained and quantitative estimate of the "price of generality" for this problem *for each fixed value of* k. Our aim is to reach tight bounds that paint a crisper picture of the complexity of the problem than what can be inferred by lower bounds parameterized only by clique-width, in the same way that the results of [23] do for k-Coloring on treewidth.

The main result of this paper is a lower bound which states that for all $k \geq 3, \epsilon > 0$, k-Coloring cannot be solved in time $O^*\left((2^k - 2 - \epsilon)^{\mathrm{cw}}\right)$, unless the SETH is false. This result gives a concrete, detailed answer to the question of how much the complexity of 3-Coloring, 4-Coloring, and generally k-Coloring, increases as one moves from treewidth to clique-width. As in the lower bound of [23], this result is established through a reduction from SAT. The main challenge here is that we need to pack a much larger amount of information per unit of width, and in particular that the graph induced by most label sets must be edge-less (otherwise many of the $2^k - 2$ choices we need to encode would be invalid). We work around this difficulty by a delicate use of the rename operation used in clique-width expressions.

Though having $2^k - 2$ in the base of the running time may seem somewhat curious (and certainly less natural than the k^{tw} bounds of [23]), we then go on to prove that this is the "correct" bound by giving a matching algorithm. The algorithm is based on standard DP techniques (including subset convolution [1, 29]), but requires a non-standard trick that "looks ahead" in the decomposition to lower the table size from $(2^k - 1)^{\mathrm{cw}}$ to $(2^k - 2)^{\mathrm{cw}}$. This improves the previously known DP algorithm of [21], which runs in $O^*\left(4^{k \cdot \mathrm{cw}}\right)$.

Beyond these results for clique-width we also consider the closely related parameters modular treewidth and modular pathwidth, which have more recently been considered as

more restricted versions of clique-width [25, 27]. The modular treewidth of a graph G is defined as the treewidth of the graph obtained from G if one collapses each twin class into a single vertex, where two vertices are twins if they have the same neighbors. By slightly altering our results for clique-width we tightly characterize the complexity of k-Coloring for these parameters: the problem is solvable in time $O^*\left(\binom{k}{\lfloor k/2 \rfloor}^{\mathrm{mtw}}\right)$, but not solvable in $O^*\left(\left(\binom{k}{\lfloor k/2 \rfloor}-\epsilon\right)^{\mathrm{mpw}}\right)$ under the SETH. Using the same reduction but relaxing the hypothesis to the ETH, we show that k-Coloring cannot be solved in time $n^{o(\mathrm{mpw})}$, and hence neither in time $n^{o(\mathrm{cw})}$ even on instances where $k = O(\log n)$. This can be seen as a strengthening of the lower bound of [14], which applies only to clique-width and uses $\Omega(n)$ colors. Our result is incomparable to the more recent double-exponential bound of [17] as it applies to the more restricted case where the number of colors is logarithmic, and is tight for this case. Indeed, any reduction giving a double-exponential bound, such as the one in [17], must inevitably use more than $\log n$ colors, otherwise it would contradict the aforementioned algorithms.

Non-binary CSPs. We mention as a secondary contribution of this paper a proof that, under the SETH, n-variable CSPs over an alphabet of size B cannot be solved in time $(B-\epsilon)^n$, for any B, ϵ. The interest of such a result is not so much technical (its proof is implicit in previous SETH-based bounds, going back to [23]), as conceptual. Such CSPs provide a convenient starting point for a SETH-based lower bound for *any base* of the exponential and hence allow us to isolate a technical part of such proofs from the main part of the reduction. This explicit statement on the hardness of CPSs has allowed the proofs of this paper to be significantly shortened, and may facilitate the design of other SETH-based hardness proofs.

2 Definitions and Preliminaries

We use standard graph-theoretic notation and assume that the reader is familiar with the basics of parameterized complexity, as well as standard notions such as treewidth [8, 13]. Let us recall the definition of clique-width (see [6, 5] for more details). A labeled graph G has clique-width w if it can be constructed using w labels and the following four basic operations: Introduce(i), for $i \in \{1, \ldots, w\}$, which constructs a single-vertex graph whose vertex has label i; Union(G_1, G_2), which constructs the disjoint union of two labeled graphs of clique-width w; Rename(i, j) which changes the label of all vertices labeled i to j; and Join(i, j) which adds all possible edges between vertices labeled i and vertices labeled j. Computing a graph's clique-width is NP-hard [12], and the best currently known approximation is exponential in clique-width [26]. In this paper, we will often assume that we are given together with a graph G, a clique-width expression constructing G. Since most of our results are negative, this only makes them stronger, as it shows that our lower bounds do not rely on the hardness of computing clique-width. We view a clique-width expression as a rooted binary tree, where the sub-tree rooted in each internal node represents the corresponding sub-graph of G. We use $\mathrm{cw}(G)$ to denote the minimum number of labels needed to construct a clique-width expression of G, and $\mathrm{tw}(G), \mathrm{pw}(G)$ to denote the treewidth and pathwidth of G respectively.

In a graph $G = (V, E)$ we say that $u, v \in V$ are false twins if $N(u) = N(v)$ and true twins if $N[u] = N[v]$, where $N[u] = N(u) \cup \{u\}$ denotes the closed neighborhood of u. We say that u, v are twins if they are true or false twins. We note that in any graph G the partition of vertices into twin classes is always unique, as the property of being twins is an equivalence relation [22]. Let G^t be the graph obtained from G by deleting from each twin class all but a single vertex. We define (following [25]) the modular treewidth of G, denoted $\mathrm{mtw}(G)$, as $\mathrm{tw}(G^t)$, and similarly the modular pathwidth $\mathrm{mpw}(G)$ as $\mathrm{pw}(G^t)$.

▶ **Lemma 1.** *For all* G, $\mathrm{pw}(G) \geq \mathrm{mpw}(G) \geq \mathrm{cw}(G) - 2$ *and* $\mathrm{pw}(G) \geq \mathrm{mpw}(G) \geq \mathrm{mtw}(G)$.

The Exponential Time Hypothesis (ETH) of Impagliazzo, Paturi, and Zane [19] states that there exists a constant $c_3 > 1$ such that 3-SAT on instances with n variables cannot be solved in time c_3^n. If the ETH is true then we can define, for all $q \geq 3$ a constant $c_q > 1$ such that q-SAT, that is, SAT on instances where clauses have maximum size q, cannot be solved in time c_q^n. The Strong Exponential Time Hypothesis (SETH) [18] states that $\lim_{q\to\infty} c_q = 2$, or equivalently that, for each $\epsilon > 0$ there exists a q such that q-SAT cannot be solved in $(2-\epsilon)^n$. We note that sometimes a slightly weaker form of the SETH is used, which states simply that SAT cannot be solved in $(2-\epsilon)^n$ for any $\epsilon > 0$. The two formulations are not currently known to be equivalent. In this paper we use the original, stronger formulation of [18] (see also e.g. [7]) which assumes that c_q tends to 2.

For any $q, B \geq 2$ we define the q-CSP-B problem as follows: we are given a set X of n variables which take values in $\{1, \ldots, B\}$, and a set $\mathcal{C}$ of q-constraints on X. A q-constraint c is defined by an ordered tuple $V(c)$ of q variables of X, and a set $S(c) \subseteq \{1, \ldots, B\}^q$ of satisfying assignments for c. The question is whether there exists an assignment $\sigma : X \to \{1, \ldots, B\}$ which satisfies all constraints $c \in \mathcal{C}$. We say that a constraint $c \in \mathcal{C}$ is satisfied if applying σ to $V(c)$ produces a tuple of assignments that appears in $S(c)$. To simplify presentation, we will assume that in the input the list $S(c)$ of the at most B^q satisfying assignments of each constraint is given explicitly, and that a q-CSP-B instance is allowed to contain constraints on fewer than q variables (as we can add dummy variables to a constraint).

3 SETH and Non-binary CSPs

The SETH states, informally, that as SAT clauses become larger, eventually the best algorithm for SAT is simply to try out all possible assignments to all variables. In this section we show that the same is essentially true for CSP with a larger, non-binary alphabet. The interest in presenting such a result is that very often we seek to prove a SETH-based lower bound showing that a problem does not admit an algorithm running in c^w, for some constant c and width parameter w (such as treewidth, or in our case clique-width). This becomes complicated when we reduce directly from SAT if c is not a power of 2 as one cannot make a one-to-one correspondence between binary SAT variables and "units of width" (in our case labels) in the new instance, which are intended to encode c choices. As a result, essentially all known SETH lower bounds of this form include as part of their construction a group gadget, which maps every t variables of the original SAT instance to p elements of the new problem, for appropriately chosen integers p, t (see e.g. [3, 9, 20, 23]). Such gadgets are, however, often cumbersome to design, because they require a problem-specific trick that expresses a mapping of assignments from a binary to a non-binary domain. We therefore prefer to construct a custom-made CSP with a convenient running time bound, which will later allow us to reduce directly to the problem we are interested in (Coloring on clique-width), in a way that maps exactly one variable to one clique-width label. This will allow our SETH-based bounds to be significantly simplified, as we will no longer have to worry about a discrepancy between the bases of the exponentials.

▶ **Theorem 2.** *For any* $B \geq 2$, $\epsilon > 0$ *we have the following: if the SETH is true, then there exists a* q *such that* n*-variable* q-CSP-B *cannot be solved in time* $O^*\left((B-\epsilon)^n\right)$.

4 SETH-based Lower Bound for Clique-width

In this section we present our main lower bound result stating that k-Coloring cannot be solved in time $O^*\left((2^k-2-\epsilon)^{\text{cw}}\right)$, for any $k \geq 3, \epsilon > 0$, under the SETH. In Section 4.1 we present some basic gadgets that will also be of use in our lower bound for modular pathwidth (Section 5). We then present the main part of the proof in Section 4.2.

4.1 List Coloring and Basic Gadgets

The high-level machinery that we will make use of in our reduction consists of two major points: first, we would like to be able to express implication constraints, that is, constraints of the form "if vertex u received color c_1, then vertex v must receive color c_2"; second, we would like to express disjunction constraints of the form "at least one of the vertices of the set $S \subseteq V$ must take a special color 1". We build this machinery in the following steps: first, we show that we can (almost) equivalently produce an instance of the more general List Coloring problem; then we use the ability to construct lists to make *weak edge gadgets*, which for a given pair of vertices (u_1, u_2) rule out a specific pair of assigned colors; using these weak edges we construct the aforementioned implication gadgets; and finally we are able to implement OR constraints using paths on vertices with appropriate lists.

We give all details for these constructions below. We remark however, for the convenience of the reader, that a high-level understanding of the informal meaning of implication gadgets and OR gadgets (precisely stated in Lemmata 7, 10) is already sufficient to follow the description of the main part of the reduction, given in Section 4.2. See also Figure 1.

List Coloring. To simplify the presentation of our reduction it will be convenient to use a slightly more general problem. In List Coloring, we are given a graph G and a list of possible colors associated with each vertex and are asked if there is a proper coloring such that each vertex uses a color from its list. This problem clearly generalizes k-Coloring, as all lists may be $\{1, \ldots, k\}$. We will make use of a reduction in the opposite direction.

▶ **Lemma 3.** *There is a polynomial-time algorithm which, given an instance of* List Coloring *on a graph G where all lists are subsets of $\{1, \ldots, k\}$, transforms it into an equivalent instance of k-*Coloring *on a graph G'. Furthermore, the algorithm transforms a clique-width expression of G with* cw *labels, to a clique-width expression of G' with* $\text{cw} + k$ *labels. If all twins of G share the same list, the algorithm transforms a modular path decomposition of width* mpw *for G, to a modular path decomposition of G' of width* $\text{mpw} + k$.

Weak Edges and Implications. Normally, the existence of an edge (u, v) in an instance of Coloring forbids the vertices u, v from obtaining the same color, whatever that color may be. We will find it convenient to construct edges that forbid only a specific pair of colors from appearing on u, v, while allowing any other combination of colors to be used on these two vertices. Similar versions of this gadget have appeared before, for example [23, 24].

▶ **Definition 4.** For two vertices u_1, u_2 of a graph G and two colors c_1, c_2 a (c_1, c_2)-*weak edge* from u_1 to u_2 consists of the following:

1. Three new vertices v_1, v_2, v_3 such that $\{u_1, v_1, v_2, v_3, u_2\}$ induces a path in this order, with endpoints u_1, u_2, and v_1, v_2, v_3 having no edges to the rest of G.
2. If $c_1 \neq c_2$ let c' be a color distinct from c_1, c_2. We assign to v_1, v_2, v_3 the lists $\{c_1, c_2\}, \{c_2, c'\}, \{c_2, c'\}$. If $c_1 = c_2$, we assign lists $\{c_1, c'\}, \{c', c''\}, \{c_1, c''\}$ to v_1, v_2, v_3 respectively, where c', c'' are two distinct colors, different from c_1.

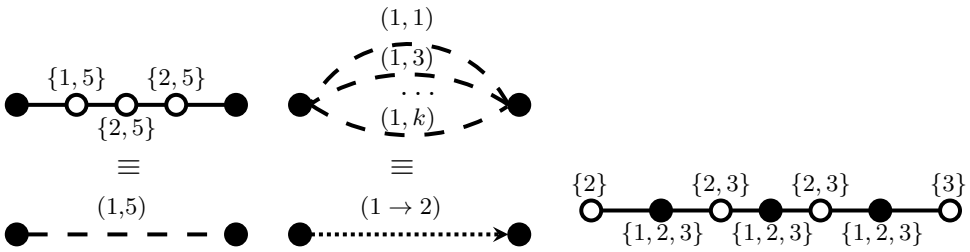


Figure 1 Basic gadgets, where empty vertices are internal and solid vertices are endpoints that will be connected to the rest of the graph. On the left, a weak edge that forbids the combination $(1, 5)$ on its endpoints. In the middle, an implication that forces color 2 on the right if color 1 is used on the left. On the right an OR gadget: one of the solid vertices must take color 1.

▶ **Lemma 5.** *Let G be an instance of* List Coloring *that contains a (c_1, c_2)-weak edge from u_1 to u_2. Then G does not admit a valid coloring that assigns colors (c_1, c_2) to (u_1, u_2). Furthermore, if G' denotes the graph obtained by deleting the internal vertices of the weak edge, any proper list coloring of G' that does not assign c_1 to u_1 or does not assign c_2 to u_2 can be extended to a proper list coloring of G.*

Let us now use the weak edges we have defined above to construct an implication gadget. The intuitive meaning of placing an implication gadget from a vertex u_1 to a vertex u_2 is to impose the constraint that if u_1 is assigned color c_1, then u_2 must be assigned color c_2.

▶ **Definition 6.** For two vertices u_1, u_2 and two colors (c_1, c_2) a $(c_1 \to c_2)$-implication from vertex u_1 to vertex u_2 is constructed as follows: for each color $c' \neq c_2$, we add a (c_1, c')-weak edge from u_1 to u_2.

▶ **Lemma 7.** *Let G be an instance of* List Coloring *that contains a $(c_1 \to c_2)$-implication from u_1 to u_2. Then G does not admit a list coloring that gives color c_1 to u_1 and a color $c' \neq c_2$ to u_2. Furthermore, if G' is the graph obtained from G by deleting the internal vertices of the implication gadget, any coloring of G' that either does not assign c_1 to u_1, or assigns c_2 to u_2 can be extended to a coloring of G.*

▶ **Lemma 8.** *Let G be an instance of* List Coloring*, and G' be the graph obtained from G by replacing every (c_1, c_2)-weak edge or $(c_1 \to c_2)$-implication gadget with endpoints u_1, u_2 with an edge (u_1, u_2) (or simply deleting the internal vertices of the weak edge if (u_1, u_2) already exists). Then* $\mathrm{pw}(G) \leq \mathrm{pw}(G') + 3$.

OR gadgets. We will also make use of a gadget that forces any valid list coloring of a graph to assign a special color 1 to one vertex out of a set of vertices. Invariably, the idea will be that this will be a color that activates some implications, allowing us to propagate information about the coloring between parts of the graph. We recall that a similar version of an OR gadget was also used in [23].

▶ **Definition 9.** An OR gadget on an independent set of vertices S, denoted $\mathrm{OR}(S)$, is constructed as follows: we assign list $\{1, 2, 3\}$ to all vertices of S; we construct a new set S' of internal vertices (that will not be connected to the rest of G), such that $|S'| = |S| + 1$ and $S \cup S'$ induces a path alternating between vertices of S and S'; we assign list $\{2, 3\}$ to all vertices of S', except the two endpoints of the path, which receive lists $\{2\}, \{3\}$, respectively.

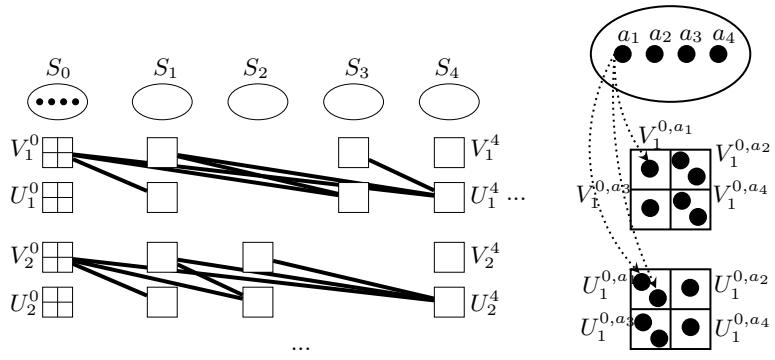


Figure 2 Left: high-level view of the reduction. Rows correspond to variables, columns to constraints. Here, variable x_1 appears in constraints c_0, c_1, c_3, c_4. Right: connections between the OR gadgets $\mathrm{OR}(S_j)$ and the V_i^j, U_i^j sets. Giving color 1 to a_1 represents selecting this assignment. This forces the use of some colors in V_1^{0,a_1} and the complementary set in U_1^{0,a_1}.

▶ **Lemma 10.** *If G is a* List Coloring *instance that contains an OR(S) gadget then G does not admit a list coloring that does not use color* 1 *in any vertex of S. Furthermore, for any vertex $u \in S$, there exists a proper list coloring of the graph induced by the gadget that assigns color* 1 *only to u.*

4.2 Reduction for Clique-width

▶ **Theorem 11.** *For any $k \geq 3, \epsilon > 0$, if there exists an algorithm solving k-coloring in time $O^*\left((2^k - 2 - \epsilon)^{\mathrm{cw}}\right)$, where* cw *is the input graph's clique-width, then the SETH is false.*

The proof of Theorem 11 consists of a reduction from a CSP produced by Theorem 2. Before giving details, let us give some intuition. Our new instance will be a graph and a clique-width expression with, roughly, n labels, where n is the number of variables of the CSP instance. The set of colors used in each label will encode the value given to a variable in a satisfying assignment. As a result, with k colors, we will have $2^k - 2$ encodings available, as every label set uses at least one color, but will never use all k colors. To verify that these assignments are correct, we will construct for each constraint an OR gadget which forces the use of color 1 on a vertex representing a particular assignment. This assignment dictates the value of each variable of the constraint, and therefore the set of colors used in some of our label sets. To verify that the assignment is consistent we use implication gadgets that force some auxilliary vertices to receive the complement of the colors dictated by the constraint assignment, and then connect these with the vertices encoding the true assignment. If the assignment used is truly consistent, these edges will end up being properly colored.

Construction. We are given $k \geq 3, \epsilon > 0$. Let $B = 2^k - 2$. Let q be the smallest integer such that n-variable q-CSP-B does not admit an $O^*\left((B - \epsilon)^n\right)$ algorithm. According to Theorem 2, q exists if the SETH is true, and it depends only on B, ϵ. Consider an arbitrary n-variable instance of q-CSP-B, call it ϕ. We use the existence of the supposed $O^*\left((2^k - 2 - \epsilon)^{\mathrm{cw}}\right)$ algorithm to obtain an $O^*\left((B - \epsilon)^n\right)$ algorithm that decides ϕ, contradicting the SETH.

We define in some arbitrary way a translation function T which, given a value $v \in \{1, \ldots, B\}$ returns a non-empty proper subset of $\{1, \ldots, k\}$. We make sure that T is defined in such a way that it is one-to-one; this is possible since the number of non-empty proper subsets of $\{1, \ldots, k\}$ is exactly $B = 2^k - 2$.

Let $X = \{x_1, \ldots, x_n\}$ be the set of the n variables of the q-CSP-B instance and $C = \{c_0, \ldots, c_{m-1}\}$ the set of its m constraints. Let $L = 3m(nk+1)$. We now construct our graph, where if we don't specify the list of a vertex it can be assumed to be $\{1, \ldots, k\}$. For each $j \in \{0, \ldots, L-1\}$ we do the following:

1. Let $j' = j \bmod m$ and let S be the set of satisfying assignments of the constraint $c_{j'}$. We construct an independent set of vertices S_j that contains a vertex for every assignment of S. We construct an $\mathrm{OR}(S_j)$ gadget on these vertices.
2. For each x_i which appears in $c_{j'}$ and for each assignment $a \in S$ we do the following:
   a. Let $v \in \{1, \ldots, B\}$ be the value given to x_i by the assignment a. Construct an independent set $V_i^{j,a}$ of $|T(v)|$ vertices and an independent set $U_i^{j,a}$ of $k - |T(v)|$ vertices. Recall that $T(v)$, the translation function, returns a set of size between 1 and $k-1$, so both these sets are non-empty.
   b. For each color $c \in T(v)$ select a distinct vertex in $V_i^{j,a}$ and add a $(1 \to c)$-implication gadget from the vertex that represents the assignment a in S_j to this vertex of $V_i^{j,a}$.
   c. For each color $c \in \{1, \ldots, k\} \setminus T(v)$ select a distinct vertex in $U_i^{j,a}$ and add a $(1 \to c)$-implication gadget from the vertex that represents the assignment a in S_j to this vertex of $U_i^{j,a}$.
   d. Connect all vertices of $U_i^{j,a}$ with all vertices of previously constructed sets $V_i^{l,a'}$, for all $l < j$ and all assignments a'.

This completes the construction, and we call the constructed List Coloring instance $G(\phi)$. The intended meaning is that the sets $V_i^{j,a}$ will use a set of colors that encodes the value of the variable x_i, while the sets $U_i^{j,a}$ will use colors from the complement of this set.

▶ **Lemma 12.** *If ϕ is a satisfiable q-CSP-B instance, then $G(\phi)$ admits a proper list coloring.*

Proof. Suppose that we have a satisfying assignment for ϕ which gives value v_i to variable x_i. The invariant we will maintain is that for all j, a, all vertices of sets $V_i^{j,a}$ will use only colors from $T(v_i)$, while all vertices of sets $U_i^{j,a}$ will use only colors from $\{1, \ldots, k\} \setminus T(v_i)$. As a result, all edges added in step 2d will be properly colored. The rest of the graph will be easy to color if we respect the informal meaning of OR and implication gadgets.

More specifically, for each $\mathrm{OR}(S_j)$ gadget we let $j' = j \bmod m$ and consider the constraint $c_{j'}$. The supposed assignment to ϕ assigns to the variables of the constraint values consistent with a satisfying assignment a of $c_{j'}$. We give color 1 to the corresponding vertex of S_j. We use colors $\{2, 3\}$ to color all remaining vertices of the OR gadget. Note that the OR gadget is connected to the rest of the graph only through implication gadgets activated by color 1. Hence, by Lemma 7 we can remove all non-activated implication gadgets. For the remaining, activated implication gadgets we color their other endpoints, which are found in the sets $V_i^{j,a}$ and $U_i^{j,a}$ with the unique viable color. For every other assignment $a' \neq a$ we color all vertices of $V_i^{j,a'}$ using a color we used in $V_i^{j,a}$, and the vertices of $U_i^{j,a'}$ using a color we used in $U_i^{j,a}$.

The promised invariant is maintained, as the vertices of $V_i^{j,a}$ are forced to receive colors from $T(v_i)$, while vertices of $U_i^{j,a}$ are forced to receive colors from the complementary set. Thus, all edges of step 2d are properly colored, and since we also properly colored the OR gadgets and implication gadgets, we have a proper coloring of the whole graph. ◀

▶ **Lemma 13.** *If $G(\phi)$ admits a proper list coloring, then ϕ is a satisfiable q-CSP-B instance.*

Proof. Suppose we have a list coloring of $G(\phi)$ given by the function $\mathbf{c} : V \to \{1, \ldots, k\}$. For a set $V' \subseteq V$ we will write $\mathbf{c}(V')$ to denote the set of colors used by $\mathbf{c}$ for vertices of V', that is, $\mathbf{c}(V') = \{c \mid \exists u \in V',\ \mathbf{c}(u) = c\ \}$. Let $j \in \{0, \ldots, L-1\}$, $j' = j \bmod m$, and S be the set

of satisfying assignment of the constraint $c_{j'}$, which contains a variable x_i. Consider the set $V_i^j = \cup_{a \in S} V_i^{j,a}$. We define the *candidate assignment* of x_i at index j as $v_i^j := T^{-1}(\mathbf{c}(V_i^j))$. In other words, to obtain the candidate assignment for x_i at index j, we take the union of all colors used in $V_i^{j,a}$, and then translate this set back into a value in $\{1, \ldots, B\}$.

We observe that for all i, j, such that x_i appears in $c_{j'}$, where $j' = j \bmod m$, there exists an assignment a such that $\mathbf{c}(V_i^{j,a}) = \{1, \ldots, k\} \setminus \mathbf{c}(U_i^{j,a})$. To see this, note that by Lemma 10, one of the vertices of the $\mathrm{OR}(S_j)$ gadget must have received color 1, say the vertex that corresponds to assignment a. All the implications incident on this vertex are therefore activated, which means that, if a gives value $v \in \{1, \ldots, B\}$ to x_i, then $\mathbf{c}(V_i^{j,a}) = T(v)$ and $\mathbf{c}(U_i^{j,a}) = \{1, \ldots, k\} \setminus T(v)$ (because of the implications of steps 2b,2c and Lemma 7).

A key observation now is the following: for all $j_2 > j_1$ and for all i such that variable x_i appears in constraints $c_{j_1'}, c_{j_2'}$ with $j_1' = j_1 \bmod m$, $j_2' = j_2 \bmod m$, we have $\mathbf{c}(V_i^{j_1}) \subseteq \mathbf{c}(V_i^{j_2})$. In other words, the set of colors used in V_i^j can only increase as j increases. To see this, suppose that there exists $c \in \mathbf{c}(V_i^{j_1}) \setminus \mathbf{c}(V_i^{j_2})$. As argued in the previous paragraph, there exists an assignment a_2 such that $\mathbf{c}(V_i^{j_2}) \supseteq \mathbf{c}(V_i^{j_2,a_2}) = \{1, \ldots, k\} \setminus \mathbf{c}(U_i^{j_2,a_2})$. Because $c \notin \mathbf{c}(V_i^{j_2,a_2})$ we must have $c \in \mathbf{c}(U_i^{j_2,a_2})$, but because of step 2d, all vertices of $U_i^{j_2,a_2}$ are connected to all of $V_i^{j_1}$. Since $c \in \mathbf{c}(V_i^{j_1})$, this contradicts the correctness of the coloring.

The property established in the previous paragraph implies that for each i there exist at most k distinct candidate assignments v_i^j we can obtain for different values of j, as each assignment is obtained by translating the set of colors used in V_i^j, this set only increases, it always contains at least one color and at most k colors. Let us say that an index j_1 is problematic if, for some $i \in \{1, \ldots, n\}$ we have the following: x_i appears in constraint $c_{j_1'}$, where $j_1' = j_1 \bmod m$; and if j_2 is the minimum index such that $j_2 > j_1$ and x_i appears in constraint $c_{j_2'}$, where $j_2' = j_2 \bmod m$, then $v_i^{j_1} \neq v_i^{j_2}$. In other words, an index is problematic if the candidate assignment it produces for a variable disagrees with the candidate assignment produced for the same variable in the next index that involves this variable. It is not hard to see that there are at most kn problematic indices, because for each variable there are at most k problematic indices. Therefore, since $L = 3m(nk + 1)$, by pigeonhole principle, there exists an interval L' of at least $3m$ consecutive non-problematic indices.

We now obtain an assignment for the original instance as follows: for each variable i we take an index $j \in L'$ such that x_i appears in constraint $c_{j'}$, where $j' = j \bmod m$, and give x_i the candidate value v_i^j from that index. Observe that, by the definition of L' the index we select is irrelevant, as all candidate values are constant throughout the interval L'.

We claim that this is a satisfying assignment. Suppose not, so there exists an unsatisfied constraint $c_{j'}$. Because L' contains $3m$ consecutive indices, there exists three indices $j_1 < j_2 < j_3 \in L'$ such that $j' = j_1 \bmod m = j_2 \bmod m = j_3 \bmod m$. We observe that for all variables x_i appearing in $c_{j'}$ we have given value $v_i^{j_2}$, that is the candidate value obtained at index j_2, since all indices in L' give the same candidate values to all variables.

Now, there exists a vertex in S_{j_2} that received color 1, representing an assignment a. If the assignment we produced is not consistent with a, there exists a variable x_i such that we have given x_i value $v = v_i^{j_2}$, while a gives it value $v' \neq v$. Consider now the set $V_i^{j_2,a}$. Because of the implication gadgets, it uses the colors $T(v') \neq T(v)$. If there exists $c \in T(v) \setminus T(v')$ then $c \in \mathbf{c}(U_i^{j_2,a})$. But $U_i^{j_2,a}$ is connected to all vertices of $V_i^{j_1}$, which, we assumed use all colors of $T(v)$, therefore also color c, contradicting the correctness of the coloring. If on the other hand there exists $c \in T(v') \setminus T(v)$, then since $\mathbf{c}(V_i^{j_3}) = T(v)$, there exists an a' such that $\mathbf{c}(U_i^{j_3,a'}) = \{1, \ldots, k\} \setminus T(v)$. Therefore, $c \in \mathbf{c}(U_i^{j_3,a'})$, while $c \in \mathbf{c}(V_i^{j_2,a})$, and by step 2d these sets are connected, again obtaining a contradiction. We therefore conclude that we must have a consistent satisfying assignment. ◀

▶ **Lemma 14.** *$G(\phi)$ can be constructed in time polynomial in $|\phi|$, and we have* $\mathrm{cw}(G(\phi)) \leq n + O(qk^2B^q) = n + f(\epsilon, k)$ *for some function f.*

Proof. For fixed $k \geq 3, \epsilon > 0$, we have that $B = 2^k - 2$ and q is a constant that only depends on B, ϵ (that is, on k, ϵ). Each constraint of the q-CSP-B instance has at most B^q satisfying assignments. Therefore, it is not hard to see that the whole construction can be performed in polynomial time, if k, ϵ, B, q are constants. For clique-width we use the following labels:

1. n main labels, representing the variables of ϕ.
2. A single junk label. Its informal meaning is that a vertex that receives this label will not be connected to anything else not yet introduced in the graph.
3. $O(B^q)$ constraint work labels.
4. $O(qk^2B^q)$ variable-constraint incidence work labels.

To give a clique-width expression we will describe how to build the graph, following essentially the steps given in the description of the construction by maintaining the following invariant: before starting iteration j, all vertices of the set $\bigcup_{j'<j} V_i^{j',a}$ (where we take the union over all assignments a), have label i, and all other vertices have the junk label.

This invariant is vacuously satisfied before the first iteration, since the graph is empty. Suppose that for some $j \in \{0, \ldots, L-1\}$ the invariant is true. We use the $O(B^q)$ constraint work labels to introduce the vertices of the $\mathrm{OR}(S_j)$ gadget of step 1, giving each vertex a distinct label. We use join operations to construct the internal edges of the OR gadget.

Then, for each variable x_i that appears in the current constraint we do the following: we use $O(k^2B^q)$ of the variable-constraint incidence work labels to introduce the vertices of $V_i^{j,a}$, $U_i^{j,a}$ as well as the implication gadgets connecting these to S_j. Again we use a distinct label for each vertex, but the number of vertices (including internal vertices of the implication gadgets) is $O(k^2B^q)$, so we have sufficiently many labels to use distinct labels for each of the q variables of the constraint. We use join operations to add the edges inside all implication gadgets. Then we use join operations to connect $U_i^{j,a}$ to all vertices $\bigcup_{j'<j} V_i^{j',a}$, for $j' < j$. This is possible, since the invariant states that all the vertices of $\bigcup_{j'<j} V_i^{j',a}$ have the same label. We then rename all the vertices of $U_i^{j,a}$, for all a to the junk label, and do the same also for internal vertices of all implication gadgets. We proceed to the next variable of the same constraint and handle it using its own $O(k^2B^q)$ labels. Once we have handled all variables of the current constraint, we rename all vertices of each $V_i^{j,a}$ to label i for all a. We then rename all vertices of the $\mathrm{OR}(S_j)$ gadget to the junk label and increase j by 1. It is not hard to see that we have maintained the invariant and constructed all edges induced by the vertices introduced in steps up to j, so repeating this process constructs the graph. ◀

5 Modular Pathwidth and ETH

We present a lower bound on the complexity of k-Coloring parameterized by modular pathwidth. Specifically, we show that, under the SETH, no algorithm can solve k-Coloring in $O^*\left(\left(\binom{k}{\lfloor k/2 \rfloor} - \epsilon\right)^{\mathrm{mpw}}\right)$. This bound is somewhat weaker than the one in Theorem 11, which is natural since modular pathwidth is more restrictive than clique-width. As we see in Section 6, however, this bound is tight. We complete this section by performing the same reduction with the ETH, rather than the SETH, as a starting point. Under this weaker assumption we prove that k-Coloring does not admit an algorithm running in $n^{o(\mathrm{mpw})}$, even when $k = O(\log n)$, which implies that the problem does not admit an algorithm running in $2^{o(k \cdot \mathrm{mpw})}$. This is tight, and also applies to clique-width (Lemma 1). In this way, our reduction gives an alternative proof that k-Coloring is unlikely to be FPT parameterized by clique-width, even in instances with a logarithmic number of colors.

5.1 SETH-based Lower Bound

▶ **Theorem 15.** *For any $k \geq 3, \epsilon > 0$, if there exists an algorithm solving k-coloring in time $O^*\left(\left(\binom{k}{\lfloor k/2 \rfloor} - \epsilon\right)^{\text{mpw}}\right)$, where* mpw *is the graph's modular pathwidth, then the SETH is false.*

As in Theorem 11, we begin our reduction from a q-CSP-B instance, where the alphabet size B is equal to the base of the exponential in our lower bound. The intuition will be that the "important" vertices of the bags in a modular tree decomposition of our graph will correspond to classes of $\lfloor k/2 \rfloor$ true twin vertices. The set of $\lfloor k/2 \rfloor$ colors used to color them will encode the value of one variable of the original instance.

Construction. We are given some $k \geq 3, \epsilon > 0$. Let $B = \binom{k}{\lfloor k/2 \rfloor}$. Let q be the smallest integer such that n-variable q-CSP-B does not admit an $O^*\left((B-\epsilon)^n\right)$ algorithm. Consider an arbitrary n-variable instance of q-CSP-B, call it ϕ. We use the existence of the supposed algorithm to obtain an $O^*\left((B-\epsilon)^n\right)$ algorithm that decides ϕ, contradicting the SETH.

As in Theorem 11 we define a one-to-one translation function T. This time, when T is given as input a value $v \in \{1, \ldots, B\}$, it returns a subset of $\{1, \ldots, k\}$ of cardinality $\lfloor k/2 \rfloor$. Let $X = \{x_1, \ldots, x_n\}$ be the set of the n variables of the q-CSP-B instance and $C = \{c_0, \ldots, c_{m-1}\}$ the set of its m constraints. We construct our graph $G(\phi)$ as follows, where if we don't specify a list for a vertex its list is $\{1, \ldots, k\}$:

1. For each variable $i \in \{1, \ldots, n\}$ we construct a clique V_i on $\lfloor k/2 \rfloor$ vertices.
2. For each $j \in \{0, \ldots, m-1\}$, let S be the set of satisfying assignments of the constraint c_j. We construct an independent set S_j with one vertex for each element of S. We construct an $\text{OR}(S_j)$ gadget on the set S_j.
3. For each $j \in \{0, \ldots, m-1\}$, each satisfying assignment a for the constraint c_j, and each variable x_i appearing in c_j we do the following:
   a. We construct a set $U_i^{j,a}$ of $\lceil k/2 \rceil$ vertices.
   b. Suppose a assigns value v to x_i. For each color $c \in \{1, \ldots, k\} \setminus T(v)$ we select a vertex of $U_i^{j,a}$. We construct a $(1 \to c)$-implication gadget from the vertex representing a in S_j to this vertex of $U_i^{j,a}$.
   c. We connect all vertices of $U_i^{j,a}$ with all vertices of V_i.

▶ **Lemma 16.** *If ϕ is a satisfiable q-CSP-B instance, then $G(\phi)$ admits a proper list coloring.*

▶ **Lemma 17.** *If $G(\phi)$ admits a proper list coloring, then ϕ is a satisfiable q-CSP-B instance.*

▶ **Lemma 18.** *$G(\phi)$ can be constructed in time polynomial in $|\phi|$, and $\text{mpw}(G) \leq n + O(1)$.*

5.2 ETH-based Lower Bound

▶ **Theorem 19.** *If there exists an algorithm that solves k-Coloring on instances with n vertices and $k = O(\log n)$ in time $n^{o(\text{mpw})}$, then the ETH is false. As a result, if there is an algorithm solving k-Coloring in time $2^{o(k \cdot \text{mpw})}$, then the ETH is false.*

6 Algorithms

We present two algorithms establishing that the lower bounds of Sections 4,5 are essentially tight. Though both algorithms are based on standard techniques, we remark that the algorithm for clique-width requires some extra effort to obtain a DP of the promised size.

6.1 Clique-width algorithm

Our algorithm is based on standard DP. Its basic idea is that a partial solution is characterized by the set of colors it uses to color a set of vertices that share the same label. This leads to a DP table of size $(2^k-1)^{\mathrm{cw}}$, by observing that for any non-empty label set, any viable partial solution will use at least one color, hence there are 2^k-1 possible subsets of $\{1,\ldots,k\}$ to consider. To improve this to $(2^k-2)^{\mathrm{cw}}$, which would match the lower bound of Theorem 11 we need a further idea which will allow us to also rule out the set that uses all k colors.

Let t be a node of the binary tree representing the clique-width expression of G, and let V_t^i be the set of vertices that have label $i \in \{1,\ldots,\mathrm{cw}\}$ in the labeled graph G_t produced by the sub-expression rooted at t. We will say that V_t^i is a *live* label set if there exists an edge in G that is incident on a vertex of V_t^i and does not appear in G_t. In other words, a label set is live if there is a join operation that involves its vertices which has not yet appeared in t. The main observation is that live label sets cannot use all k colors in a valid partial solution, since then the subsequent join operation will fail. Non-live label sets, on the other hand, are irrelevant, since if the coloring is already valid for such a set it is guaranteed to remain valid. Our DP algorithm will therefore keep track of the partial colorings only of live label sets, and thus produce a DP table of size $(2^k-2)^{\mathrm{cw}}$. In this sense, our DP algorithm is slightly non-standard, as part of its procedure involves "looking ahead" in the graph to determine if a label set is live or not. What remains is the problem of implementing the DP so that it takes time linear in the table size; this is handled using the techniques introduced in [1, 29].

▶ **Theorem 20.** *There is an algorithm which, given a graph G, an integer k, and a clique-width expression for G with* cw *labels decides if G is k-colorable in time $O^*\left((2^k-2)^{\mathrm{cw}}\right)$.*

6.2 Modular Treewidth Algorithm

For modular treewidth, we remark that k-Coloring for this parameter can be seen as an equivalent version of Multi-Coloring parameterized by treewidth. In Multi-Coloring, each vertex v has a demand $b(v)$, and we are asked to assign $b(v)$ distinct colors to each vertex so that neighboring vertices have disjoint colors (see e.g. [2]). In our context, the vertex representing a class of b true twins corresponds to a vertex with demand b.

▶ **Theorem 21.** *There is an algorithm which, given a graph G, an integer k, and a modular tree decomposition of G of width* mtw*, decides if G is k-colorable in time $O^*\left(\binom{k}{\lfloor k/2\rfloor}^{\mathrm{mtw}}\right)$.*

7 Conclusions – Open Problems

We have given tight bounds for k-Coloring parameterized by clique-width, complementing previously known bounds for treewidth. A natural question is now how robust these bounds are, especially in the context of approximation. Specifically, does there exist a constant factor approximation algorithm for k-Coloring running in $O^*\left((k-\epsilon)^{\mathrm{tw}}\right)$ or $O^*\left((2^k-2-\epsilon)^{\mathrm{cw}}\right)$? Current knowledge cannot even rule out the existence of such algorithms with a small *additive* approximation error and this area is still largely unexplored.

References

1 Hans L. Bodlaender, Erik Jan van Leeuwen, Johan M. M. van Rooij, and Martin Vatshelle. Faster algorithms on branch and clique decompositions. In *MFCS*, volume 6281 of *Lecture Notes in Computer Science*, pages 174–185. Springer, 2010.

2 Marthe Bonamy, Lukasz Kowalik, Michal Pilipczuk, Arkadiusz Socala, and Marcin Wrochna. Tight lower bounds for the complexity of multicoloring. In *ESA*, volume 87 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

3 Glencora Borradaile and Hung Le. Optimal dynamic program for r-domination problems over tree decompositions. In *IPEC*, volume 63 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

4 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

5 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

6 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

7 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

9 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS*, pages 150–159. IEEE Computer Society, 2011.

10 David P. Dailey. Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete. *Discrete Mathematics*, 30(3):289–293, 1980.

11 Uriel Feige and Joe Kilian. Zero knowledge and the chromatic number. *J. Comput. Syst. Sci.*, 57(2):187–199, 1998.

12 Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is np-complete. *SIAM J. Discrete Math.*, 23(2):909–939, 2009.

13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

14 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010.

15 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In *IPEC*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013.

16 Robert Ganian. Twin-cover: Beyond vertex cover in parameterized algorithmics. In *IPEC*, volume 7112 of *Lecture Notes in Computer Science*, pages 259–271. Springer, 2011.

17 Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Cliquewidth III: the odd case of graph coloring parameterized by cliquewidth. In *SODA*, pages 262–273. SIAM, 2018.

18 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

20 Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. In *CIAC*, volume 10236 of *Lecture Notes in Computer Science*, pages 345–356, 2017.

21 Daniel Kobler and Udi Rotics. Edge dominating set and colorings on graphs with fixed clique-width. *Discrete Applied Mathematics*, 126(2-3):197–221, 2003.

22 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/s00453-011-9554-x.

23 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In *SODA*, pages 777–789. SIAM, 2011.

24 Dániel Marx and Valia Mitsou. Double-exponential and triple-exponential bounds for choosability problems parameterized by treewidth. In *ICALP*, volume 55 of *LIPIcs*, pages 28:1–28:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

25 Stefan Mengel. Parameterized compilation lower bounds for restricted cnf-formulas. In *SAT*, volume 9710 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2016.

26 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006.

27 Daniël Paulusma, Friedrich Slivovsky, and Stefan Szeider. Model counting for CNF formulas of bounded modular treewidth. *Algorithmica*, 76(1):168–194, 2016.

28 Sigve Hortemo Sæther and Jan Arne Telle. Between treewidth and clique-width. *Algorithmica*, 75(1):218–253, 2016.

29 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *ESA*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009.

30 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.

A Centralized Local Algorithm for the Sparse Spanning Graph Problem

Christoph Lenzen
Max Planck Institute for Informatics, Saarbrücken, Germany
clenzen@mpi-inf.mpg.de

Reut Levi[1]
Weizmann Institute of Science, Rehovot, Israel
reut.levi@weizmann.ac.il
https://orcid.org/0000-0003-3167-1766

Abstract

Constructing a sparse *spanning subgraph* is a fundamental primitive in graph theory. In this paper, we study this problem in the Centralized Local model, where the goal is to decide whether an edge is part of the spanning subgraph by examining only a small part of the input; yet, answers must be globally consistent and independent of prior queries.

Unfortunately, maximally sparse spanning subgraphs, i.e., spanning trees, cannot be constructed efficiently in this model. Therefore, we settle for a spanning subgraph containing at most $(1+\epsilon)n$ edges (where n is the number of vertices and ϵ is a given approximation/sparsity parameter). We achieve a query complexity of $\tilde{O}(\text{poly}(\Delta/\varepsilon)n^{2/3})$,[2] where Δ is the maximum degree of the input graph. Our algorithm is the first to do so on arbitrary bounded degree graphs. Moreover, we achieve the additional property that our algorithm outputs a spanning subgraph of bounded stretch i.e., distances are approximately preserved. With high probability, for each deleted edge there is a path of $O(\log n \cdot (\Delta + \log n)/\epsilon)$ hops in the output that connects its endpoints.

2012 ACM Subject Classification Theory of computation → Sparsification and spanners

Keywords and phrases local, spanning graph, sparse

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.87

Related Version `https://arxiv.org/abs/1703.05418`

Acknowledgements We thank the anonymous reviewers for their suggestions and comments.

1 Introduction

When operating on very large graphs, it is often impractical or infeasible to (i) hold the entire graph in the local memory of a processing unit, (ii) run linear-time (or even slower) algorithms, or even (iii) have only a single processing unit perform computations sequentially. These constraints inspired the Centralized Local model [17], which essentially views the input as being stored in a (likely distributed) database that provides query access to external processing units. To minimize the coordination overhead of such a system, it is furthermore required that there is no shared memory or communication between the querying processes, except for shared randomness provided alongside the access to the input. The idea is now to

[1] Supported by ERC-CoG grant 772839.
[2] $\tilde{O}$-notation hides polylogarithmic factors in n.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 87; pp. 87:1–87:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

run sublinear-time algorithms that extract useful global properties of the graph and/or to examine the input graph locally upon demand by applications.

Studying graphs in this model, lead to the emergence of algorithms that provide query access to a variety of graph-theoretical structures like, e.g., independent or dominating sets. In such a case, it is crucial that locally evaluating whether a node participates in such a set is *consistent* with the same evaluation for other nodes. This is a non-trivial task, as local decisions can only be coordinated implicitly via the structure of the input (which is to be examined as little as possible) and the shared randomness. Nonetheless, this budding field brought forth a number of elegant algorithms solving, e.g., maximal independent set, hypergraph coloring, k-CNF, approximate maximum matching and approximate minimum vertex cover for bipartite graphs [1, 4, 5, 11, 12, 13, 17].

In this work, we consider another very basic graph structure: sparse spanning subgraphs. Here, the task is to select a subset of the edges of the (connected) input graph so that the output is still connected, but has only few edges. By "few" we mean that, for some input parameter $\varepsilon > 0$, the number of selected edges is at most $(1 + \varepsilon)n$, where n denotes the number of nodes. One may see this as a relaxed version of the problem of outputting a spanning tree of the graph, which is a too rigid requirement when looking for fast algorithms: on a cycle, a single edge has to be deleted, but this necessitates to first verify that the input graph is not, in fact, a line.

▶ **Definition 1** ([9]). An algorithm $\mathcal{A}$ is a *Local Sparse Spanning Graph (LSSG) algorithm* if, given $n, \Delta \geq 1$, a parameter $\epsilon \geq 0$, and query access to the incidence list representation of a connected graph $G = (V, E)$ over n vertices and of degree at most Δ, it provides oracle access to a subgraph $G' = (V, E')$ of G such that:

1. G' is connected.
2. $|E'| \leq (1 + \epsilon) \cdot n$ with high probability (w.h.p.),[3] where E' is determined by G and the internal randomness of $\mathcal{A}$.

By "providing oracle access to G'" we mean that on input $\{u, v\} \in E$, $\mathcal{A}$ returns whether $\{u, v\} \in E'$, and for any sequence of edges, $\mathcal{A}$ answers consistently with respect to the same G'.

We are interested in LSSG algorithms that, for each given edge, perform as few queries as possible to G. Observe that Item 2 implies that the answers of an LLSG algorithm to queries cannot depend on previously asked queries.

We note that relaxing from requiring a tree as output makes it possible to ask for additional guarantees that, in general, cannot be met by a spanning tree. Instead of merely preserving connectivity, it becomes possible to maintain *distances* up to small factors (for unweighted graphs). Such subgraphs are known as (sparse, multiplicative) *spanners* [14, 15]. In fact, choosing $\varepsilon \in o(1)$ then yields *ultra-sparse* spanners that are $o(n)$ edges away from being trees.

1.1 Our Contribution

We give the first non-trivial LSSG algorithm in the Centralized Local model that runs on arbitrary graphs. We achieve a query complexity of $\tilde{O}(\text{poly}(\Delta/\varepsilon)n^{2/3})$ per edge, w.h.p. Moreover, we guarantee that for each edge that is not selected into the spanner, w.h.p. there

[3] That is, with probability at least $1 - 1/n^c$ for an arbitrary constant $c > 0$ that is chosen upfront.

is a path of $O(\log n \cdot (\Delta + \log n)/\epsilon)$ hops consisting of edges that are selected into the spanner; this is referred to as a *stretch* of $O(\log n \cdot (\Delta + \log n)/\epsilon)$.

For simplicity, assume for the moment that Δ and ε are constants. Our algorithm combines the following key ideas.

- We classify edges as expanding if there are sufficiently many (roughly $n^{1/3}$) nodes within $O(\log n)$ hops of their endpoints. For non-expanding edges, we can efficiently simulate a standard distributed spanner algorithm at small query complexity, as solutions of running time $O(\log n)$ are known (e.g. [3]).
- On the node set induced by expanding edges, we can construct a partition into Voronoi cells with respect to roughly $n^{2/3}$ randomly sampled centers. The Voronoi cells are spanned by trees of depth $O(\log n)$, as expanding nodes have their closest center within $O(\log n)$ hops w.h.p. Finding the closest center has query complexity $\tilde{O}(n^{1/3})$.
- We refine the partition into Voronoi cells further into *clusters* of $\tilde{O}(n^{1/3})$ nodes. We simply let a node be a singleton cluster if its subtree in the spanning tree of its cell contains more than $\tilde{O}(n^{1/3})$ nodes. This construction has query complexity $\tilde{O}(n^{2/3})$ for constructing a complete cluster, yet ensures that there are $\tilde{O}(n^{2/3})$ clusters in total due to the low depth of the trees that span the Voronoi cells; moreover, each cluster is completely contained in some Voronoi cell.
- It remains to select few edges to interconnect the Voronoi cells. This is the main challenge, for which the above properties of the partition are crucial. To keep the number of selected edges small in expectation, we mark a subset of expected size $\tilde{\Theta}(n^{1/3})$ of the clusters by marking each Voronoi cell (and thereby its constituent clusters) with probability $n^{-1/3}$. We then try to ensure that (i) clusters select an edge to each adjacent marked Voronoi cell and (ii) for each marked Voronoi cell adjacent to an adjacent cluster, they select one edge connecting to *some* cluster adjacent to this cell.
- The main issue with the previous step is that we cannot afford to construct each adjacent cluster, preventing us from guaranteeing (ii). We circumvent this obstacle by identifying for adjacent clusters in which cell they are and keeping an edge for the purpose of (ii) if it satisfies a certain minimality requirement with respect to the *rank* of the cell used for symmetry breaking purposes. This way, we avoid construction of adjacent clusters, instead needing to determine the rank of their Voronoi cells only. This way, we maintain query complexity $\tilde{O}(n^{2/3})$.
- However, this now entails an inductive argument for ensuring connectivity, which also affects stretch. By choosing Voronoi cell ranks uniformly at random, we ensure that the length of such an inductive chain is bounded by $O(\log n)$ w.h.p. Together with the depth of the Voronoi cell trees of $O(\log n)$ and the stretch of the spanner algorithm for non-expanding edges (also $O(\log n)$), this yields the total bound of $O(\log^2 n)$ on the stretch of our scheme.

Finally, we note that we can place the above routine in a wrapper verifying that, w.h.p., the number of globally selected edges does not significantly exceed the expectation. If this is not the case, the wrapper starts the process all over. Since in each attempt the success probability is constant (and the verifier succeeds w.h.p.), we get within $O(\log n)$ attempts that the bound on the number of selected edges is satisfied w.h.p. and the routine terminates.

Relation to Property Testing. As observed in [10], testing cycle-freeness with one sided-error in the bounded-degree model can be reduced to the LSSG problem. From this reduction it follows that we obtain a tester for cycle-freeness that works in $\tilde{O}(n^{2/3})$ time. Czumaj et al. [2] studied the problem of C_k-minor freeness with one sided-error. For cycle-freeness

(C_3-minor freeness) their complexity is $\tilde{O}(\sqrt{n})$, therefore their complexity is better than ours. However, we would like to point out that an LSSG algorithm gives a stronger guarantee than a one-sided error tester for cycle-freeness. An LSSG algorithm can be used to find for all but at most $(1+\epsilon)|V|$ edges e, a cycle that e belongs to, i.e., a witness for the violation by e can be provided. In contrast, a one-sided error tester merely guarantees to find a single cycle in instances that are ϵ-far from being cycle-free.

Perhaps more importantly, our approach proves useful when testing for other minors. Recently, Fichtenberger et al. [6] built on our work to construct one-sided error tester for outerplanarity and other properties that are characterized by a set of forbidden minors.

1.2 Related work

The problem of finding a sparse spanning subgraph in the Centralized Local model was first studied in [9], where the authors show a lower bound of $\Omega(\sqrt{n})$ queries for constant ϵ and Δ (see also survey by Rubinfeld [18]). They also present an upper bound with nearly tight query complexity for graphs that have very good expansion properties. However, for general (bounded degree) graphs their algorithm might query the entire graph for completing a single call to the oracle. They also provide an efficient algorithm for minor-free graphs that was later improved in [8]. The algorithm presented in [8] achieves a query complexity that is polynomial in Δ and $1/\epsilon$ and is independent of n. The stretch factor of this algorithm is also independent of n and depends only on Δ, $1/\epsilon$, and the size of the excluded minor.

A characterization of the query complexity of the problem was presented in [7]. Specifically, [7] provide an upper bound (which builds on an algorithm in [9]) that has a query complexity that is independent of n (however, super-exponential in $1/\epsilon$) for families of graphs which are, roughly speaking, sufficiently non-expanding everywhere. On the other hand, they show that, for a family of graphs with expansion properties that are slightly better, any local algorithm must have a query complexity that depends on n.

In the (distributed) Local model, Ram and Vicari [16] study the same problem and provide an algorithm that runs in $\min\{D, O(\log n)\}$ rounds, where D is diameter of the input graph. Their algorithm achieves the sparsity property by breaking all short cycles.

2 Preliminaries

The graphs we consider are undirected and have a known degree bound Δ, and we assume we have query access to their incidence list representation. Namely, for any vertex v and index $1 \leq i \leq \Delta$, it is possible to obtain the i^{th} neighbor of v by performing a query to the graph (if v has less than i neighbors, then a special symbol is returned). Without loss of generality, we assume that graphs are simple, i.e., contain neither loops nor parallel edges.[4] The number of vertices in the graph is n and we assume that each vertex v has a unique id, which for simplicity we also denote by v. There is a total order on the ids, i.e., given any two distinct ids u and v, we can decide whether $u < v$ or $v < u$.

Let $G = (V, E)$ be a graph, where $V = [n]$. We denote the distance between two vertices u and v in G by $d_G(u, v)$. For vertex $v \in V$ and an integer r, let $\Gamma_r(v, G)$ denote the set of vertices at distance at most r from v. When the graph G is clear from the context, we shall use the shorthands $d(u, v)$ and $\Gamma_r(v)$ for $d_G(u, v)$ and $\Gamma_r(v, G)$, respectively.

[4] The answer on a self-loop can always be negative, and we can default to rejecting all but the first edge between two nodes.

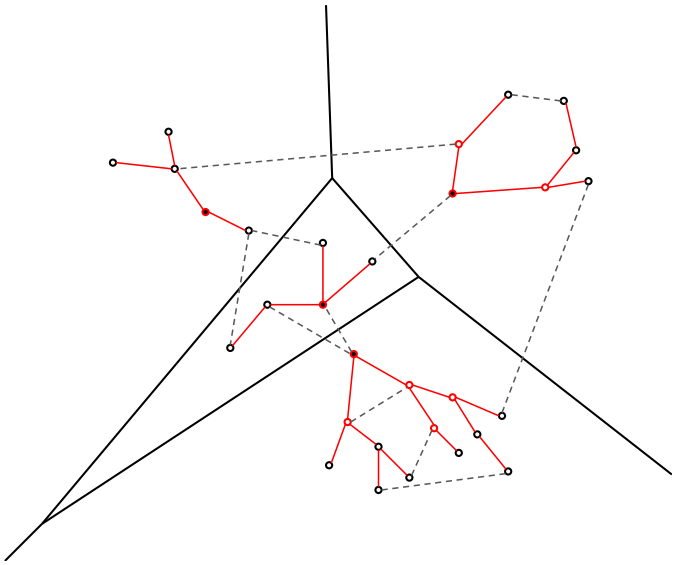


Figure 1 Partition of a graph into cells and clusters, where $k = 6$. Black lines are the borders of Voronoi cells, whose centers have black fillings. Red edges belong to the BFS trees spanning the clusters, while dashed gray lines are non-tree edges. Red circles indicate singleton clusters (if the node has a red child) or the roots of subtrees that form a cluster (if the children are black).

The total order on the vertices induces a total order r on the edges of the graph in the following straightforward manner: $r(\{u,v\}) < r(\{u',v'\})$ if and only if $\min\{u,v\} < \min\{u',v'\}$ or $\min\{u,v\} = \min\{u',v'\}$ and $\max\{u,v\} < \max\{u',v'\}$. The total order over the vertices also induces an order over those vertices visited by a Breadth First Search (BFS) starting from any given vertex v, and whenever we refer to a BFS, we mean that it is performed according to this order.

Whenever referring to one of the above orders, we may refer to the *rank* of an element in the respective order. This is simply the index of the respective element when listing all elements ascendingly with respect to the order.

For a graph $G = (V,E)$ and a pair of disjoint subsets of vertices $A \subset V$ and $B \subset V$ let $E_G(A,B) \stackrel{\text{def}}{=} \{(u,v) \in E \,|\, u \in A \wedge v \in B\}$. When it is clear from the context, we omit the subscript. We say that a pair of subsets of vertices A and B are *adjacent* if $E_G(A,B) \neq \emptyset$.

3 An Algorithm that Works under a Promise

We begin by describing an LSSG algorithm which works under the following promise on the input graph $G = (V,E)$. Sample ℓ uniformly at random from $[b \log n / \log(1+\epsilon), b \log n / \log(1+\epsilon) + \Delta/\epsilon\}]$, and let $k \stackrel{\text{def}}{=} cn^{1/3} \ln n \cdot \ell\Delta/\epsilon$, where c and b are sufficiently large constants. For every $v \in V$, let $i_v \stackrel{\text{def}}{=} \min_r\{|\Gamma_r(v)| \geq k\}$. We are promised that $\max_{v \in V}\{i_v\} \leq \ell$. In words, we assume that the ℓ-hop neighborhood of every vertex in G contains at least k vertices. First, we fix a simple partition of V.

3.1 The Underlying Partition

Centers. Pick a set $S \subset |V|$ of $r \stackrel{\text{def}}{=} \Theta(\epsilon n^{2/3} / \ln n)$ vertices at random. We shall refer to the vertices in S as *centers*. For each vertex $v \in V$, its *center*, denoted by $c(v)$, is the center which is closest to v amongst all centers (break ties between centers according to the id of the center).

Voronoi cells. The *Voronoi cell* of a vertex v, denoted by $\mathrm{Vor}(v)$, is the set of all vertices u for which $c(u) = c(v)$. Additionally, we assign to each cell a random rank, so that there is a uniformly random total order on the cells; note carefully that the rank of a cell thus differs from the rank of its center (which is given by its identifier, which is not assigned randomly). We remark that we can determine the rank of the cell from the shared randomness and the cell's identifier, for which we simply use the identifier of its center.

Clusters. For each Voronoi cell, consider the BFS tree spanning it, which is rooted at the respective center. For every $v \in V$, let $p(v)$ denote the *parent* of v in this BFS tree. If v is a center then $p(v) = v$. For every $v \in V \setminus S$, let $T(v)$ denote the subtree of v in the above-mentioned BFS tree when we remove the edge $\{v, p(v)\}$; for $v \in S$, $T(v)$ is simply the entire tree. Now consider a Voronoi cell. If the cell contains at most k vertices, then the *cluster* of all the vertices in the Voronoi cell is the cell itself. Otherwise, there are two cases. If $T(v)$ contains at least k vertices, then the cluster of v is the singleton $\{v\}$. Otherwise, v has a unique ancestor u (including v) for which $T(u)$ contains less than k vertices and $T(p(u))$ contains at least k vertices. The cluster of v is the set of vertices in $T(u)$. For a cluster C, let $c(C)$ denote the center of the vertices in C (all the vertices in the same cluster have the same center as they all belong to the same Voronoi cell). Let $\mathrm{Vor}(C)$ denote the Voronoi cell of the vertices in C.

This describes a partition of V into Voronoi cells, and a refinement of this partition into clusters. See Figure 1 for an illustration.

3.2 The Edge Set

Our spanner, $H = (V, E')$, initially contains, for each Voronoi cell Vor the edges of the BFS tree that spans Vor, i.e., the BFS tree rooted at the center of Vor spanning the subgraph induced by Vor (see Section 2 for more details). Clearly, these edges also span the clusters. Next, we describe which edges we add to E' in order to connect the clusters.

Marked Clusters and Clusters-of-Clusters

Each center in S is *marked* independently with probability $p \stackrel{\text{def}}{=} 1/n^{1/3}$. If a center is marked, then we say that its Voronoi cell is marked and all the clusters in this cell are marked as well.

Cluster-of-clusters. For every marked cluster, C, define the *cluster-of-clusters* of C, denoted by $\mathcal{C}(C)$, to be the set of clusters which consists of C and all the clusters which are adjacent to C. A cluster B is *participating* in $\mathcal{C}(C)$ if $B \in \mathcal{C}(C)$ and the edge of minimum rank in $E(B, \mathrm{Vor}(C))$ also belongs to $E(B, C)$. Thus, if B is adjacent to $\mathrm{Vor}(C)$ and $\mathrm{Vor}(C)$ is marked, then there is a unique cluster $D \subseteq \mathrm{Vor}(C)$ such that B participates in $\mathcal{C}(D)$. See Figure 2 for a visualization.

The Edges between Clusters

By saying that we *connect* two adjacent subsets of vertices A and B, we mean that we add the minimum ranked edge in $E(A, B)$ to E'. For a cluster A, define its *adjacent centers* $\mathrm{Vor}(\partial A) \stackrel{\text{def}}{=} \{\mathrm{Vor}(v) \mid \exists u \in A \text{ such that } \{u, v\} \in E\} \setminus \{\mathrm{Vor}(A)\}$, i.e., the set of Voronoi cells that are adjacent to A. This definition explicitly excludes $\mathrm{Vor}(A)$, as there is no need to connect A to its own Voronoi cell.

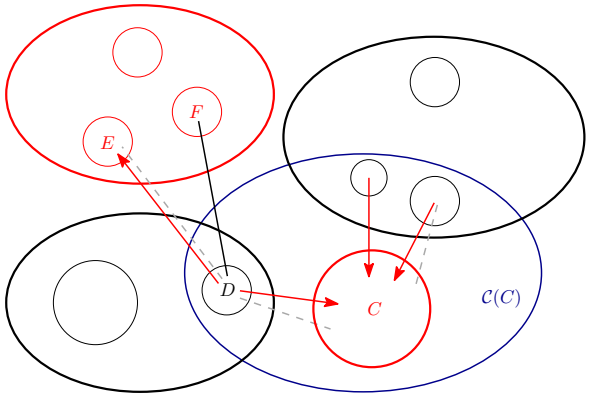


Figure 2 Illustration of marked clusters and clusters of clusters. Thick red and black ovals are marked and unmarked cells, respectively. Thin circles are clusters, where cluster C comprises its entire cell. Thick edges are the ones of minimum rank between their incident clusters, while the dotted edges do not meet this criterion. The arrows of red edges indicate participation in the respective adjacent marked cluster; note that D does not participate in $\mathcal{C}(F)$, as for each adjacent marked cell Vor it exclusively participates in the cluster-of-clusters connected to it by the edge of minimum rank in $E(D, \mathrm{Vor})$. $\mathcal{C}(C)$ is marked in blue; all its constituent clusters also participate in $\mathcal{C}(C)$, as $\mathrm{Vor}(C) = C$.

We next describe how we connect the clusters. The high-level idea is to make sure that every marked cluster and the clusters that participate in the respective cluster-of-clusters remains connected. This implies that the cluster-of-clusters remain connected as well, as every Voronoi cell is connected. For clusters which are not adjacent to any marked cluster we make sure to keep them connected to all adjacent Voronoi cells. Formally:

1. We connect every cluster to every adjacent marked cluster.
2. Each cluster A that is not participating in any cluster-of-clusters (i.e., no cell adjacent to A is marked) we connect to each adjacent cell.
3. Suppose cluster A is adjacent to cluster B, where B is adjacent to a marked cell Vor. Denote by C the (unique) cluster in cell Vor for which B participates in $\mathcal{C}(C)$. We connect A with B if the following conditions hold:
   - $\mathrm{Vor}(B)$ has minimum rank amongst $\mathrm{Vor}(\partial A) \cap \mathrm{Vor}(\partial C)$
   - the minimum ranked edge in $E(A, \mathrm{Vor}(B))$ is also in $E(A, B)$

Figure 3 showcases the third rule. Roughly speaking, the idea is that we want to connect cluster A with $\mathcal{C}(C)$ to preserve connectivity, however, at the same time, we want to make sure that we do not add too many edges to our spanning subgraph. Therefore, we connect A and B only if we do not see an evidence that A and $\mathcal{C}(C)$ remain connected through a different path.

3.3 Sparsity

▶ **Lemma 1.** *The number of clusters, denoted by s, is at most $|S| + n\ell(\Delta + 1)/k$.*

Proof. We first observe that, due to the promise on G, it follows that for every $v \in V$, $d(v, c(v)) \leq \ell$ w.h.p. Recall the terminology from Subsection 3.1. Consider v for which $|T(v)| \geq k$ and therefore its cluster is the singleton $\{v\}$. We say that a vertex u is *special* if $|T(u)| \geq k$ and for every child of u in $T(u)$, t, it holds that $|T(t)| < k$. By an inductive argument, it follows that v is an ancestor of a special vertex. Since for every pair of special vertices u and w, $T(u)$ and $T(w)$ are vertex disjoint, we obtain that there are at most n/k

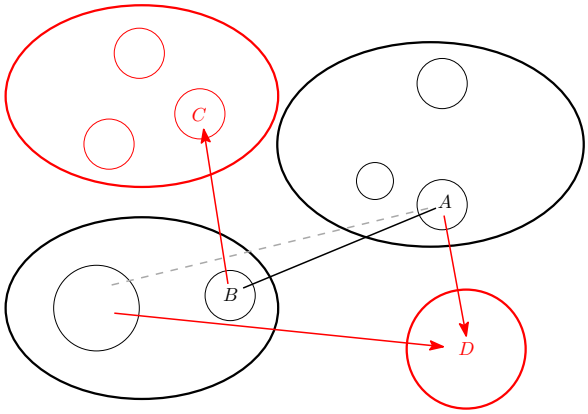


Figure 3 Illustration of the third edge selection rule. In this example, the thick black edge has minimum rank in $E(A, \mathrm{Vor}(B))$ and B has minimum rank in $\partial A \cap \partial C$. However, the rank of $\mathrm{Vor}(D)$ is smaller than that of $\mathrm{Vor}(B)$, and hence the dashed edge is not selected. Here, A will select the direct edge connecting it to D, due to the first rule.

special vertices. Since for every special vertex, there are at most ℓ ancestors, the total number of vertices v with $|T(v)| \geq k$ is bounded by $n\ell/k$.

Observe that any cluster either (i) is a singleton $\{v\}$ with $|T(v)| \geq k$, (ii) contains a node v such that $|T(p(v))| \geq k$, or (iii) is an entire Voronoi cell. We just bounded the number of clusters of type (i) by $n\ell/k$, and immediately get a bound on the number of type (ii) clusters of $n\ell\Delta/k$. The number of type (iii) clusters is bounded by the number of Voronoi cells $|S|$, showing the desired bound on s. ◀

▶ **Lemma 2.** $\mathrm{Exp}(|E'|) \leq (1 + O(\epsilon))|V|$.

Proof. The number of edges which are taken due to Condition 1 is at most s times the number of marked clusters. In expectation, there are $s \cdot p$ marked clusters, yielding at most $s^2 p$ edges in expectation. Since $s = O(\epsilon n^{2/3}/\ln n)$ and $p = 1/n^{1/3}$ we obtain that $s^2 p = O(\epsilon n/\ln n)$.

Let A be a cluster. The number of edges which are adjacent to A and are taken due to Condition 3 is bounded by the total number of clusters-of-clusters. The number of clusters-of-clusters is exactly the number of marked clusters. Thus, the total number of edges which are taken due to Condition 3 is bounded by $s^2 p$.

Observe that the probability that cluster A is not adjacent to a marked cell is $(1-p)^{|\mathrm{Vor}(\partial A)|} \leq e^{-p|\mathrm{Vor}(\partial A)|}$. Hence, if $|\mathrm{Vor}(\partial A)| \geq 3p^{-1}\ln n$, A is adjacent to a marked cell w.h.p. Using a union bound over all clusters, it follows that w.h.p. each cluster A without an adjacent marked cell satisfies that $|\mathrm{Vor}(\partial A)| \leq 3p^{-1}\ln n$ with probability at least $1 - 1/n^2$; the probability at most $1/n^2$ event that this bound is violated cannot contribute more than $|E|/n^2 < 1$ to the expectation. Therefore, the total number of edges which are taken due to Condition 2 is bounded by $(3s\ln n)/p + 1 = O(\epsilon n)$.

Since the number of edges we add due to the BFS trees of the Voronoi cells is at most $|V| - 1$, we conclude that $\mathrm{Exp}(|E|') \leq (1 + O(\epsilon))|V|$, as desired. ◀

3.4 Connectivity and Stretch

▶ **Lemma 3.** *H is connected.*

Proof. Recall that H contains a spanning tree on every Voronoi cell, hence it suffices to show that we can connect any pair of Voronoi cells by a path between some of their vertices. Moreover, the facts that G is connected and the Voronoi cells are a partition of V imply that it is sufficient to prove this for any pair of adjacent Voronoi cells. Accordingly, let Vor and Vor_1 be two cells such that $E(\mathrm{Vor}, \mathrm{Vor}_1) \neq \emptyset$.

Consider clusters $A \subseteq \mathrm{Vor}$ and $B \subseteq \mathrm{Vor}_1$ such that the edge e of minimum rank in $E(\mathrm{Vor}, \mathrm{Vor}_1)$ is in $E(A, B)$. If B is not adjacent to a marked cell, then Condition 2 implies that e is selected into H. Thus, we may assume that B is adjacent to a marked cell Vor'. Accordingly, there exists a marked cluster $C \subseteq \mathrm{Vor}'$ such that B is participating in $\mathcal{C}(C)$.

If the rank of Vor_1 is minimum in $\mathrm{Vor}(\partial C) \cap \mathrm{Vor}(\partial A)$, then e is selected into H by Condition 3 and we are done. Otherwise, observe that Vor_1 is connected to Vor', as the edge of minimum rank in $E(B, C)$ is selected into H by Condition 1. Therefore, it suffices to show that Vor gets connected to Vor'. Let Vor_2 be the cell of minimum rank among $\mathrm{Vor}(\partial C) \cap \mathrm{Vor}(\partial A)$. Let $D \subseteq \mathrm{Vor}_2$ be the cluster satisfying that the edge e' of minimum rank in $E(A, \mathrm{Vor}_2)$ is in $E(A, D)$. Note that Vor_2 is connected to Vor' (which we saw to be connected to Vor_1), as there is some cluster $D' \subseteq \mathrm{Vor}(D)$ that is adjacent to C and selects the edge of minimum rank in $E(D', C)$ by Condition 1.

Overall, we see that it is sufficient to show that Vor gets connected to Vor_2, where Vor_2 has smaller rank than Vor_1. We now repeat the above reasoning inductively. In step i, we either succeed in establishing connectivity between Vor and Vor_i, or we determine a cell Vor_{i+1} that has smaller rank than Vor_i and is connected to Vor_i. As any sequence of Voronoi cells of descending ranks must be finite, the induction halts after finitely many steps. Because the induction invariant is that Vor_{i+1} is connected to Vor_i, this establishes connectivity between Vor and Vor_1, completing the proof. ◀

▶ **Lemma 4.** *Denote by G_{Vor} the graph obtained from G by contracting Voronoi cells and by H_{Vor} its subgraph obtained when doing the same in H. If the cells' ranks are uniformly random, w.h.p. H_{Vor} is a spanner of G_{Vor} of stretch $O(\log n)$.*

Proof. Recall the proof of Lemma 3. We established connectivity by an inductive argument, where each step increased the number of traversed Voronoi cells by two. Hence, it suffices to show that the induction halts after $O(\log n)$ steps w.h.p.

To see this, observe first that G_{Vor} is independent of the ranks assigned to Voronoi cells and pick any pair of adjacent cells Vor, Vor_1, i.e., neighbors in G_{Vor}. We perform the induction again, assigning ranks from high to low only as needed in each step, according to the following process. In each step, we query the rank of some cells, and given an answer of rank r, the ranks of all cells of rank at least r are revealed as well. In step i, we begin by querying the rank of Vor_i. Consider the cluster $D_i \subseteq \mathrm{Vor}_i$ adjacent to A satisfying that the edge with minimum rank in $E(\mathrm{Vor}_i, A)$ is also in $E(D_i, A)$. We can assume without loss of generality that D_i is adjacent to a marked cluster F_i and that it is participating in $\mathcal{C}(F_i)$ (as otherwise D_i connects to A directly and we can terminate the process). If the ranks of all the cells adjacent to both F_i and A were already revealed, then the process terminates. Otherwise, we query the rank of all such cells whose rank is still unrevealed. We set the cell of the queried cluster that has minimum rank to be Vor_{i+1} and we continue to the next step.

We claim that, in each step i, either the process terminates, or the rank of Vor_{i+1} is at most half of the rank of Vor_i with probability at least $1/2$. To verify this, observe that in the beginning of step i, any cell center whose rank was not revealed so far has rank which

is uniformly distributed in $[r_i - 1]$, where r_i is the rank of Vor_i.[5] With probability at least $1/2$, such a rank is at most $r_i/2$. If Vor_i has no adjacent cells whose ranks have not been revealed yet, the process terminates. Hence, regardless of whether the process terminates or not, the claim holds.

By Chernoff's bound, we conclude that the process terminates within $O(\log n)$ steps w.h.p., as r_1 is bounded by the number of Voronoi cells, which itself is trivially bounded by n. By the union bound over all pairs of cells Vor and Vor_1, we get the desired guarantee. ◀

▶ **Corollary 1.** *W.h.p., H is a spanner of G of stretch $O(\log n \cdot (\Delta + \log n)/\epsilon)$.*

Proof. Due to the promise on G, w.h.p. the spanning trees on Voronoi cells have depth $\ell \in O((\Delta + \log n)/\epsilon)$. Hence, for any edge within a Voronoi cell, the claim holds w.h.p. Moreover, for an edge connecting different Voronoi cells, by Lemma 4, w.h.p. there is a path of length $O(\log n)$ in H_{Vor} connecting the respective cells. Navigating with at most $2\ell \in O((\Delta + \log n)/\epsilon)$ hops in each traversed cell, we obtain a suitable path of length $O(\log n \cdot (\Delta + \log n)/\epsilon)$ in H. ◀

4 The Algorithm for General Graphs

We use a combination of the algorithm in Section 3 with the algorithm by Elkin and Neiman for ultra-sparse spanners [3]. We call a vertex v *remote*, with respect to a set of centers, if the ℓ-hop neighborhood of v does not include a center. Fix S, let R denote the set of remote vertices with respect to S, and abbreviate $\bar{R} \stackrel{\text{def}}{=} V \setminus R$.

First Step. Run the algorithm from Section 3 on the subgraph induced by $\bar{R}$, i.e., $\{u, v\} \in E$ with $u, v \in \bar{R}$ is added to E' if and only if the algorithm outputs the edge.

Second Step. Run the algorithm of Elkin and Neiman [3] on the subgraph induced by R, i.e., $\{u, v\} \in E$ with $u, v \in R$ is added to E' if and only if the algorithm outputs the edge.[6] Their algorithm proceeds as follows. Given an integer h, each vertex v draws r_v according to the exponential distribution with parameter $\beta = \ln(n/\delta)/h$, where δ is a parameter that controls the success probability of the algorithm. Each vertex v receives r_u from every vertex u within distance h, and stores $m_u(v) = r_u - d(u, v)$ and a neighbor on a shortest path between v and u, denoted by $n_u(v)$. The edges that are added to the spanner are $C(v) = \{\{v, n_u(v)\} \mid m_u(v) \geq \max_{w \in V}\{m_w(v) - 1\}\}$, for every $v \in R$.

▶ **Corollary 2.** *M[Claim 2.3 in [3]] With probability at least $1 - \delta$, $r_v < h$ for all $v \in V$.*

M We choose $\delta = 1/n^{b-1}$, where $b \geq 2$ is a constant (see begining of Section 3), and $h = \ell$. The following lemma implies that the total number of edges that we add to H in the second step is at most $|R| \cdot (n^b)^{1/\ell} \leq |R|(1 + \epsilon)$ in expectation.

▶ **Lemma 5** (Proof of Lemma 2.2 in [3]). *For every $v \in R$, $\mathrm{Exp}[C(v)] \leq (n/\delta)^{1/h}$.*

[5] In step 1, we first query Vor_1 and then observe that this statement holds.
[6] The algorithm is described for connected graphs; we simply apply it to each connected component of R.

Third Step. Add to E' all edges $e \in E(R, \bar{R})$.

The following lemma implies that the expected number of edges which are added in the third step is at most ϵn.

▶ **Lemma 6.** $\mathrm{Exp}[|E(R, \bar{R})|] \leq \epsilon n$.

Proof. Observe that for an edge $\{u, v\} \in E$, there is at most one integer r such that $\Gamma_r(u) \cap S = \emptyset$ and $\Gamma_r(v) \cap S \neq \emptyset$ (or vice versa). If there is no such r or $\ell \neq r$, then the edge is not in $E(R, \bar{R})$. Over the random choice of ℓ, the probability of the event that the edge is included is at most $\Pr[\ell = r] \leq \epsilon/\Delta$. The lemma follows by linearity of expectation. ◀

4.1 Stretch Factor

Consider any edge $e = \{u, v\} \in E \setminus E'$ we removed. If $u, v \in \bar{R}$, e was removed by the Algorithm from Section 3, which was applied to the subgraph induced by $\bar{R}$. Applying Corollary 1 to the connected component of e, we get that w.h.p. there is a path of length $O(\log n \cdot (\Delta + \log n)/\epsilon)$ from u to v in H. If $u, v \in R$, by Claim 2 and the choice of parameters, w.h.p. there is a path of length $O((\Delta + \log n)/\epsilon)$ from u to v in H. As $e \notin E(R, \bar{R})$ by the third step, we arrive at the following corollary.

▶ **Corollary 3.** *The above algorithm guarantees stretch $O(\log n \cdot (\Delta + \log n)/\epsilon)$ w.h.p. and satisfies that $\mathrm{Exp}[|E'|] \in (1 + O(\epsilon))n$.*

5 The Local Algorithm

In this section we prove our main theorem.

▶ **Theorem 2.** *Algorithm 1 is an LSSG algorithm. For any graph G over n vertices of maximum degree at most Δ and $\epsilon > 0$, its query complexity, space complexity (length of the random seed), and running time are $\tilde{O}(n^{2/3}) \cdot \mathrm{poly}(1/\epsilon, \Delta)$.*

Proof. The correctness of the algorithm follows from the previous sections. We shall prove that its complexity is as claimed. We analyze the complexity in terms of n. There are additional factors that depend polynomially in Δ and $1/\epsilon$. The following claims hold w.h.p., simultaneously, for all vertices.

Recall that a vertex u is remote if $\Gamma_\ell(u)$ does not contain a center. Due to the sampling probability for centers, a center is found after exploring $\tilde{O}(n^{1/3})$ vertices. Therefore, we can decide for any vertex u whether it is in R with query and time complexity $\tilde{O}(n^{1/3})$. Moreover, if $u \in \bar{R}$, without additional cost the respective subroutine can return $c(u)$, the center of u, and $d(u, c(u))$ (as we explore in a BFS fashion).

For Step 1, we need to determine $\Gamma_\ell(u) \cap R$. To do this, it suffices to explore $\Gamma_\ell(u)$, and for each vertex in it, to determine whether it is in R or not. Since $|\Gamma_\ell(u)| = \tilde{O}(n^{1/3})$ for every vertex $u \in R$, we obtain that the query and time complexity of this step is $\tilde{O}(n^{2/3})$, in total. Accordingly, Step 2 has query and time complexity $\tilde{O}(n^{1/3})$.

If $u, v \in \bar{R}$, the algorithm proceeds as in Section 3. We show first that we can reconstruct clusters efficiently. W.l.o.g., consider u. We determine $c(u)$ and $d(u, c(u))$. For each neighbor w of u, we determine whether it is in $\bar{R}$ (if not, the node is discarded), its center $c(w)$, and the distance $d(w, c(w))$. As $u \in \bar{R}$, and assuming that $u \neq c(u)$, it must have at least one neighbor w in distance $d(u, c(u)) - 1$ of $c(u)$; any such w satisfies that $c(w) = c(u)$, as otherwise $d(u, c(w)) = d(w, c(w))$ and $c(w) < c(u)$, a contradiction to the tie-breaking rule for centers. Among these candidates w, we know that the one with minimum rank is the

Algorithm 1 LSSG for general graphs.

Input: $\{u, v\} \in E$

Output: whether $\{u, v\}$ is in E' or not.

1. If $u, v \in R$, compute the output of algorithm of Elkin and Neiman at u and v when running it on the connected component of u and v in the subgraph induced by R. Return **true** if $\{u, v\} \in C(u) \cup C(v)$ and **false** otherwise.
2. If $\{u, v\} \in E(R, \bar{R})$, return **true**.
3. Otherwise, $u, v \in \bar{R}$ and we proceed according to Section 3, where all nodes in R are ignored:
   a. If $\mathrm{Vor}(u) = \mathrm{Vor}(v)$, return **true** if $\{u, v\}$ is in the BFS tree of $\mathrm{Vor}(u)$ and **false** otherwise.
   b. Otherwise, let Q and W denote the clusters of u and v, respectively. Return **true** if at least one of the following conditions hold for $A = Q$ and $B = W$, or symmetrically, for $A = W$ and $B = Q$, and **false** otherwise.
      i. A is a marked cluster and $\{u, v\}$ has minimum rank amongst the edges in $E(A, B)$.
      ii. A is not participating in any cluster-of-clusters. Namely, all the clusters which are adjacent to A are not marked. In this case, we take $\{u, v\}$ if it has minimum rank amongst the edges in $E(A, \mathrm{Vor}(B))$.
      iii. There exists a marked cluster C such that A is participating in $\mathcal{C}(C)$, and the following holds:
         - $\mathrm{Vor}(A)$ has minimum rank amongst $\mathrm{Vor}(\partial B) \cap \mathrm{Vor}(\partial C)$
         - $\{u, v\}$ has minimum rank amongst the edges in $E(B, \mathrm{Vor}(A))$.

parent of u in the BFS tree of $\mathrm{Vor}(u)$ rooted at $c(u)$, due to the tie-breaking rule for the BFS construction. We can use this subroutine to partially explore the BFS of $\mathrm{Vor}(u)$: given any node $w \in \bar{R}$, we can determine its parent, $p(w)$, and children in $\mathrm{Vor}(w)$ at query and time complexity $\tilde{O}(n^{1/3})$ as follows. The shortest path that has the smallest lexicographical order from $c(w)$ to w is the path connecting these vertices in the BFS tree. Therefore, the parent of w in this tree can be found by performing a BFS from w until the first level in which $c(w)$ is reached. To determine the children we simply run this subroutine for each neighbor, y, of w and find out whether $w = p(y)$ or not. In order to determine whether $T(w) < k$ or $T(w) \geq k$ (by partially or completely exploring $T(w)$), we need to run this subroutine $O(k)$ times, therefore this can be obtained with query complexity $\tilde{O}(n^{2/3})$. if $T(w) < k$, we determine $T(w)$ completely. We collect this information for u and its $\ell - 1$ ancestors, and determine the cluster Q of u. Finally, we repeat the procedure to reconstruct the cluster W of v, and determine for all nodes adjacent to either cluster whether they are in $\bar{R}$ and, if so, their centers. The query and time complexity of this operation is $\tilde{O}(n^{2/3})$ in total.

For the cases $A = Q$, $B = W$ and $A = W$, $B = Q$, respectively, with the above information we can determine

- whether $\mathrm{Vor}(u) = \mathrm{Vor}(v)$,
- if $\mathrm{Vor}(u) = \mathrm{Vor}(v)$, whether $p(u) = v$ or $p(v) = u$,
- if $\mathrm{Vor}(u) \neq \mathrm{Vor}(v)$,
  - whether A is marked (i.e., $c(A)$ has been marked) and whether $\{u, v\}$ has minimum rank in $E(A, B)$,
  - whether A is not adjacent to any marked cluster (i.e., none of the adjacent nodes' centers has been marked) and whether $\{u, v\}$ has minimum rank in $E(A, \mathrm{Vor}(B))$, and
  - whether there is a marked cluster C adjacent to A so that A participates in $\mathcal{C}(C)$,

$\mathrm{Vor}(A)$ has minimum rank in $\mathrm{Vor}(\partial B) \cap \mathrm{Vor}(\partial C)$, and $\{u, v\}$ has minimum rank in $E(B, \mathrm{Vor}(A))$. We note that since we have degree bounded by Δ, the number of vertices in A is $\tilde{O}(n^{1/3})$, and the probability that a cell is marked is $n^{-1/3}$, the number of cluster-of-clusters that A participates in is $\tilde{O}(\Delta)$ w.h.p.

In other words, we can perform all necessary checks to decide whether $\{u, v\} \in E'$ or not.

The algorithm in Section 3 requires $\tilde{O}(n^{2/3})$ random bits for the selection of centers and marked clusters. For the emulation of the algorithm of Elkin and Neiman it suffices that the random variables $\{r_u\}$ be $\tilde{O}(n^{1/3})$-wise independent, because the outcome of the algorithm for a vertex $v \in R$ depends only on the random variables of at most $\tilde{O}(n^{1/3})$ vertices (the vertices in $\Gamma_\ell(v)$). Thus, overall $\tilde{O}(n^{2/3})$ random bits (up to $\mathrm{poly}(\Delta/\epsilon)$ factors) are sufficient. ◀

References

1. N. Alon, R. Rubinfeld, S. Vardi, and N. Xie. Space-efficient local computation algorithms. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1132–1139, 2012.
2. Artur Czumaj, Oded Goldreich, Dana Ron, C. Seshadhri, Asaf Shapira, and Christian Sohler. Finding cycles and trees in sublinear time. *Random Structures and Algorithms*, 45(2):139–184, 2014.
3. Michael Elkin and Ofer Neiman. Efficient algorithms for constructing very sparse spanners and emulators. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 652–669, 2017.
4. G. Even, M. Medina, and D. Ron. Deterministic stateless centralized local algorithms for bounded degree graphs. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 394–405, 2014. doi:10.1007/978-3-662-44777-2_33.
5. U. Feige, Y Mansour, and R. E. Schapire. Learning and inference in the presence of corrupted inputs. In *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015*, pages 637–657, 2015.
6. Hendrik Fichtenberger, Reut Levi, Yadu Vasudev, and Maximilian Wötzel. On testing minor-freeness in bounded degree graphs with one-sided error. Unpublished manuscript, 2017.
7. R. Levi, G. Moshkovitz, D. Ron, R. Rubinfeld, and A. Shapira. Constructing near spanning trees with few local inspections. *Random Structures & Algorithms*, 50:n/a–n/a, 2016. doi:10.1002/rsa.20652.
8. R. Levi and D. Ron. A quasi-polynomial time partition oracle for graphs with an excluded minor. *ACM Trans. Algorithms*, 11(3):24:1–24:13, 2015.
9. R. Levi, D. Ron, and R. Rubinfeld. Local algorithms for sparse spanning graphs. In *Proceedings of the Eighteenth International Workshop on Randomization and Computation (RANDOM)*, pages 826–842, 2014.
10. R. Levi, D. Ron, and R. Rubinfeld. Local algorithms for sparse spanning graphs. *CoRR*, abs/1402.3609, 2014. URL: http://arxiv.org/abs/1402.3609.
11. R. Levi, R. Rubinfeld, and A. Yodpinyanee. Local computation algorithms for graphs of non-constant degrees. *Algorithmica*, pages 1–24, 2016.
12. Y. Mansour, A. Rubinstein, S. Vardi, and N. Xie. Converting online algorithms to local computation algorithms. In *Automata, Languages and Programming: Thirty-Ninth International Colloquium (ICALP)*, pages 653–664, 2012.

13 Y. Mansour and S. Vardi. A local computation approximation scheme to maximum matching. In *Proceedings of the Sixteenth International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 260–273, 2013.

14 D. Peleg and A. A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13:99–116, 1989.

15 D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. *SIAM Journal on Computing*, 18:229–243, 1989.

16 L. S. Ram and E. Vicari. Distributed small connected spanning subgraph: Breaking the diameter bound. Technical report, ETH Zürich, 2011.

17 R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie. Fast local computation algorithms. In *Proceedings of The Second Symposium on Innovations in Computer Science (ICS)*, pages 223–238, 2011.

18 Ronitt Rubinfeld. Can we locally compute sparse connected subgraphs? In *Computer Science - Theory and Applications - 12th International Computer Science Symposium in Russia, CSR 2017, Kazan, Russia, June 8-12, 2017, Proceedings*, pages 38–47, 2017. doi:10.1007/978-3-319-58747-9_6.

Chain, Generalization of Covering Code, and Deterministic Algorithm for k-SAT

Sixue Liu
Department of Computer Science, Princeton University
35 Olden Street, Princeton, NJ 08540, USA
http://www.cs.princeton.edu/~sixuel
sixuel@cs.princeton.edu

Abstract

We present the current fastest deterministic algorithm for k-SAT, improving the upper bound $(2-2/k)^{n+o(n)}$ due to Moser and Scheder in STOC 2011. The algorithm combines a branching algorithm with the derandomized local search, whose analysis relies on a special sequence of clauses called chain, and a generalization of covering code based on linear programming.

We also provide a more intelligent branching algorithm for 3-SAT to establish the upper bound 1.32793^n, improved from 1.3303^n.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms

Keywords and phrases Satisfiability, derandomization, local search

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.88

Related Version A full version of the paper is available at https://arxiv.org/abs/1804.07901.

Acknowledgements I want to thank Yuping Luo, S. Matthew Weinberg and Periklis A. Papakonstantinou for helpful discussions, and the anonymous reviewers for their valuable comments.

1 Introduction

As the fundamental NP-complete problems, k-SAT and especially 3-SAT have been extensively studied for decades. Numerous conceptual breakthroughs have been put forward via continued progress of exponential-time algorithms, including randomized and deterministic ones.

The first provable algorithm solving k-SAT on n variables in less than 2^n steps is presented by Monien and Speckenmeyer, using the concept of autark assignment [10]. Later their bound 1.619^n for 3-SAT is improved to 1.579^n and 1.505^n respectively [15, 8]. One should note that these algorithms follow a branching manner, i.e., recursively reducing the formula size by branching and fixing variables deterministically, thus are called branching algorithms.

As for randomized algorithm, two influential ones are PPSZ and Schöning's local search [12, 16]. There has been a long line of research improving the bound $(4/3)^n$ of local search for 3-SAT, including HSSW and combining with PPSZ [5, 6], until Hertli closes the gap between unique and general cases for PPSZ [4] (by unique it means the formula has only one satisfying assignment). In a word, considering randomized algorithm, PPSZ for k-SAT is currently the fastest, although with one-sided error (see PPSZ in Table 1). Unfortunately, general PPSZ seems tough to derandomize due to the excessive usage of random bits [13].

In contrast to the hardness in derandomizing PPSZ, local search can be derandomized using the so-called covering code [2]. Subsequent deterministic algorithms focus on boosting local search for 3-SAT to the bounds 1.473^n and 1.465^n [1, 14]. In 2011, Moser and Scheder fully derandomize Schöning's local search with another covering code for the choice of flipping

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 88; pp. 88:1–88:13


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 The rounded up base c in the upper bound c^n of our deterministic algorithm for k-SAT and the corresponding upper bound in previous results [9, 11, 2] and randomized algorithm [12, 4].

k	Our Results	Makino et al.	Moser&Scheder	Dantsin et al.	PPSZ(randomized)
3	**1.32793**	1.3303	1.33334	1.5	1.30704
4	**1.49857**	-	1.50001	1.6	1.46895
5	**1.59946**	-	1.60001	1.66667	1.56943
6	**1.66646**	-	1.66667	1.71429	1.63788

variables within the unsatisfied clauses, which is immediately improved by derandomizing HSSW for 3-SAT, leading to the current best upper bounds for k-SAT (see Table 1) [11, 9]. Since then, all random coins in Schöning's local search are replaced by deterministic choices, and the bounds remain untouched. How to break the barrier?

The difficulty arises in both directions. If attacking this without local search, one has to derandomize PPSZ or propose radically new algorithm. Else if attacking this from derandomizing local search-based algorithm, one must greatly reduce the searching space.

Our method is a combination of a branching algorithm and the derandomized local search. As we mentioned in the second paragraph of this paper, branching algorithm is intrinsically deterministic, therefore it remains to leverage the upper bounds for both of them by some tradeoff. The tradeoff we found is the weighted size of set of chains, where a chain is a sequence of clauses sharing variable with the clauses next to them only, such that a branching algorithm either solves the formula within desired time or returns a large enough set of chains. The algorithm is based on the study of autark assignment from [10] with further refinement, whose output can be regarded as a generalization of maximal independent clauses set from HSSW [5], which reduces the k-CNF to a $(k-1)$-CNF. The searching space equipped with chains is rather different from those in previous derandomizations [2, 9, 11]: It is a Cartesian product of finite number of non-uniform spaces. Using linear programming, we prove that such space can be perfectly covered, and searched by derandomized local search within aimed time. Additionally, unlike the numerical upper bound in HSSW, we give the closed-form.

The rest of the paper is organized as follows. In §2 we give basic notations, definitions related to chain and algorithmic framework. We show how to generalize covering code to cover any space equipped with chains in §3. Then we use such code in derandomized local search in §4. In §5, we prove upper bound for k-SAT, followed by an intelligent branching algorithm for 3-SAT in §6 for improvement. Some upper bound results are highlighted in Table 1, with main results formally stated in Theorem 14 of §5 and Theorem 21 of §6.

2 Preliminaries

2.1 Notations

We study formula in Conjunctive Normal Form (CNF). Let $V = \{v_i | i \in [n]\}$ be a set of n boolean variables. For all $i \in [n]$, a literal l_i is either v_i or $\bar{v}_i$. A clause C is a disjunction of literals and a CNF F is a conjunction of clauses. A k-clause is a clause that consists of exactly k literals, and an $\leq k$-clause consists of at most k literals. If every clause in F is $\leq k$-clause, then F is a k-CNF.

An *assignment* is a function $\alpha : V \mapsto \{0, 1\}$ that maps each $v \in V$ to truth value $\{0, 1\}$. A *partial assignment* is the function restricted on $V' \subseteq V$. We use $F|\alpha(V')$ to denote the formula derived by fixing the values of variables in V' according to partial assignment $\alpha(V')$.

A clause C is said to be *satisfied* by α if α assigns at least one literal in C to 1. F is *satisfiable* iff there exists an α satisfying all clauses in F, and we call such α a *satisfying assignment* of F. The k-SAT problem asks to find a satisfying assignment of a given k-CNF F or to prove its non-existence if F is unsatisfiable.

Let X be a literal or a clause or a collection of either of them, we use $V(X)$ to denote the set of all the variables appear in X. We say that X and X' are *independent* if $V(X) \cap V(X') = \emptyset$, or X *overlaps* with X' if otherwise.

A *word* of length n is a vector from $\{0,1\}^n$. The *Hamming space* $H \subseteq \{0,1\}^n$ is a set of words. Given two words $\alpha_1, \alpha_2 \in H$, the *Hamming distance* $d(\alpha_1, \alpha_2) = \|\alpha_1 - \alpha_2\|_1$ is the number of bits α_1 and α_2 disagree. The reason of using α for word as same as for assignment is straightforward: Giving each variable an index $i \in [n]$, a word of length n naturally corresponds to an assignment, which will be used interchangeably.

Throughout the paper, n always denotes the number of variables in the formula and will be omitted if the context is clear. We use $O^*(f(n)) = \text{poly}(n) \cdot f(n)$ to suppress polynomial factor, and use $\mathcal{O}(f(n)) = 2^{o(n)} \cdot f(n)$ to suppress sub-exponential factor.

2.2 Preliminaries for Chain

In this subsection, we propose our central concepts, which are the basis of our analysis.

▶ **Definition 1.** Given integers $k \geq 3$ and $\tau \geq 1$, a *τ-chain* $\mathcal{S}^{(k)}$ is a sequence of τ k-clauses $\langle C_1, \ldots, C_\tau \rangle$ satisfies that $\forall i, j \in [\tau]$, $V(C_i) \cap V(C_j) = \emptyset$ iff $|i - j| > 1$.

If the context is clear, we will use $\mathcal{S}$, τ-chain or simply chain for short.

▶ **Definition 2.** A set of chains $\mathcal{I}$ is called an *instance* if $\forall \mathcal{S}, \mathcal{S}' \in \mathcal{I}$, $V(\mathcal{S}) \cap V(\mathcal{S}') = \emptyset$ for $\mathcal{S} \neq \mathcal{S}'$.

In other words, each clause in chain only and must overlap with the clauses next to it (if exist), and chains in an instance are mutually independent.

▶ **Definition 3.** Given chain $\mathcal{S}$, define the *solution space* of $\mathcal{S}$ as $A \subseteq \{0,1\}^{|V(\mathcal{S})|}$ such that partial assignment α on $V(\mathcal{S})$ satisfies all clauses in $\mathcal{S}$ iff $\alpha(V(\mathcal{S})) \in A$. [1]

We define vital algebraic property of chain, which will play a key role in the construction of covering code.

▶ **Definition 4.** Let A be the solution space of chain $\mathcal{S}^{(k)}$, define $\lambda \in \mathbb{R}$ and $\pi : A \mapsto [0,1]$ as the *characteristic value* and *characteristic distribution* of $\mathcal{S}^{(k)}$ respectively, where λ and π are feasible solution to the following linear programming LP_A:

$$\sum_{a \in A} \pi(a) = 1$$

$$\lambda = \sum_{a \in A} \left(\pi(a) \cdot \left(\frac{1}{k-1}\right)^{d(a,a^*)} \right) \qquad \forall a^* \in A$$

$$\pi(a) \geq 0 \qquad \forall a \in A$$

▶ Remark. The variables in LP_A are λ and $\pi(a)$ ($\forall a \in A$). There are $|A| + 1$ variables and $|A| + 1$ equality constraints in LP_A. One can work out the determinant of the coefficient matrix to see it has full rank, so the solution is unique if feasible. Specifically, $\lambda \in (0,1)$.

[1] This essentially defines the set of all satisfying assignments for a chain. As a simple example in 3-CNF, 1-chain $\langle x_1 \vee x_2 \vee x_3 \rangle$ has solution space $A = \{0,1\}^3 \backslash 0^3$.

Algorithm 1: Algorithmic Framework.

Input: k-CNF F
Output: a satisfying assignment or `Unsatisfiable`
1: $\mathsf{BR}(F)$ solves F or returns an instance $\mathcal{I}$
2: **if** F is not solved **then**
3: $\quad\mathsf{DLS}(F, \mathcal{I})$
4: **end if**

2.3 Algorithmic Framework

Our algorithm (Algorithm 1) is a combination of a branching algorithm called BR, and a derandomized local search called DLS. BR either solves F or provides a large enough instance to DLS for further use, which essentially reduces the Hamming space exponentially.

3 Generalization of Covering Code

First of all, we introduce the covering code, then show how to generalize it for the purpose of our derandomized local search.

3.1 Preliminaries for Covering Code

The *Hamming ball* of *radius* r and *center* α $B_\alpha(r) = \{\alpha' | d(\alpha, \alpha') \leq r\}$ is the set of all words with Hamming distance at most r from α. A *covering code* of *radius* r for Hamming space H is a set of words $C(r) \subseteq H$ satisfies $\forall \alpha' \in H, \exists \alpha \in C(r)$, such that $d(\alpha, \alpha') \leq r$, i.e., $H \subseteq \bigcup_{\alpha \in C(r)} B_\alpha(r)$, and we say $C(r)$ *covers* H.

Let ℓ be a non-negative integer and set $[\ell]^* = [\ell] \cup \{0\}$, a set of covering codes $\{C(r) | r \in [\ell]^*\}$ is an *ℓ-covering code* for H if $\forall r \in [\ell]^*, C(r) \subseteq H$ and $H \subseteq \bigcup_{r \in [\ell]^*} \bigcup_{\alpha \in C(r)} B_\alpha(r)$, i.e., $\{C(r) | r \in [\ell]^*\}$ covers H.

The following lemma gives the construction time and size of covering codes for the uniform Hamming spaces $\{0,1\}^n$.

▶ **Lemma 5** ([2]). *Given $\rho \in (0, \frac{1}{2})$, there exists a covering code $C(\rho n)$ for Hamming space $\{0,1\}^n$, such that $|C(\rho n)| \leq O^*(2^{(1-h(\rho))n})$ and $C(\rho n)$ can be deterministically constructed in time $O^*(2^{(1-h(\rho))n})$, where $h(\rho) = -\rho \log \rho - (1-\rho) \log (1-\rho)$ is the* binary entropy function.

3.2 Generalized Covering Code

In this subsection we introduce our generalized covering code, including its size and construction time.

First of all we take a detour to define the *Cartesian product* of σ sets of words as $X_1 \times \cdots \times X_\sigma = \prod_{i \in [\sigma]} X_i = \{\uplus_{i \in [\sigma]} \alpha_i | \forall i \in [\sigma], \alpha_i \in X_i\}$, where $\uplus_{i \in [\sigma]} \alpha_i$ is the concatenation from α_1 to α_σ. Then we claim that the Cartesian product of covering codes is also a good covering code for the Cartesian product of the Hamming spaces they covered separately. The proof of this general result can be found in the full version of the paper.

▶ **Lemma 6.** *Given integer $\chi > 1$, for each $i \in [\chi]$, let H_i be a Hamming space and $C_i(r_i)$ be a covering code for H_i. If $C_i(r_i)$ can be deterministically constructed in time $O^*(f_i(n))$ and $|C_i(r_i)| \leq O^*(g_i(n))$ for all $i \in [\chi]$, then there exists covering code $\mathfrak{C}$ of radius $\sum_{i \in [\chi]} r_i$ for Hamming space $\prod_{i \in [\chi]} H_i$ such that $\mathfrak{C}$ can be deterministically constructed in time $O^*(\sum_{i \in [\chi]} f_i(n) + \prod_{i \in [\chi]} g_i(n))$ and $|\mathfrak{C}| \leq O^*(\prod_{i \in [\chi]} g_i(n))$.*

Algorithm 2: Derandomized Local Search: DLS.

Input: k-CNF F, instance $\mathcal{I}$
Output: a satisfying assignment or `Unsatisfiable`

1: construct covering code $\mathfrak{C}$ for Hamming space $H(F,\mathcal{I})$ (Definition 9)
2: **for** every word $\alpha \in \mathfrak{C}$ **do**
3: **if** searchball-fast(F,α,r) find a satisfying assignment α^* for F **then**
4: **return** α^*
5: **end if**
6: **end for**
7: **return** `Unsatisfiable`

Our result on generalized covering code is given below. We give its proof sketch here, and the detailed proof can be found in full version of the paper.

▶ **Lemma 7.** *Let A be the solution space of chain $\mathcal{S}^{(k)}$ whose characteristic value is λ, for any $\nu = \Theta(n)$, there exists an ℓ-covering code $\{C(r)|r \in [\ell]^*\}$ for Hamming space $H = A^\nu$ where $\ell = \lfloor -\nu \log_{k-1} \lambda + 2 \rfloor$, such that $|C(r)| \leq O^*(\lambda^{-\nu}/(k-1)^r)$ and $C(r)$ can be deterministically constructed in time $O^*(\lambda^{-\nu}/(k-1)^r)$, for all $r \in [\ell]^*$.*

Proof Sketch. Firstly, we show the existence of such ℓ-covering code by a probabilistic method. For each $r \in [l]^*$, we build $C(r)$ from $\emptyset$ by repeating the following for $O^*(\lambda^{-\nu}/(k-1)^r)$ times independently: Choose ν words independently from A according to characteristic distribution π (Definition 4) and concatenate them to get a word $\alpha \in A^\nu$, then add α to $C(r)$ with replacement. Clearly, $|C(r)| \leq O^*(\lambda^{-\nu}/(k-1)^r)$. By deliberately choosing the repeating rounds, we can prove that the probability of the event that any code in A^ν is not covered by $C(r)$ is extremely small, such that a union bound for all codes is strictly less than 1, therefore proved the existence.

Secondly, we construct the code deterministically. W.l.o.g., let $d \geq 2$ be a constant divisor of ν. By partitioning H into d blocks and applying the approximation algorithm for the set covering problem in [2], we obtain a good covering code for each block within affordable time. Then we concatenate all covering code by taking their Cartesian product, and the size and construction time follows from Lemma 6. ◀

4 Derandomized Local Search

In this section, we present our derandomized local search (DLS), see Algorithm 2.

The algorithm first constructs the generalized covering code and stores it (Line 1), then calls searchball-fast (Line 3) to search inside each Hamming ball, where searchball-fast refers to the same algorithm proposed in [11], whose running time is stated in the following lemma.

▶ **Lemma 8** ([11]). *Given k-CNF F, if there exists a satisfying assignment α^* for F in $B_\alpha(r)$, then α^* can be found by searchball-fast in time $(k-1)^{r+o(r)}$.*

Our generalized covering code is able to cover the following Hamming space.

▶ **Definition 9.** Given k-CNF F and instance $\mathcal{I}$, the Hamming space for F and $\mathcal{I}$ is defined as $H(F,\mathcal{I}) = H_0 \times \prod_i H_i$, where:

- $H_0 = \{0,1\}^{n'}$ where $n' = n - |V(\mathcal{I})|$.
- $H_i = {A_i}^{\nu_i}$ for all i, where A_i is a solution space and $\nu_i = \Theta(n)$ is the number of chains in $\mathcal{I}$ with solution space A_i. [2]

[2] As we shall see in §5 and §6, there are only finite number of different solution spaces and finite elements

Apparently all satisfying assignments of F lie in $H(F,\mathcal{I})$, because $\prod_i H_i$ contains all assignments on $V(\mathcal{I})$ which satisfy all clauses in $\mathcal{I}$ and H_0 contains all possible assignments of variables outside $\mathcal{I}$. Therefore to solve F, it is sufficient to search the entire $H(F,\mathcal{I})$.

▶ **Definition 10.** Given $\rho \in (0, \frac{1}{2})$ and Hamming space $H(F,\mathcal{I})$ as above, for $L \in \mathbb{Z}^*$, define covering code $\mathfrak{C}(L)$ for $H(F,\mathcal{I})$ as a set of covering codes $\{C(r)|(r-\rho n') \in [L]^*\}$ satisfies that $C(r) \subseteq H(F,\mathcal{I})$ for all r and $H(F,\mathcal{I}) \subseteq \bigcup_{(r-\rho n')\in[L]^*} \bigcup_{\alpha\in C(r)} B_\alpha(r)$, i.e., $\mathfrak{C}(L)$ covers $H(F,\mathcal{I})$.

▶ **Lemma 11.** *Given Hamming space $H(F,\mathcal{I})$ and A_i, ν_i as above, let $L = \sum_i \ell_i$ where $\ell_i = \lfloor -\nu_i \log \lambda_i + 2 \rfloor$ and λ_i is the characteristic value of chain with solution space A_i. Given $\rho \in (0, \frac{1}{2})$, covering code $\mathfrak{C}(L) = \{C(r)|(r-\rho n') \in [L]^*\}$ for $H(F,\mathcal{I})$ can be deterministically constructed in time $O^*(2^{(1-h(\rho))n'} \prod_i {\lambda_i}^{-\nu_i})$ and $|C(r)| \leq O^*(2^{(1-h(\rho))n'}/(k-1)^{r-\rho n'} \prod_i {\lambda_i}^{-\nu_i})$ for all $(r-\rho n') \in [L]^*$.*

Proof. To construct $\mathfrak{C}(L)$ for $H(F,\mathcal{I})$, we construct covering code $C_0(\rho n')$ for $H_0 = \{0,1\}^{n'}$ and ℓ_i-covering code for $H_i = {A_i}^{\nu_i}$ for all i, then take a Cartesian product of all the codes. By Lemma 5, the time taken for constructing $C_0(\rho n')$ is $O^*(2^{(1-h(\rho))n'})$, and $|C_0(\rho n')| \leq O^*(2^{(1-h(\rho))n'})$. By Lemma 7, for each i, the time taken for constructing $C(r_i)$ for each $r_i \in [\ell_i]^*$ is $O^*({\lambda_i}^{-\nu_i}/(k-1)^{r_i})$ and $|C(r_i)| \leq O^*({\lambda_i}^{-\nu_i}/(k-1)^{r_i})$. So by Lemma 6, we have that $|C(r)|$ can be upper bounded by:

$$2^{(1-h(\rho))n'} \cdot \sum_{\sum_i r_i = r-\rho n'} \left(\prod_i O^*({\lambda_i}^{-\nu_i}/(k-1)^{r_i}) \right) = O^*(2^{(1-h(\rho))n'}/(k-1)^{r-\rho n'} \prod_i {\lambda_i}^{-\nu_i}).$$

The equality holds because L is a linear combination of ν_i with constant coefficients and $\nu_i = \Theta(n)$, thus there are $O(1)$ terms in the product since $\sum_i \nu_i \leq n$. Meanwhile, there are $O^*(1)$ ways to partition $(r-\rho n')$ into constant number of integers, thus the outer sum has $O^*(1)$ terms. Together we get an $O^*(1)$ factor in RHS.

The construction time includes constructing each covering code for H_i $(i \geq 0)$ and concatenating each of them by Lemma 6, which is dominated by the concatenation time. As a result, the time taken to construct $C(r)$ for all $(r-\rho n') \in [L]^*$ is:

$$\sum_{(r-\rho n')\in[L]^*} O^*(2^{(1-h(\rho))n'}/(k-1)^{r-\rho n'} \prod_i {\lambda_i}^{-\nu_i}) = O^*(2^{(1-h(\rho))n'} \prod_i {\lambda_i}^{-\nu_i}),$$

because it is the sum of a geometric series. Therefore conclude the proof. ◀

Using our generalized covering code and applying Lemma 8 for searchball-fast (Line 3 in Algorithm 2), we can upper bound the running time of DLS.

▶ **Lemma 12.** *Given k-CNF F and instance $\mathcal{I}$, DLS runs in time $T_{DLS} = \mathcal{O}((\frac{2(k-1)}{k})^{n'} \cdot \prod_i {\lambda_i}^{-\nu_i})$, where $n' = n - |V(\mathcal{I})|$, λ_i is the characteristic value of chain $\mathcal{S}_i$ and ν_i is number of chains in $\mathcal{I}$ with the same solution space to $\mathcal{S}_i$.*

Proof. The running time includes the construction time for $\mathfrak{C}(L)$ and the total searching time in all Hamming balls. It is easy to show that the total time is dominated by the

in each solution space. Thus for those $\nu_i = o(n)$, we can enumerate all possible combinations of assignments on them and just get a sub-exponential slowdown, i,e., an $\mathcal{O}(1)$ factor in the upper bound.

Algorithm 3: Branching Algorithm BR for k-SAT.

Input: k-CNF F
Output: a satisfying assignment or `Unsatisfiable` or an instance $\mathcal{I}$

staring from $\mathcal{I} \leftarrow \emptyset$, **for** 1-chain $\mathcal{S} : V(\mathcal{I}) \cap V(\mathcal{S}) = \emptyset$, **do** $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{S}$
if $|\mathcal{I}| < \nu n$ **then**
  for each assignment $\alpha \in \{\{0,1\}^k \backslash 0^k\}$ of $\mathcal{I}$ **do**
    solve $F|\alpha$ by deterministic $(k-1)$-SAT algorithm
    return the satisfying assignment if satisfiable
  end for
  return `Unsatisfiable`
else
  return $\mathcal{I}$
end if

searching time using Lemma 11, thus we have the following equation after multiplying a sub-exponential factor $\mathcal{O}(1)$ for the other $o(n)$ chains not in $\mathcal{I}$:

$$\begin{aligned} T_{\text{DLS}} &= \mathcal{O}(1) \cdot \sum_{(r-\rho n')\in[L]^*} \left(|C(r)| \cdot (k-1)^{r+o(r)} \right) \\ &= \mathcal{O}(1) \cdot \sum_{(r-\rho n')\in[L]^*} \left(O^*(2^{(1-h(\rho))n'}/(k-1)^{r-\rho n'} \prod_i {\lambda_i}^{-\nu_i}) \cdot (k-1)^{r+o(r)} \right) \\ &= \mathcal{O}(2^{(1-h(\rho)+\rho\log(k-1))n'} \cdot \prod_i {\lambda_i}^{-\nu_i}) = \mathcal{O}((\frac{2(k-1)}{k})^{n'} \cdot \prod_i {\lambda_i}^{-\nu_i}). \end{aligned}$$

The first equality follows from Lemma 8, the second inequality is from Lemma 11, and the last equality follows by setting $\rho = \frac{1}{k}$. Therefore we proved this lemma. ◀

5 Upper Bound for k-SAT

In this section, we give our main result on upper bound for k-SAT.

A simple branching algorithm BR for general k-SAT is given in Algorithm 3: Greedily construct a maximal instance $\mathcal{I}$ consisting of independent 1-chains and branch on all satisfying assignments of it if $|\mathcal{I}|$ is small. [3] After fixing all variables in $V(\mathcal{I})$, the remaining formula is a $(k-1)$-CNF due to the maximality of $\mathcal{I}$. Therefore the running time of BR is at most:

$$T_{\text{BR}} = \mathcal{O}((2^k-1)^{|\mathcal{I}|} \cdot {c_{k-1}}^{n-k|\mathcal{I}|}), \tag{1}$$

where $\mathcal{O}({c_{k-1}}^n)$ is the worst-case upper bound of a deterministic $(k-1)$-SAT algorithm.

On the other hand, since there are only 1-chains in $\mathcal{I}$, by Lemma 12 we have:

$$T_{\text{DLS}} = \mathcal{O}((\frac{2(k-1)}{k})^{n-k|\mathcal{I}|} \cdot \lambda^{-|\mathcal{I}|}). \tag{2}$$

It remains to calculate the characteristic value λ of 1-chain $\mathcal{S}^{(k)}$. We prove the following lemma for the unique solution of linear programming LP_A in Definition 4.

[3] W.l.o.g., one can negate all negative literals in $\mathcal{I}$ to transform the solution space of 1-chain to $\{0,1\}^k \backslash 0^k$.

▶ **Lemma 13.** *For 1-chain $\mathcal{S}^{(k)}$, let A be its solution space, then the characteristic distribution π satisfies*

$$\pi(a) = \frac{(k-1)^k}{(2k-2)^k-(k-2)^k} \cdot (1-(\frac{-1}{k-1})^{d(a,0^k)}) \text{ for all } a \in A,$$

and the characteristic value $\lambda = \frac{k^k}{(2k-2)^k-(k-2)^k}$.

Proof. We prove that this is a feasible solution to LP_A. Constraint $\pi(a) \geq 0$ ($\forall a \in A$) is easy to verify. To show constraint $\sum_{a\in A}\pi(a) = 1$ holds, let $y = d(a, 0^k)$ and note there are $\binom{k}{y}$ different $a \in A$ with $d(a, 0^k) = y$, then multiply $\frac{(2k-2)^k-(k-2)^k}{(k-1)^k}$ on both sides:

$$\frac{(2k-2)^k-(k-2)^k}{(k-1)^k} \cdot \sum_{a\in A}\pi(a) = \sum_{1\leq y\leq k}\left((1-(\frac{-1}{k-1})^y)\cdot\binom{k}{y}\right)$$

$$= \sum_{0\leq y\leq k}\binom{k}{y} - \sum_{0\leq y\leq k}\binom{k}{y}(\frac{-1}{k-1})^y = 2^k - (\frac{k-2}{k-1})^k.$$

Thus $\sum_{a\in A}\pi(a) = 1$ holds.

To prove $\lambda = \sum_{a\in A}\left(\pi(a)\cdot(\frac{1}{k-1})^{d(a,a^*)}\right)$, similar to the previous case, we multiply $\frac{(2k-2)^k-(k-2)^k}{(k-1)^k}$ on both sides. Note that adding the term at $a = 0^k$ does not change the sum, then for all $a^* \in A$, we have:

$$\text{RHS} = \sum_{a\in\{0,1\}^k}(1-(\frac{-1}{k-1})^{d(a,0^k)})\cdot(\frac{1}{k-1})^{d(a,a^*)}$$

$$= \sum_{a\in\{0,1\}^k}(\frac{1}{k-1})^{d(a,a^*)} - \sum_{a\in\{0,1\}^k}(-1)^{d(a,0^k)}(\frac{1}{k-1})^{d(a,0^k)+d(a,a^*)}.$$

The first term is equal to $(\frac{k}{k-1})^k = \text{LHS}$. To prove the second term is 0, note that $\exists i \in [k]$ such that some bit $a_i^* = 1$. Partition $\{0,1\}^k$ into two sets $S_0 = \{a \in \{0,1\}^k | a_i = 0\}$ and $S_1 = \{a \in \{0,1\}^k | a_i = 1\}$. We have the following bijection: For each $a \in S_0$, negate the i-th bit to get $a' \in S_1$. Then $d(a, 0^k) + d(a, a^*) = d(a', 0^k) + d(a', a^*)$ and $(-1)^{d(a,0^k)} = -(-1)^{d(a',0^k)}$, so the sum is 0. Therefore we verified the constraint and proved the lemma. ◀

Observe from (1) and (2) that T_{BR} is an increasing function of $|\mathcal{I}|$, while T_{DLS} is a decreasing function of it, so $T_{\mathsf{BR}} = T_{\mathsf{DLS}}$ gives the worst-case upper bound for k-SAT. We solve this equation by plugging in λ from Lemma 13 to get νn as the worst-case $|\mathcal{I}|$, and obtain the following theorem as our main result on k-SAT.

▶ **Theorem 14.** *Given $k \geq 3$, if there exists a deterministic algorithm for $(k-1)$-SAT that runs in time $\mathcal{O}({c_{k-1}}^n)$, then there exists a deterministic algorithm for k-SAT that runs in time $\mathcal{O}({c_k}^n)$, where $c_k = (2^k-1)^\nu \cdot {c_{k-1}}^{1-k\nu}$ and $\nu = \frac{\log(2k-2)-\log k-\log c_{k-1}}{\log(2^k-1)-\log(1-(\frac{k-2}{2k-2})^k)-k\log c_{k-1}}$.*

Note that the upper bound for 3-SAT implied by this theorem is $O(1.33026^n)$, but we can do better by applying Theorem 21 (presented later) for $c_3 = 3^{\log\frac{4}{3}/\log\frac{64}{21}} < 1.32793$ to prove all upper bounds for k-SAT ($k \geq 4$) in Table 1 of §1.

6 Upper Bound for 3-SAT

We provide a better upper bound for 3-SAT by a more intelligent branching algorithm.

First of all, we introduce some additional notations in 3-CNF simplification, then we present our branching algorithm for 3-SAT from high-level to all its components. Lastly we show how to combine it with derandomized local search to achieve a tighter upper bound.

Algorithm 4: Branching Algorithm BR for 3-SAT.

Input: 3-CNF F, clause sequence $\mathcal{C}$
Output: a satisfying assignment or `Unsatisfiable` or a clause sequence $\mathcal{C}$

1: simplify F by *procedure* $\mathcal{P}$
2: **if** $\bot \in F$ **then**
3: **return** `Unsatisfiable`
4: **else if** *condition* Φ holds **then**
5: stop the recursion, *transform* $\mathcal{C}$ to an instance $\mathcal{I}$ and **return** $\mathcal{I}$
6: **else if** F is 2-CNF **then**
7: deterministically solve F in polynomial time
8: **if** F is satisfiable **then**
9: stop the recursion and **return** the satisfying assignment
10: **else**
11: **return** `Unsatisfiable`
12: **end if**
13: **else**
14: choose a clause C according to *rule* Υ
15: for every satisfying assignment α_C of C, call $\mathsf{BR}(F|\alpha_C, \mathcal{C} \cup C^{\mathcal{F}})$
16: **return** `Unsatisfiable`
17: **end if**

6.1 Additional Notations

For every clause $C \in F$, if partial assignment α satisfies C, then C is removed in $F|\alpha$. Otherwise, the literals in C assigned to 0 under α are removed from C. If all the literals in C are removed, which means C is unsatisfied under α, we replace C by $\bot$ in $F|\alpha$. Let $G = F|\alpha$, for every $C \in F$, we use C^F to denote the clause C in F and $C^G \in G$ the new clause derived from C by assigning variables according to α. We use $\mathcal{F}$ to denote the original input 3-CNF without instantiating any variable, and $C^{\mathcal{F}}$ is called the *original form* of clause C.

Let $\mathsf{UP}(F)$ be the CNF derived by running *Unit Propagation* on F until there is no 1-clause in F. Clearly F is satisfiable iff $\mathsf{UP}(F)$ is satisfiable, and UP runs in polynomial time [3].

We will also use the set definition of CNF, i.e., for a CNF $F = \bigwedge_{i\in[m]} C_i$, it is equivalent to write $F = \{C_i | i \in [m]\}$. Define $\mathcal{T}(F), \mathcal{B}(F), \mathcal{U}(F)$ as the set of all the 3-clauses, 2-clauses and 1-clauses in F respectively. We have that any 3-CNF $F = \mathcal{T}(F) \cup \mathcal{B}(F) \cup \mathcal{U}(F)$.

6.2 Branching Algorithm for 3-SAT

In this subsection, we give our branching algorithm for 3-SAT (Algorithm 4). The algorithm is recursive and follows a depth-first search manner:

- Stop the recursion when certain conditions are met (Line 4 and Line 8).
- Backtrack when the current branch is unsatisfiable (Line 3, Line 11 and Line 16).
- Branch on all possible satisfying assignments on a clause and recursively call itself (Line 15). Return `Unsatisfiable` if all branches return `Unsatisfiable`.
- Clause sequence $\mathcal{C}$ stores all the branching clauses from root to the current node.

It is easy to show this algorithm is correct as long as *procedure* $\mathcal{P}$ maintains satisfiability.

In what follows, we introduce (i) the *procedure* $\mathcal{P}$ for simplification (Line 1); (ii) the clause choosing *rule* Υ (Line 14); (iii) the *transformation* from clause sequence to instance (Line 5); (iv) the termination *condition* Φ (Line 4). All of them are devoted to analyzing the running time of BR as a function of instance.

6.2.1 Simplification Procedure

The simplification relies on the following two lemmas.

▶ **Lemma 15** ([10]). *Given 3-CNF F and partial assignment α, define $\mathcal{TB}(F,\alpha) = \{C | C \in \mathcal{B}(\mathsf{UP}(F|\alpha), C^F \in \mathcal{T}(F)\}$. If $\bot \notin \mathsf{UP}(F|\alpha)$ and $\mathcal{TB}(F,\alpha) = \emptyset$, then F is satisfiable iff $\mathsf{UP}(F|\alpha)$ is satisfiable and α is called an* autark.

We refer our readers to Chapter 11 in [7] for a simple proof (also see full version of the paper). We also provide the following stronger lemma to further reduce the formula size.

▶ **Lemma 16.** *Given 3-CNF F and $(l_1 \vee l_2) \in \mathcal{B}(F)$, if $\exists C \in \mathcal{TB}(F, l_1 = 1)$ such that $l_2 \in C$, then F is satisfiable iff $F \backslash C^F \cup C$ is satisfiable.*

Proof. Clearly F is satisfiable if $F \backslash C^F \cup C$ is. Suppose $C = l_2 \vee l_3$ and let α be a satisfying assignment of F. If $\alpha(l_1) = 1$, then $\mathsf{UP}(F|l_1 = 1)$ is satisfiable, thus $F \backslash C^F \cup C$ is also satisfiable since $C \in \mathsf{UP}(F|l_1 = 1)$. Else if $\alpha(l_1) = 0$, then $\alpha(l_2) = 1$ due to $l_1 \vee l_2$, so α satisfies C and the conclusion follows. ◀

As a result, 3-CNF F can be simplified by the following polynomial-time *procedure* $\mathcal{P}$: for every $(l_1 \vee l_2) \in \mathcal{B}(F)$, if $l_1 = 1$ or $l_2 = 1$ is an autark, then apply Lemma 15 to simplify F; else apply Lemma 16 to simplify F if possible.

▶ **Lemma 17.** *After running $\mathcal{P}$ on 3-CNF F, for any $(l_1 \vee l_2) \in \mathcal{B}(F)$ and for any 2-clause $C \in \mathcal{TB}(F, l_1 = 1)$, it must be $l_2 \notin C$. This also holds when switching l_1 and l_2.*

Proof. If $\mathcal{TB}(F, l_1 = 1) = \emptyset$, then $l_1 = 1$ is an autark and F can be simplified by Lemma 15. If $C \in \mathcal{TB}(F, l_1 = 1)$ and $l_2 \in C$, then F can be simplified by Lemma 16. ◀

6.2.2 Clause Choosing Rule

Now we present our clause choosing *rule* Υ. By Lemma 15 we can always begin with branching on a 2-clause with a cost of factor 2 in the upper bound: Choose an arbitrary literal in any 3-clause and branch on its two assignments $\{0, 1\}$. This will result in a new 2-clause otherwise it is an autark and we fix it and continue to choose another literal.

Now let us show the overlapping cases between the current branching clause to the next branching clause. Let C_0 be the branching clause in the father node where $C_0^{\mathcal{F}} = l_0 \vee l_1 \vee l_2$, and let F_0 be the formula in the father node. The *rule* Υ works as follows: if $\alpha_{C_0}(l_1) = 1$, choose arbitrary $C_1 \in \mathcal{TB}(F_0, l_1 = 1)$; else if $\alpha_{C_0}(l_2) = 1$, choose arbitrary $C_1 \in \mathcal{TB}(F_0, l_2 = 1)$.

We only discuss the case $\alpha_{C_0}(l_1) = 1$ due to symmetry. We enumerate all the possible forms of $C_1^{\mathcal{F}}$ by discussing what literal is eliminated followed by whether l_2 or $\bar{l}_2$ is contained:

1. $C_1^{\mathcal{F}} \backslash C_1 = l_3$. C_1 becomes a 2-clause due to elimination of l_3. There are three cases: (i) $C_1 = l_2 \vee l_4$, (ii) $C_1 = \bar{l}_2 \vee l_4$ or (iii) $C_1 = l_4 \vee l_5$.
2. $C_1^{\mathcal{F}} \backslash C_1 = \bar{l}_1$. C_1 becomes a 2-clause due to elimination of $\bar{l}_1$. There are three cases: (i) $C_1 = l_2 \vee l_3$, (ii) $C_1 = \bar{l}_2 \vee l_3$ or (iii) $C_1 = l_3 \vee l_4$.
3. $C_1^{\mathcal{F}} \backslash C_1 = l_2$. This means $l_1 = 1 \Rightarrow l_2 = 0$, and $\alpha_{C_0}(l_1 l_2) = 11$ can be excluded.
4. $C_1^{\mathcal{F}} \backslash C_1 = \bar{l}_2$. This means $l_1 = 1 \Rightarrow l_2 = 1$, and $\alpha_{C_0}(l_1 l_2) = 10$ can be excluded.

Both Case 1.(i) and Case 2.(i) are impossible due to Lemma 17. To sum up, we immediately have the following by merging similar cases with branch number bounded from above:

- Case 1.(iii): it takes at most 3 branches in the father node to get $l_3 \vee l_4 \vee l_5$.
- Case 1.(ii), Case 2.(iii) and Case 4: it takes at most 3 branches in the father node to get $\bar{l}_1 \vee l_3 \vee l_4$ or $\bar{l}_2 \vee l_3 \vee l_4$.
- Case 3: it takes at most 2 branches in the father node to get $l_2 \vee l_3 \vee l_4$.
- Case 2.(ii): it takes at most 3 branches in the father node to get $\bar{l}_1 \vee \bar{l}_2 \vee l_3$.

To fit *rule* Υ, there must be at least one literal assigned to 1 in the branching clause. Except Case 2.(ii), we get a 2-clause C_1, and *rule* Υ still applies.

Now consider the case $C_1^{\mathcal{F}} = \bar{l}_1 \vee \bar{l}_2 \vee l_3$. If $\alpha(l_1 l_2) = 11$, we have $C_1^F = l_3$, otherwise we have $C_1^F = 1 \vee l_3$. In other words, the assignment satisfying $C_0 \wedge C_1$ should be $\alpha(l_1 l_2 l_3) \in \{010, 100, 011, 101, 111\}$. Note that $\alpha(l_3) = 0$ in the first two assignments, which does not fit *rule* Υ. In this case, we do the following: Choose an arbitrary literal in any 3-clause and branch on its two assignments $\{0, 1\}$. Continue this process we will eventually get a new 2-clause (Lemma 15). Now the first two assignments $\alpha(l_1 l_2 l_3) \in \{010, 100\}$ has 4 branches because of the new branched literal, and we have that all 7 branches fit *rule* Υ because either $l_3 = 1$ or there is a new 2-clause. Our key observation is the following: These 7 branches correspond to all satisfying assignments of $C_0 \wedge C_1$, which can be amortized to think that C_1 has 3 branches and C_0 has 7/3 branches. As a conclusion, we modify the last case to be:

- Case 2.(ii): it takes at most 7/3 branches in the father node to get $\bar{l}_1 \vee \bar{l}_2 \vee l_3$.

6.2.3 Transformation from Clause Sequence to Instance

We show how to transform a clause sequence $\mathcal{C}$ to an instance, then take a symbolic detour to better formalize the cost of generating chains, i.e., the running time of BR.

Similar to above, let C_1 be the clause chosen by *rule* Υ and let C_0 be the branching clause in the father node, moreover let C be the branching clause in the grandfather node. In other words, C_1, C_0, C are the last three clauses in $\mathcal{C}$. C_1 used to be a 3-clause in the father node since $C_1 \in \mathcal{T}(F)$, thus C_1 is independent with C because all literals in C are assigned to some values in F, so C_1 can only overlap with C_0. Therefore, clauses in $\mathcal{C}$ can only (but not necessarily) overlap with the clauses next to them.

By the case discussion in §6.2.2, there are only 4 overlapping cases between C_0 and C_1, which we call *independent* for $\langle l_0 \vee l_1 \vee l_2,\ l_3 \vee l_4 \vee l_5\rangle$, *negative* for $\langle l_0 \vee l_1 \vee l_2,\ \bar{l}_1 \vee l_3 \vee l_4\rangle$ or $\langle l_0 \vee l_1 \vee l_2,\ \bar{l}_2 \vee l_3 \vee l_4\rangle$, *positive* for $\langle l_0 \vee l_1 \vee l_2,\ l_2 \vee l_3 \vee l_4\rangle$ and *two-negative* for $\langle l_0 \vee l_1 \vee l_2,\ \bar{l}_1 \vee \bar{l}_2 \vee l_3\rangle$. There is a natural mapping from clause sequence to a string.

▶ **Definition 18.** Let $\mathcal{C}$ be a clause sequence, define function $\zeta : \mathcal{C} \mapsto \Gamma^{|\mathcal{C}|}$, where $\Gamma = \{*, \mathtt{n}, \mathtt{p}, \mathtt{t}\}$, satisfies that the i-th bit of $\zeta(\mathcal{C})$ is $*$ if $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$ are independent, or $\mathtt{n}$ if negative, or $\mathtt{p}$ if positive, or $\mathtt{t}$ if two-negative for all $i \in [|\mathcal{C}| - 1]$, and the $|\mathcal{C}|$-th bit of $\zeta(\mathcal{C})$ is $*$. A τ-chain $\mathcal{S}$ is also a clause sequence of length τ, so ζ maps $\mathcal{S}$ to Γ^τ. Two chains $\mathcal{S}_1$ and $\mathcal{S}_2$ are *isomorphic* if $\zeta(\mathcal{S}_1) = \zeta(\mathcal{S}_2)$.

Then the *transformation* from $\mathcal{C}$ to $\mathcal{I}$ naturally follows: Partition $\zeta(\mathcal{C})$ by $*$, then every substring corresponds to a chain, just add this chain to $\mathcal{I}$. Now we can formalize the cost.

▶ **Lemma 19.** *Given 3-CNF $\mathcal{F}$, let $\mathcal{C}$ be the clause sequence in time T of running BR$(\mathcal{F}, \emptyset)$, it must be $T \le O^*(2^{\kappa_1} \cdot 3^{\kappa_2} \cdot (7/3)^{\kappa_3})$, where κ_1 is the number of $\mathtt{p}$ in $\zeta(\mathcal{C})$, κ_2 is the number of $*$ and $\mathtt{n}$ in $\zeta(\mathcal{C})$, and κ_3 is the number of $\mathtt{t}$ in $\zeta(\mathcal{C})$.*

Proof. By Definition 18 and case discussion in §6.2.2, the conclusion follows. ◀

6.2.4 Termination Condition

We show how the cost of generating chains implies the termination *condition* Φ. We map every chain to an integer as the *type* of the chain such that isomorphic chains have the same type. Formally, let $\mathcal{I}(\mathcal{C})$ be the instance transformed from $\mathcal{C}$, and let $\Sigma = \{\zeta(\mathcal{S}) | \mathcal{S} \in \mathcal{I}(\mathcal{C})\}$ be the set of distinct strings with no repetition. Define bijective function $g : \Sigma \mapsto [\theta]$ that maps each string $\zeta(\mathcal{S})$ in Σ to a distinct integer as the *type* of chain $\mathcal{S}$, where $\theta = |\Sigma|$ is the number of types of chain in $\mathcal{C}$ and g can be arbitrary fixed bijection. Define *branch number* b_i of type-i chain $\mathcal{S}$ as $b_i = 2^{\kappa_1} \cdot 3^{\kappa_2} \cdot (7/3)^{\kappa_3}$, where κ_1 is the number of `p` in $\zeta(\mathcal{S})$, κ_2 is the number of `*` and `n` in $\zeta(\mathcal{S})$, and κ_3 is the number of `t` in $\zeta(\mathcal{S})$. Also define the *chain vector* $\vec{\nu} \in \mathbb{Z}^\theta$ for $\mathcal{I}(\mathcal{C})$ satisfies $\nu_i = |\{\mathcal{S} \in \mathcal{I}(\mathcal{C}) | (g \circ \zeta)(\mathcal{S}) = i\}|$ for all $i \in [\theta]$, i.e., ν_i is the number of type-i chains in $\mathcal{I}(\mathcal{C})$. We can rewrite Lemma 19 as the following.

▶ **Corollary 20.** *Given 3-CNF $\mathcal{F}$, let $\mathcal{I}$ be the instance in time T of running $\mathsf{BR}(\mathcal{F}, \emptyset)$, it must be $T \le T_{\mathsf{BR}} = O^*(\prod_{i\in[\theta]} b_i^{\nu_i})$, where b_i is the branch number of type-i chain and $\vec{\nu}$ is the chain vector for $\mathcal{I}$.*

To achieve worst-case upper bound $\mathcal{O}(c^n)$ for solving 3-SAT, we must have $T_{\mathsf{BR}} \le \mathcal{O}(c^n)$, which is $\prod_{i=1}^{\theta} b_i^{\nu_i} \le c^n$. This immediately gives us the termination *condition* Φ: $(\sum_{i\in[\theta]} \nu_i \cdot \log b_i)/\log c > n$.

Therefore, we can hardwire such condition into the algorithm to achieve the desired upper bound, as calculated in next subsection.

6.3 Combination of Two Algorithms

By combining BR and DLS as in Algorithm 1, we have that the worst-case upper bound $\mathcal{O}(c^n)$ is attained when $T_{\mathsf{BR}} = T_{\mathsf{DLS}}$, which is:

$$c^n = \prod_{i\in[\theta]} b_i^{\nu_i} = (\frac{4}{3})^{n'} \cdot \prod_{i\in[\theta]} {\lambda_i}^{-\nu_i}, \tag{3}$$

followed by Corollary 20 and Lemma 12. Let η_i be the number of variables in a type-i chain for all $i \in [\theta]$, we have that $n' = n - |V(\mathcal{I})| = n - \sum_{i\in[\theta]} \eta_i \nu_i$. Taking the logarithm and divided by n, (3) becomes:

$$\log c = \sum_{i\in[\theta]} \frac{\nu_i}{n} \log b_i = \log \frac{4}{3} - \sum_{i\in[\theta]} \frac{\nu_i}{n} (\eta_i \log \frac{4}{3} + \log \lambda_i). \tag{4}$$

The second equation is a linear constraint over $\frac{1}{n} \cdot \vec{\nu}$, which gives that $\log c$ is maximized when $\nu_i = 0$ for all $i \neq \arg\max_{i\in[\theta]} \{\log b_i/(\log b_i + \eta_i \log \frac{4}{3} + \log \lambda_i)\}$.

Based on calculation of LP_A (see full version of the paper), we show that chain $\mathcal{S}$ with $\zeta(\mathcal{S}) =$ `*` (say, type-1 chain) corresponds to the maximum value above, namely:

$$\arg\max_{i\in[\theta]} \{\log b_i/(\log b_i + \eta_i \log \frac{4}{3} + \log \lambda_i)\} = 1.$$

In other words, all chains in $\mathcal{I}$ are 1-chain. Substitute $\lambda_1 = \frac{3}{7}, b_1 = 3, \eta_1 = 3$ and $\nu_i = 0$ for all $i \in [2, \theta]$ into (4), we obtain our main result on 3-SAT as follow.

▶ **Theorem 21.** *There exists a deterministic algorithm for 3-SAT that runs in time $\mathcal{O}(3^{n \log \frac{4}{3} / \log \frac{64}{21}})$.*

This immediately implies the upper bound $O(1.32793^n)$ for 3-SAT in Table 1 of §1.

References

1. Tobias Brüggemann and Walter Kern. An improved deterministic local search algorithm for 3-sat. *Theoretical Computer Science*, 329(1-3):303–313, 2004.
2. Evgeny Dantsin, Andreas Goerdt, Edward A Hirsch, Ravi Kannan, Jon Kleinberg, Christos Papadimitriou, Prabhakar Raghavan, and Uwe Schöning. A deterministic $(2\text{-}2/(k+1))^n$ algorithm for k-sat based on local search. *Theoretical Computer Science*, 289(1):69–83, 2002.
3. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.
4. Timon Hertli. 3-sat faster and simpler—unique-sat bounds for ppsz hold in general. *SIAM Journal on Computing*, 43(2):718–729, 2014.
5. Thomas Hofmeister, Uwe Schöning, Rainer Schuler, and Osamu Watanabe. A probabilistic 3-sat algorithm further improved. In *19th Annual Symposium on Theoretical Aspects of Computer Science, STACS 2002*, pages 192–202. Springer, 2002.
6. Kazuo Iwama and Suguru Tamaki. Improved upper bounds for 3-sat. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004*, volume 4, pages 328–328, 2004.
7. Hans Kleine Büning and Oliver Kullmann. Minimal unsatisfiability and autarkies. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 339–401. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-339.
8. Oliver Kullmann. New methods for 3-sat decision and worst-case analysis. *Theoretical Computer Science*, 223(1-2):1–72, 1999.
9. Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto. Derandomizing the HSSW algorithm for 3-sat. *Algorithmica*, 67(2):112–124, 2013.
10. Burkhard Monien and Ewald Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10(3):287–295, 1985.
11. Robin A. Moser and Dominik Scheder. A full derandomization of schöning's k-sat algorithm. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing, STOC 2011*, pages 245–252, 2011.
12. Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-sat. *J. ACM*, 52(3):337–364, 2005. doi:10.1145/1066100.1066101.
13. Daniel Rolf. Derandomization of PPSZ for unique- k-sat. In *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005*, pages 216–225, 2005.
14. Dominik Scheder. Guided search and a faster deterministic algorithm for 3-sat. In *the 3rd Latin American Theoretical Informatics Symposium, LATIN 2008*, pages 60–71, 2008.
15. Ingo Schiermeyer. Solving 3-satisfiability in less than 1.579^n steps. *Computer Science Logic*, pages 379–394, 1970.
16. Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS 1999*, pages 410–414, 1999.

Stable-Matching Voronoi Diagrams: Combinatorial Complexity and Algorithms

Gill Barequet
Technion – Israel Inst. of Technology, Haifa, Israel
barequet@cs.technion.ac.il

David Eppstein
University of California, Irvine, U.S.
eppstein@uci.edu

Michael T. Goodrich
University of California, Irvine, U.S.
goodrich@uci.edu

Nil Mamano
University of California, Irvine, U.S.
nmamano@uci.edu

Abstract

We study algorithms and combinatorial complexity bounds for *stable-matching Voronoi diagrams*, where a set, S, of n point sites in the plane determines a stable matching between the points in $\mathbb{R}^2$ and the sites in S such that (i) the points prefer sites closer to them and sites prefer points closer to them, and (ii) each site has a quota indicating the area of the set of points that can be matched to it. Thus, a stable-matching Voronoi diagram is a solution to the classic post office problem with the added (realistic) constraint that each post office has a limit on the size of its jurisdiction. Previous work provided existence and uniqueness proofs, but did not analyze its combinatorial or algorithmic complexity. We show that a stable-matching Voronoi diagram of n sites has $O(n^{2+\varepsilon})$ faces and edges, for any $\varepsilon > 0$, and show that this bound is almost tight by giving a family of diagrams with $\Theta(n^2)$ faces and edges. We also provide a discrete algorithm for constructing it in $O(n^3 + n^2 f(n))$ time, where $f(n)$ is the runtime of a geometric primitive that can be performed in the real-RAM model or can be approximated numerically. This is necessary, as the diagram cannot be computed exactly in an algebraic model of computation.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Voronoi diagram, stable matching, combinatorial complexity, lower bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.89

Related Version A full version of the paper is available at [6], `https://arxiv.org/abs/1804.09411`.

Funding This work was partially supported by DARPA under agreement no. AFRL FA8750-15-2-0092 and NSF grants 1228639, 1526631,2 1217322, 1618301, and 1616248. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

Acknowledgements We would like to thank Nina Amenta for several helpful discussions regarding the topics of this paper. We also thank the anonymous reviewers for many useful comments.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 89; pp. 89:1–89:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The *Voronoi diagram* is a well-known geometric structure with a broad spectrum of applications in computational geometry and other areas of Computer Science, e.g., see [3, 5, 8, 18, 20, 22, 24, 7]. The Voronoi diagram partitions the plane into regions. Given a finite set S of points, called sites, each point in the plane is assigned to the region of its closest site in S. Although the Voronoi diagram has been generalized in many ways, its standard definition specifies that each *Voronoi cell* or *region* of a site s is the set $V(s)$ defined as $\{p \in \mathbb{R}^2 \mid d(p,s) \leq d(p,s') \quad \forall s' \neq s \in S\}$, where $d(\cdot,\cdot)$ denotes the distance between two points. The properties of standard Voronoi diagrams have been thoroughly studied (e.g., see [3, 5]). For example, it is well-known that in a standard Voronoi diagram for point sites in the plane every Voronoi cell is a connected, convex polygon whose boundaries lie along perpendicular bisectors of pairs of sites.

On a seemingly unrelated topic, the theory of *stable matchings* studies how to match entities in two sets, each of which has its own preferences about the elements of the other set, in a "stable" manner. It is used, for instance, to match hospitals and medical students starting their residencies [21], as well as in on-line advertisement auctions (e.g., see [2]). It was originally formulated by Gale and Shapley [12] in the context of establishing marriages between n men and n women, where each man ranks the women by preference, and the women rank the men. A matching between the men and women is *stable* if there is no *blocking pair*, that is, a man and woman who prefer each other over their assigned partners according to the matching. Gale and Shapley [12] show that a stable solution always exists for any set of preferences, and they provide an algorithm that runs in $O(n^2)$ time.

When generalized to the one-to-many case, the stable matching problem is also known as the *college admission* problem [23] and can be formulated as a matching of n students to k colleges, where each student has a preference ranking of the colleges and each college has a preference ranking of the students and a *quota* indicating how many students it can accept.

In this paper, we are interested in studying the algorithmic and combinatorial complexity of the diagrams that we call *stable-matching Voronoi diagrams*, which combine the notions of Voronoi diagrams and the one-to-many stable matching problem. These diagrams were introduced by Hoffman *et al.* [15], who provided a mathematical definition and existence and uniqueness proofs for such structures for potentially countably infinite sets of sites, but they did not study their algorithmic or combinatorial complexities. A stable-matching Voronoi diagram is defined with respect to a set of sites in $\mathbb{R}^2$, which in this paper we restrict to finite sets of n distinct points, each of which has an assigned numerical *quota* (which is also known as its "*appetite*") indicating the area of the region of points assigned to it. A preference relationship is defined in terms of distance, so that each point p in $\mathbb{R}^2$ prefers sites ordered by distance, from closest to farthest, and each site likewise prefers points ordered by distance. The stable-matching Voronoi diagram, then, is a partition of the plane into regions, such that (i) each site is associated with a region of area equal to its appetite, and (ii) the assignment of points to sites is stable in the sense that there is no site–point pair whose members prefer each other over their assigned matches. See Figure 1.

Formally, we can define a stable-matching Voronoi diagram for a set of sites as follows.

▶ **Definition 1.** Given a set S of n points (called sites) in $\mathbb{R}^2$ and a numerical appetite $A_s > 0$ for each $s \in S$, the *stable-matching Voronoi diagram* of S is a division of $\mathbb{R}^2$ into $n+1$ regions, such that for each site $s \in S$ there is a corresponding region C_s of area A_s, and there is an extra region, $C_\emptyset$, for the the remaining "unmatched" points, and such that there are no blocking pairs. A blocking pair is a site $s \in S$ and a point $p \in \mathbb{R}^2$ such that (i)

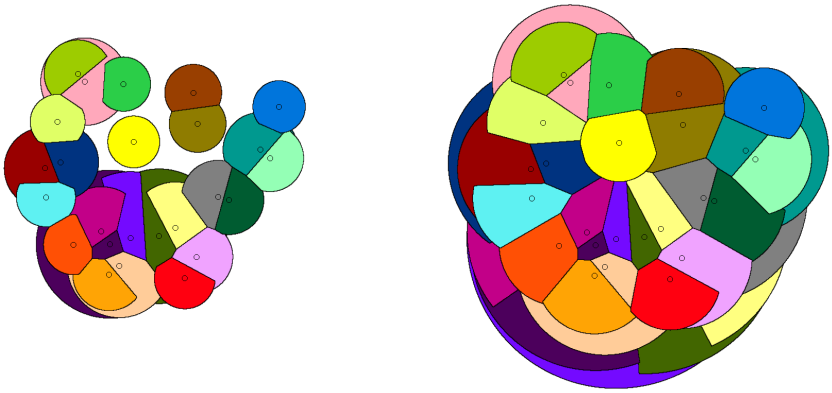


Figure 1 Stable-matching Voronoi diagrams for a set of 25 sites, where each site in the left diagram has an appetite of 1 and each site in the right diagram has an appetite of 2. Each color corresponds to an individual cell, which is not necessarily convex or even connected.

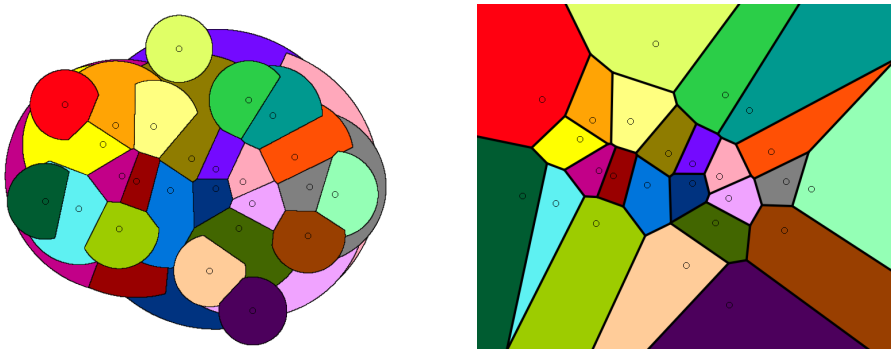


Figure 2 A stable-matching Voronoi diagram (left) and a standard Voronoi diagram (right) for the same set of 25 sites. Each color represents a region.

$p \notin C_s$, (*ii*) $d(p,s) < \max\{d(p',s) \mid p' \in C_s\}$, and (*iii*) $p \in C_\emptyset$ or $d(p,s) < d(p,s')$, where s' is the site such that $p \in C_{s'}$.

Figure 2 shows a side-by-side comparison of the standard and stable-matching Voronoi diagrams. Note that the standard Voronoi diagram is stable in the same sense as the stable-matching Voronoi diagram. This is because, by definition, every point is matched to its first choice among the sites, so there can be no blocking pairs. Thus, standard Voronoi diagram can be seen as a stable-matching Voronoi diagram with infinite appetites, in the following sense: for any point p in $\mathbb{R}^2$, for sufficiently large appetites for all the sites, p will belong to the region of the same site in the standard and stable-matching Voronoi diagrams.

In this paper, we focus on the case where all the appetites are equal, i.e., $A_s = A$, for some $A > 0$, but our results are easily adapted to the general case of arbitrary positive appetites.

A stable-matching Voronoi diagram solves the *post office* problem [19], of assigning points to their closest post office, under the (realistic) real-world assumption that each post office has an upper bound on the size of its jurisdiction. Such notions may also be useful for political districting, where sites could, for instance, represent polling stations, and appetites could represent their capacities. In this context, the distance preferences for a stable-matching Voronoi diagram might determine a type of "compactness" that avoids the strange regions

that are the subjects of recent court cases involving gerrymandering. Nevertheless, depending on the appetites and sites, the regions for sites in a stable-matching Voronoi diagram are not necessarily convex or even connected (e.g., see Figure 1). Thus, we are interested in this paper in characterizing the worst-case combinatorial complexity of such diagrams as well as finding an efficient algorithm for constructing them.

Previous Related Work. There are large volumes of work on the topics of Voronoi diagrams and stable matchings; hence, we refer the interested reader to surveys or books on the subjects (e.g., see [3, 5, 14, 16]).

A generalization of Voronoi diagram of particular relevance are *power diagrams*, where a weight associated to each site indicates how strongly the site draws the points in its neighborhood. Aurenhammer *et al.* [4] showed that, given a quota for each site, it is always possible to find weights for the sites such that, in the power diagram, the region of each site will have area equal to its prescribed quota. Thus, both stable-matching Voronoi diagrams and power diagrams are Voronoi-like diagrams with predetermined region sizes. Power diagrams minimize the total square distance between the sites and their associated points, while stable-matching Voronoi diagrams result in a stable matching.

As mentioned above, Hoffman *et al.* [15] gave definitions and existence and uniqueness proofs for the structures we call stable-matching Voronoi diagrams, albeit for potentially countably infinite sets of point sites. Rather than giving a discrete algorithm for constructing such diagrams, however, they described a continuous process that results in the stable-matching Voronoi diagram: Start growing a circle from all the sites at the same time and at the same rate, matching the sites with all the points encountered by the circles that are not matched yet – when a site fulfills its appetite, its circle stops growing. The process ends when all the circles have stopped growing. Of course, such a continuous process could be approximated numerically, but it is not an effective discrete algorithm, which is one of the interests of the present paper.

In other previous work, Eppstein *et al.* [10] studied the problem of constructing a stable-matching Voronoi diagram in a discrete $n \times n$ grid setting, where both sites and points are pixels. Later, Eppstein *et al.* [9] considered a related stable-matching problem in the context of planar graphs and road networks. In these two previous works, the entities analogous to sites and points are either pixels or vertices; hence, these previous algorithms did not encounter the algorithmic and combinatorial challenges raised by stable-matching Voronoi diagrams for sites and points in the plane.

Our Contributions. In Section 2, we give a geometric interpretation of stable-matching Voronoi diagrams as the lower envelope of a set of cones, and discuss some basic properties of stable-matching Voronoi diagrams.

In Section 3, we give an $O(n^{2+\varepsilon})$ upper bound, for any $\varepsilon > 0$, and an $\Omega(n^2)$ lower bound for the number of faces and edges of a stable-matching Voronoi diagrams in the worst case, where n is the number of sites.

In Section 4, we show that stable-matching Voronoi diagrams cannot be computed exactly in an algebraic model of computation. In light of this, we provide a discrete algorithm for constructing them that runs in $O(n^3 + n^2 f(n))$ time, where $f(n)$ is the runtime of a geometric primitive (which we identify) that encapsulates this difficulty. The geometric primitive can be computed in the real-RAM model or can be approximated numerically. We conclude in Section 5.

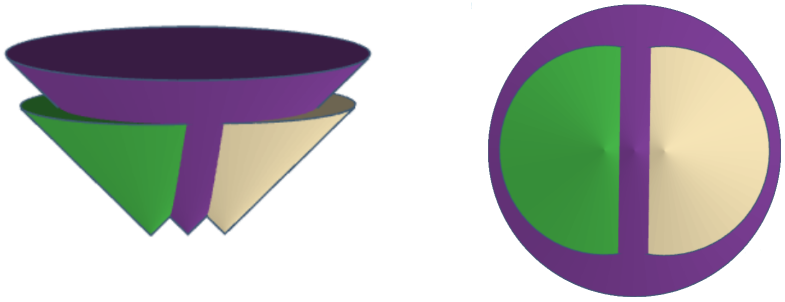


Figure 3 View of a stable-matching Voronoi diagram of 3 sites as the lower envelope of a set of cones.

2 The Geometry of Stable-Matching Voronoi Diagrams

As is now well-known, a (2-dimensional) Voronoi diagram can be viewed as a lower envelope of cones in 3 dimensions, as follows [11]. Suppose that the sites are embedded in the plane $z = 0$. That is, we map each site $s = (x_s, y_s)$ to the 3-dimensional point $(x_s, y_s, 0)$. Then, we draw one cone for each site, with the site as the apex, and growing to $+\infty$ all with the same slope. If we then view the cones from below, i.e., from $z = -\infty$ towards $z = +\infty$, the part of the cone of each site that we see corresponds to the Voronoi cell of the site. This is because two such cones intersect at points that are equally distant to the two apices. As a result, the xy-projection of their intersection corresponds to the perpendicular bisector of the apices, and the boundaries of the Voronoi cells in the Voronoi diagram are determined by the perpendicular bisectors with neighboring sites.

Similarly, a stable-matching Voronoi diagram can also be viewed as the lower envelope of a set of cones, where, instead of extending to $+\infty$, cones are cut off at a finite height (which is a potentially different height for each cone, even if the associated sites have the same appetite). This system of cones can be generated by a dynamic process that begins with cones of height zero and then grows them all at the same rate, halting the growth of each cone as soon as its area in the lower envelope reaches its appetite (see Figure 3).

A stable-matching Voronoi diagram consists of three types of elements:

- A *face* is a maximal, closed, connected subset of a stable cell. The stable cells can be disconnected, that is, a cell can have more than one face. There is also one or more *empty faces*, which are maximal connected regions not assigned to any site. One of the empty faces is the *external face*, which is the only face with infinite area.
- An *edge* is a maximal line segment or circular arc on the boundary of two faces. We call the two types of edges *straight* and *curved edges*, respectively. For curved edges, we distinguish between its incident convex face (the one inside the circle along which the edge lies) and its incident concave face.
- A *vertex* is a point shared by more than one edge. Generally, edges end at vertices, but curved edges may have no endpoints when they form a complete circle. This situation arises when the region of a site is isolated from other sites.

For a given appetite, shared by all the sites, we say that sites are not in general position if two curved boundaries of the stable-matching Voronoi diagram touch at a point, p, that is not an endpoint (e.g., two circles of radius 1 with centers 2 units apart) and, hence, for those curved edges, the concave face is the same at both sides of p. In this special case, we consider that the curved edges end at the vertex p.

In order to study the topology of the distance-stable Voronoi diagram, let the *bounding*

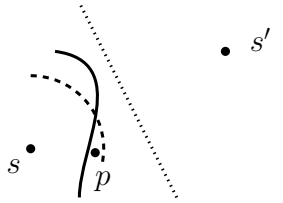


Figure 4 Illustration of the setting in the proof of Lemma 2. It shows the perpendicular bisector of two sites s and s' (as a dotted line), the boundary of the bounding disk, B_s, of s (as a dashed circular arc), and a hypothetical boundary between the regions of sites s and s' (as a solid curve). In this setting, s and p would be a blocking pair.

disk, B_s, of a site, s, be the smallest (closed) disk centered at s that contains the stable cell of s. The bounding disks arise in the topology of the diagram due to the following lemma:

▶ **Lemma 2.** *If part of the boundary between a face of site s and a face of site s' lies in the half-plane closer to s than to s', then that part of the boundary must lie along the boundary of the bounding disk B_s of s, and the convex face must belong to s.*

Proof. The boundary between the faces of s and s' cannot lie outside of B_s, by definition of the bounding disk. If the boundary is in the half-plane closer to s, then it also cannot be in the interior of B_s, because then there would exist a point p inside B_s and in the half-plane closer to s, but matched to s' (see Figure 4). In such a situation, s and p would be a blocking pair: s prefers p to the point(s) matched to it along B_s, and p prefers s to s'. ◀

▶ **Lemma 3.** *The union of non-empty faces of the diagram is the union of the bounding disks of all the sites.*

Proof. For any site s, all the points inside the bounding disk of s must be matched. Otherwise, there would be a point, say, p, not matched to anyone but closer to s than points actually matched to s (along the boundary of B_s), which would be unstable, as p and s would be a blocking pair. Moreover, points outside of all the bounding disks cannot be matched to anyone, by definition of the bounding disks. ◀

▶ **Lemma 4** (Characterization of edges)**.**

1. *A straight edge separating faces of sites s and s' can only lie along the perpendicular bisector of s and s'.*
2. *A curved edge whose convex face belongs to site s lies along the boundary of the bounding disk of s. Moreover, if the concave face belongs to a site s', the edge must be contained in the half-plane closer to s than s'.*
3. *Empty faces can only be concave faces of curved edges.*

Proof. Claims (1) and (2) are consequences of Lemma 2, and Claim (3) is a consequence of Lemma 3. ◀

Generally, every site has at least one curved edge. The only case where that may not happen is when the appetite of a site is exactly the same as the area of its Voronoi cell.

3 Combinatorial Complexity

3.1 An Upper Bound on the Number of Faces

As mentioned in Section 2, a stable-matching Voronoi diagram can be viewed as the lower envelope of a set of cones. The study of *Davenport-Schinzel sequences* has yielded results that characterize the combinatorial complexity of the lower envelope of certain sets of functions (e.g., see Sharir and Agarwal [1]), including cones.

Formally, the *lower envelope* (also called *minimization diagram*) of a set of bivariate continuous functions $F = \{f_1(x, y), \ldots, f_n(x, y)\}$ is the function $E_F(x, y) = \min_{1 \leq i \leq n} f_i(x, y)$. The lower envelope of F partitions the plane into maximal connected regions such that E_F is attained by a single function f_i (or by no function at all). The *combinatorial complexity* of the lower envelope E_F, denoted $K(F)$, is the number of maximal connected regions of E_F. To prove our upper bound, we use the following result:

▶ **Lemma 5.** *Sharir and Agarwal [1, p. 191] The combinatorial complexity $K(F)$ of the lower envelope of a collection F of n (partially defined) functions that satisfy the assumptions below* [1] *is $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$.*

- *Each $f_i \in F$ is a portion of an algebraic surface of the form $P(x_1, \ldots, x_d) = 0$, for some polynomial P of constant maximum degree.*
- *The vertical projection of each $f_i \in F$ onto the xy-plane is a planar region bounded by a constant number of algebraic arcs of constant maximum degree.*

▶ **Corollary 6.** *A stable-matching Voronoi diagram for n sites has $O(n^{2+\varepsilon})$ faces, for any $\varepsilon > 0$.*

Proof. It is clear that the finite, "upside-down" cones whose lower envelope forms the stable-matching Voronoi diagram of a set of sites satisfy the assumptions from Lemma 5. In particular, their projection onto the xy-plane are disks. Note that the bound still applies if we include the empty faces, as the assumptions still hold if we add an extra bivariate function $f_{n+1}(x, y) = z^*$, where z^* is any value higher than the height of any cone (i.e., f_{n+1} is a plane that "hovers" over the cones). Such a function would have a face in the lower envelope for each empty face in the stable-matching Voronoi diagram. ◀

3.2 An Upper bound on the Number of Edges and Vertices

Euler's formula relates the number of faces in a planar graph with the number of vertices and edges. By viewing the stable-matching Voronoi diagram as a graph, we can use Euler's formula to prove that the $O(n^{2+\varepsilon})$ upper bound also applies to the number of edges and vertices. In order to do so, we need the following lemma, for which we defer the proof to the full version of the paper [6] due to space constraints.

▶ **Lemma 7.** *The average degree is at least* 2.25.

In this section (Lemmas 7 and 8), we assume that sites are in general position (as defined in Section 2). However, note that non-general-position constructions cannot yield the worst-case complexity. This is because if two curved boundaries coincide exactly at a point that is not an endpoint, we can perturb slightly the site locations to move them a little closer, which creates a new vertex and edge.

[1] The theorem, as stated in [1] (Theorem 7.7), includes some additional assumptions, but then shows that they are not essential.

▶ **Lemma 8.** *Let V, E and F be the number of vertices, edges, and faces of the stable-matching Voronoi diagram of a set of sites S. Then, $V \leq 8F - 16$ and $E \leq 9F - 18$.*

Proof. For this proof, suppose that there are no curved edges that form a full circle. Note that the presence of such edges can only reduce the number of vertices and edges, as for each such edge there is a site with a single edge and no vertices.

Without such edges, the vertices and edges of the stable-matching Voronoi diagram form a planar graph, and V, E, F are the number of vertices, edges, and faces of this graph, respectively. Moreover, let C be the number of connected components. Due to Euler's formula for planar graphs, we have $F = E - V + C + 1$, and thus $F \geq E - V + 2$. Moreover, by Lemma 7, the sum of degrees is at least $2.25V$, so $2E \geq 2.25V$. Combining the two relations above, we have $V \leq 8F - 16$ and $E \leq 9F - 18$. ◀

We conclude by stating the main theorem of this section, which is a combination of Corollary 6 and Lemma 8:

▶ **Theorem 9.** *A stable-matching Voronoi diagram for n sites has $O(n^{2+\varepsilon})$ faces, vertices, and edges, for any $\varepsilon > 0$.*

3.3 Lower Bound

▶ **Lemma 10.** *A stable-matching Voronoi diagram for n sites has $\Omega(n^2)$ faces in the worst case.*

Proof. Let $m = \lfloor n/2 \rfloor$. We will construct a diagram in two steps, first placing m sites, and then m more. If n is odd, the remaining site can be placed anywhere in the plane that does not intersect with the others.

First, consider the stable-matching Voronoi diagram of m sites, each with appetite one, placed evenly on a circle with very small radius ε_1. Then, perturb them slightly so that the circular angle between two specific sites s, s' is slightly bigger than the rest: $2\pi/n + \varepsilon_2$, while the angle between any other pair of sites is $2\pi/n - \varepsilon_2/(n-1)$, again for a very small $\varepsilon_2 > 0$. As a result, the standard Voronoi regions of the sites are infinite angular regions, but those of s and s' are slightly wider than those of the remaining sites.

The result is a diagram such as the one in Figure 5, Left. To see this, consider the circle-growing method from [15] described in Section 1. All the sites start growing their region as a circle, which quickly overlap with the other sites. As a result, each site is constrained to grow in the angular region corresponding to its own Voronoi region. Since s and s' have wider angles, they fill their appetite slightly before the rest, which grow at the same rate. After s and s' stop growing, the angular regions "available" to grow for the sites adjacent to s and s' increases to include the Voronoi regions of s and s'. Since now the neighbors of s and s' grow their regions faster than the rest, they also fill their appetite before the remaining sites. This process continues so that, in the end, one half of the sites "wrap" around the circle all the way to the Voronoi region of s, and the other half wrap around to the Voronoi region of s'.

As a result, we get m almost equal-area wedges surrounded by $O(m)$ very thin circular strips. We can choose ε_2 to set exactly how far the stable cell of s and s' reach, and therefore the overall width of all the circular strips. Then, we can set ε_1 small enough to ensure that all the circular strips wrap around the entire circle. Note that, as ε_1 approaches 0, the entire diagram approaches a circle of area m where all the sites want to grow equally far in every angular region.

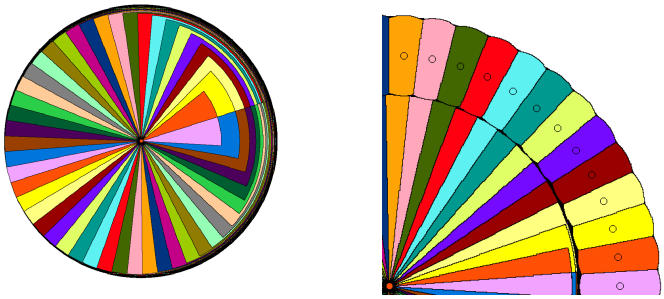


Figure 5 Left: A stable-matching Voronoi diagram of 50 sites situated around a small circle with the perturbed angles. **Right:** Zoom of a section of the stable-matching Voronoi diagram after adding 50 more sites situated evenly along a bigger, concentric circle (and with a smaller ε_2 than in the left figure).

Then, we place the remaining m sites evenly along an exterior concentric circle (see Figure 5, Right), and at a distance so that each outer site will have its appetite satisfied only by cutting into the circular strip. Because the wedges of the sites along the interior circle are very thin, the outer sites are closer to the circular strips than the inner sites; hence, each one can "steal" points in the strips away from the inner sites (which then need to get additional points from beyond the regions of the outer sites). At least half of the strips are as long as half the circle, and hence they get each broken into $\Theta(m)$ faces. Therefore, the circular strips are collectively broken into a total of $\Theta(m^2) = \Theta(n^2)$ faces. ◀

4 Paint-by-Numbers Algorithm

There are two similar incremental approaches to construct the stable matching Voronoi-diagram by processing sites one at a time. One option is to add the region of a site at each step. The other option, which we describe in this section, is to add the bounding disk of a site at each step, partitioned according to the diagram. In both cases, the key is to process the sites by increasing radii of their bounding disks. The major challenge lies in finding the bounding disks, which we will encapsulate in a geometric primitive (Definition 16). We focus on the second approach because the needed geometric primitive is simpler.

Finding the sites by increasing radii of their bounding disks might seem impossible at first, since, when our algorithm starts, the radii of the bounding disks for almost all the sites will, in general, be unknown. Nevertheless, as our algorithm progresses, we can know the radius of the smallest bounding disk of the sites remaining to be processed. Thus, at each iteration, we find a site, s, with smallest-radius bounding disk, B_s, from among the remaining unprocessed sites. Then, reminiscent of a "paint-by-numbers" drawing, we add to the matching all the unmatched points in B_s, using the boundary of B_s and the cells of a standard Voronoi diagram of the remaining unprocessed sites as the guides of how to "color" points. Figure 6 shows two snapshots of the intermediate steps of our paint-by-numbers algorithm. Suppose, then, that we are given a set, S, of n sites in the plane, each with appetite A.

Our algorithm is based on the following result. For a site s, let r_s be the radius of its bounding circle, B_s.

▶ **Lemma 11.** *If $r_s \leq r_{s'}$, no point p inside B_s such that $d(p, s') < d(p, s)$ is matched to s.*

Proof. Any such point p prefers s' to s. Moreover, we have that $d(p, s') < d(p, s) \leq r_s \leq r_{s'}$.

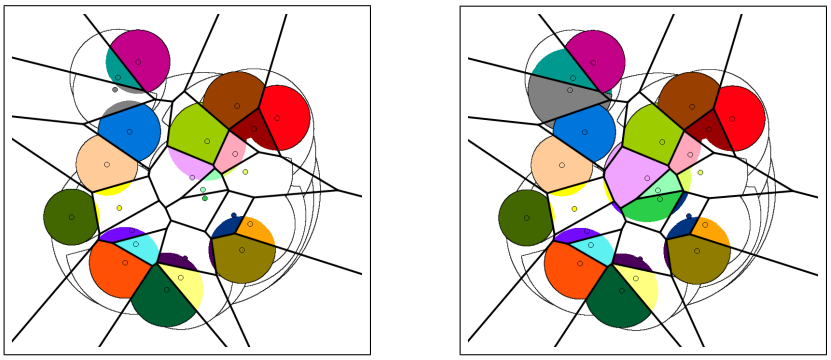


Figure 6 Partial matching done by the incremental paint-by-numbers algorithm after 10 iterations (left) and 12 iterations (right). The sites with the 11th and 12th smallest bounding disks are the gray site in the top-left and the light violet site in the center, respectively. In each case, the edges of the standard Voronoi diagram and the stable-matching Voronoi diagram are overlaid in thick and thin black lines, respectively.

Thus, s' prefers p to the points matched to s' along the boundary of $B_{s'}$. Hence, if p was matched to s, p and s' would be a blocking pair. ◀

▶ **Corollary 12.** *The stable cell of a site s is contained in $V(s)$, the Voronoi cell of s in the standard Voronoi diagram of the subset of sites whose bounding disks have radius at least r_s.*

Proof. The stable cell of s is contained in B_s, but any point in B_s that is not in $V(s)$ is closer to a site other than s whose bounding disk has radius at least r_s. According to Lemma 11, any such point cannot be matched to s. ◀

Our incremental paint-by-numbers algorithm proceeds as follows:

1. Initialize $X = S$ as the set of sites remaining to be processed and $X^* = \emptyset$ as the set of sites that have already been processed. Moreover, for each site s, initialize its partially constructed stable cell C_s as empty, and its remaining appetite, $A_s = A$.
2. Compute a standard Voronoi diagram, V, of the sites in X.
3. Repeat until X is empty:
   a. For each site s in X, calculate r_s, the current estimate for the radius of the bounding disk for s, as follows; r_s is the radius such that the area of the subset of $V(s)$ (the Voronoi cell of s in V) within that radius of s, excluding the bounding disks of sites in X^*, equals A_s.
   b. Choose a site s whose radius r_s is minimum. (By Corollary 12, r_s is the radius of B_s.)
   c. Compute $B_s^* = B_s \setminus \{B_{s'} \mid s' \in X^*\}$, the bounding disk of s minus the already-matched bounding disks.
   d. Match the points in B_s^*: for each site s' in X (including s), add to $C_{s'}$ the (possibly empty) intersection $B_s^* \cap V(s')$, and decrease the appetite, $A_{s'}$, of s', by the area of this intersection.
   e. Move s from X to X^*, and remove s from V.

▶ **Theorem 13.** *The stable-matching Voronoi diagram of a set, S, of n sites can be computed in the real-RAM model in $O(n^3)$ time plus $O(n^2)$ calls to a geometric primitive that has input complexity $O(n)$.*

Let us provide a proof of Theorem 13, beginning with the correctness of our paint-by-numbers algorithm.

▶ **Lemma 14** (Correctness). *The paint-by-numbers algorithm correctly computes the stable-matching Voronoi diagram of a set, S, of n sites.*

Proof. The algorithm matches all the points inside the bounding disks of the sites, and only those. Therefore, it matches precisely the points in the stable-matching Voronoi diagram (Lemma 3). Note that, when finding the site s in X with the smallest bounding disk, we can rely on Corollary 12, which says that the stable cell of s must be contained in $V(s)$.

It remains to be seen that the matching of the points in each bounding disk is correct. Consider the iteration where we process the site s, and let p be a point in B_s^* and in the inside of $V(s')$ (such that $s' \neq s$). The algorithm matches p to s', so we need to see that p belongs to the stable cell of s'. On the one hand, p does not belong to the stable cell of any site in X^* because p lies outside their bounding disks. Moreover, by virtue of being in $V(s')$, p prefers s' to any other site in X. On the other hand, $d(p, s') < d(p, s) \leq r_s \leq r_{s'}$, so s' also prefers p to some of its matched points. Hence, if p and s' were not matched to each other, they would be a blocking pair.

Similarly, in the case where p is in the inside of $V(s)$, the algorithm matches p to s and p belongs in the stable cell of s. ◀

Finding the radii r_s (Step 3a) is the most challenging step in our algorithm. In fact, Observation 15, which we prove in the full version of the paper [6], speaks to its difficulty.

▶ Observation 15. For infinitely-many sets of sites in general position and with algebraic coordinates, the radius of some of the sites' bounding disks cannot be computed exactly in an algebraic model of computation.

To circumvent this problem, we encapsulate the difficulty in computing each r_s analytically in a geometric primitive that can be performed in the real-RAM model or approximated numerically in an algebraic model of computation. That is, for the sake of the algorithm description, we assume the existence of a black-box function that allows us to compute the following geometric primitive.

▶ **Definition 16** (Geometric primitive). Given a convex polygon P, a point s in P, an appetite A, and a set C of disks, return the radius r (if it exists) such that A equals the area of the intersection of $P \setminus C$ and a disk centered at s with radius r.

In the context of our algorithm, the point s is a site in X, the appetite, A, is the remaining appetite, A_s, of s, the polygon P is the Voronoi cell $V(s)$, and the set of disks C is the set of bounding disks of the sites in X^*. Note that such a primitive could be approximated numerically to arbitrary precision, e.g., with a binary search and a simple decision algorithm.

Another detail for our algorithm that we need to spell out is how to maintain the partially constructed cell C_s of each site s. The structure C_s consists of a set of disjoint regions, each of which is delimited by straight and curved edges. Note that the straight edges are part of actual edges of the stable cell of s, as the other side is already matched to some other site. However, the convex curved edges (edges such that the convex face belongs to C_s) found before the iteration of s are not part of edges in the stable cell of s, as they lie along the boundary of the bounding disks of sites other than s (see, e.g., the gray regions in Figure 6, Left). Only at the iteration of s we find the proper convex curved edges of the stable cell of s. The points on the concave face of one of these "fake" convex edges of C_s must also belong to C_s. In some posterior iteration, when the region in the concave face is matched as part of another bounding disk $B_{s'}^*$, the region of $B_{s'}^*$ added to C_s will have a matching concave edge. Thus, when we add a concave edge to C_s, we look for a corresponding convex edge in C_s,

and if we find one, we "glue" them together, by removing the overlapping part. Similarly, we also merge straight edges sharing an endpoint.

Let us turn, then, to the analysis of the running time of the paint-by-numbers algorithm. For the number of calls to the geometric primitive, note that there are n iterations, and at each iteration we call the geometric primitive $O(n)$ times. The Voronoi diagram V has $O(n)$ edges and there are $O(n)$ already-matched disks, so the input of each call has $O(n)$ size. That is, we make $O(n^2)$ calls to the geometric primitive, each of which has combinatorial complexity $O(n)$.

The remaining steps of each iteration all can be done in $O(n^2)$ time, by the following observations:

1. The running time of Step 3b is clearly $O(n)$.
2. The combinatorial complexity of the standard Voronoi diagram, V, is $O(n)$ and it can be initially computed (Step 2) in $O(n \log n)$ time (e.g., see [3, 5]) and updated after the removal of any point site (Step 3e) in $O(n)$ time [13].
3. By the previous observation (2), for any single bounding disk, B_s, chosen in an iteration of our algorithm, B_s has an intersection with V that has combinatorial complexity $O(n)$.
4. The union of the set of bounding disks for the points in X^* has combinatorial complexity $O(n)$ [17, 1].
5. By the previous observation (4), for any single bounding disk, B_s, chosen in an iteration of our algorithm, the combinatorial complexity of B_s^*, that is, B_s minus all the bounding disks for sites in X^*, is $O(n)$. Thus, the running time of Step 3c in any iteration is $O(n)$.
6. Combining the above observations 3 and 5, the combinatorial complexity of the pieces of B_s^* intersecting V has combinatorial complexity $O(n^2)$. Thus, the running time of Step 3d in any iteration is $O(n^2)$.

Therefore, the total running time of our paint-by-numbers algorithm is $O(n^3 + n^2 f(n))$, where $f(n)$ is the running time of the geometric primitive defined above. This completes the proof of Theorem 13.

5 Conclusions

We have studied stable-matching Voronoi diagrams, providing characterizations of their combinatorial complexity and a first discrete algorithm for constructing them. The fact that (*i*) both the standard and stable-matching Voronoi diagrams share the stability property, in terms of preferences based on distances, and that (*ii*) both have similar geometric constructions in terms of the lower envelopes of cones, supports the idea that stable-matching Voronoi diagrams are a natural generalization of the standard Voronoi diagram to sized-constrained regions. However, this comes at the cost of convexity and connectivity; indeed, we have shown that a stable-matching Voronoi diagram may have $O(n^{2+\varepsilon})$ faces and edges, for any $\varepsilon > 0$. It would be interesting to see if this bound can be brought down to $O(n^2)$, which would make it tight. Constructing a stable-matching Voronoi diagram is also more computationally challenging than the construction of a standard Voronoi diagram. We have given a first algorithm which runs in $O(n^3 + n^2 f(n))$-time, where $f(n)$ is the runtime of a geometric primitive that we defined to encapsulate the computations that cannot be carried analytically. While such primitives cannot be avoided, a step forward from our algorithm would be one that relies only in primitives with constant-sized inputs.

References

1. P. K. Agarwal and Micha Sharir. Davenport–Schinzel Sequences and Their Geometric Applications. Technical Report DUKE–TR–1995–21, Duke University, Durham, NC, USA, 1995.
2. Gagan Aggarwal, S. Muthukrishnan, Dávid Pál, and Martin Pál. General auction mechanism for search advertising. In *18th Int. Conf. on the World Wide Web (WWW)*, pages 241–250. ACM, 2009. doi:10.1145/1526709.1526742.
3. Franz Aurenhammer. Voronoi diagrams—A survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991. doi:10.1145/116873.116880.
4. Franz Aurenhammer, Friedrich Hoffmann, and Boris Aronov. Minkowski-type theorems and least-squares clustering. *Algorithmica*, 20(1):61–76, 1998. doi:10.1007/PL00009187.
5. Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, 2013.
6. Gill Barequet, David Eppstein, Michael T. Goodrich, and Nil Mamano. Stable-Matching Voronoi Diagrams: Combinatorial Complexity and Algorithms. Electronic preprint arXiv:1804.09411, 2018.
7. Priyadarshi Bhattacharya and Marina L. Gavrilova. Roadmap-based path planning – Using the Voronoi diagram for a clearance-based shortest path. *IEEE Robotics Automation Magazine*, 15(2):58–66, 2008. doi:10.1109/MRA.2008.921540.
8. Jonathan W. Brandt and V. Ralph Algazi. Continuous skeleton computation by Voronoi diagram. *CVGIP: Image Understanding*, 55(3):329–338, 1992. doi:10.1016/1049-9660(92)90030-7.
9. David Eppstein, Michael T. Goodrich, Doruk Korkmaz, and Nil Mamano. Defining equitable geographic districts in road networks via stable matching. In *25th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*, 2017.
10. David Eppstein, Michael T. Goodrich, and Nil Mamano. Algorithms for stable matching and clustering in a grid. In *18th Int. Workshop on Combinatorial Image Analysis (IWCIA)*, volume 10256 of *LNCS*, pages 117–131. Springer, 2017. doi:10.1007/978-3-319-59108-7_10.
11. Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2(2):153–174, 1987. doi:10.1007/BF01840357.
12. David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962. doi:10.2307/2312726.
13. Ihor G. Gowda, David G. Kirkpatrick, Der Tsai Lee, and Amnon Naamad. Dynamic Voronoi diagrams. *IEEE Transactions on Information Theory*, 29(5):724–731, September 1983. doi:10.1109/TIT.1983.1056738.
14. Dan Gusfield and Robert W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, Cambridge, MA, USA, 1989.
15. Christopher Hoffman, Alexander E. Holroyd, and Yuval Peres. A stable marriage of Poisson and Lebesgue. *Annals of Probability*, 34(4):1241–1272, 2006. doi:10.1214/009117906000000098.
16. Kazuo Iwama and Shuichi Miyazaki. A survey of the stable marriage problem and its variants. In *IEEE Int. Conf. on Informatics Education and Research for Knowledge-Circulating Society (ICKS)*, pages 131–136, 2008. doi:10.1109/ICKS.2008.7.
17. Klara Kedem, Ron Livné, János Pach, and Micha Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.*, 1(1):59–71, 1986. doi:10.1007/BF02187683.
18. Koichi Kise, Akinori Sato, and Motoi Iwata. Segmentation of page images using the area Voronoi diagram. *Computer Vision and Image Understanding*, 70(3):370–382, 1998. doi:10.1006/cviu.1998.0684.

19 Donald E. Knuth. *The Art of Computer Programming, Vol. 3: Sorting and Searching*. Addison-Wesley, Reading, MA, 2nd edition, 1998.

20 Seapahn Meguerdichian, Farinaz Koushanfar, Gang Qu, and Miodrag Potkonjak. Exposure in wireless ad-hoc sensor networks. In *7th Int. Conf. on Mobile Computing and Networking (MobiCom)*, pages 139–150. ACM, 2001. doi:10.1145/381677.381691.

21 National Resident Matching Program, 2017. URL: http://www.nrmp.org.

22 Martin Petřek, Pavlína Košinová, Jaroslav Koča, and Michal Otyepka. MOLE: A Voronoi diagram-based explorer of molecular channels, pores, and tunnels. *Structure*, 15(11):1357–1363, 2007. doi:10.1016/j.str.2007.10.007.

23 Alvin E. Roth and Marilda Sotomayor. The college admissions problem revisited. *Econometrica*, 57(3):559–570, 1989. doi:10.2307/1911052.

24 Ivan Stojmenović, Anand Prakash Ruhil, and D. K. Lobiyal. Voronoi diagram and convex hull based geocasting and routing in wireless networks. *Wireless Communications and Mobile Computing*, 6(2):247–258, 2006. doi:10.1002/wcm.384.

Improved Algorithms for Adaptive Compressed Sensing

Vasileios Nakos
Harvard University, Cambridge, USA
vasileiosnakos@g.harvard.edu

Xiaofei Shi
Carnegie Mellon University, Pittsburgh, USA
xiaofeis@andrew.cmu.edu

David P. Woodruff
Carnegie Mellon University, Pittsburgh, USA
dwoodruf@cs.cmu.edu

Hongyang Zhang
Carnegie Mellon University, Pittsburgh, USA
hongyanz@cs.cmu.edu

Abstract

In the problem of adaptive compressed sensing, one wants to estimate an approximately k-sparse vector $x \in \mathbb{R}^n$ from m linear measurements $A_1x, A_2x, \ldots, A_mx$, where A_i can be chosen based on the outcomes $A_1x, \ldots, A_{i-1}x$ of previous measurements. The goal is to output a vector $\hat{x}$ for which

$$\|x - \hat{x}\|_p \leq C \cdot \min_{k\text{-sparse } x'} \|x - x'\|_q,$$

with probability at least $2/3$, where $C > 0$ is an approximation factor. Indyk, Price and Woodruff (FOCS'11) gave an algorithm for $p = q = 2$ for $C = 1+\epsilon$ with $\mathcal{O}((k/\epsilon)\text{loglog}(n/k))$ measurements and $\mathcal{O}(\log^*(k)\text{loglog}(n))$ rounds of adaptivity. We first improve their bounds, obtaining a scheme with $\mathcal{O}(k \cdot \text{loglog}(n/k) + (k/\epsilon) \cdot \text{loglog}(1/\epsilon))$ measurements and $\mathcal{O}(\log^*(k)\text{loglog}(n))$ rounds, as well as a scheme with $\mathcal{O}((k/\epsilon) \cdot \text{loglog}(n \log(n/k)))$ measurements and an optimal $\mathcal{O}(\text{loglog}(n))$ rounds. We then provide novel adaptive compressed sensing schemes with improved bounds for (p, p) for every $0 < p < 2$. We show that the improvement from $O(k \log(n/k))$ measurements to $O(k \log\log(n/k))$ measurements in the adaptive setting can persist with a better ϵ-dependence for other values of p and q. For example, when $(p, q) = (1, 1)$, we obtain $O(\frac{k}{\sqrt{\epsilon}} \cdot \log\log n \log^3(\frac{1}{\epsilon}))$ measurements. We obtain nearly matching lower bounds, showing our algorithms are close to optimal. Along the way, we also obtain the first nearly-optimal bounds for (p, p) schemes for every $0 < p < 2$ even in the non-adaptive setting.

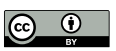
2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Compressed Sensing, Adaptivity, High-Dimensional Vectors

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.90

Related Version A full version of the paper is available at `https://arxiv.org/pdf/1804.09673.pdf`.

Funding This work was partially supported by NSF grant IIS-144741.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 90; pp. 90:1–90:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Compressed sensing, also known as sparse recovery, is a central object of study in data stream algorithms, with applications to monitoring network traffic [7], analysis of genetic data [19, 12], and many other domains [16]. The problem can be stated as recovering an underlying signal $x \in \mathbb{R}^n$ from *measurements* $A_1 \cdot x, ..., A_m \cdot x$ with the C-approximate ℓ_p/ℓ_q recovery guarantee being

$$\|x - \hat{x}\|_p \leq C \min_{k\text{-sparse } x'} \|x - x'\|_q, \tag{1}$$

where the A_i are drawn from a distribution and $m \ll n$. The focus of this work is on *adaptive compressed sensing*, in which the measurements are chosen in rounds, and the choice of measurement in each round depends on the outcome of the measurements in previous rounds.

Adaptive compressed sensing has been studied in a number of different works [11, 4, 8, 9, 14, 1, 10, 18] in theoretical computer science, machine learning, image processing, and many other domains [10, 18, 2]. In theoretical computer science and machine learning, adaptive compressed sensing serves as an important tool to obtain sublinear algorithms for active learning in both time and space [10, 5, 18, 2]. In image processing, the study of adaptive compressed sensing has led to compressed acquisition of sequential images with various applications in celestial navigation and attitude determination [6].

Despite a large amount of works on adaptive compressed sensing, the power of adaptivity remains a long-standing open problem. Indyk, Price, and Woodruff [10] were the first to show that without any assumptions on the signal x, one can obtain a number m of measurements which is a $\log(n)/\log\log(n)$ factor smaller than what can be achieved in the non-adaptive setting. Specifically, for $p = q = 2$ and $C = 1 + \epsilon$, they show that $m = \mathcal{O}(\frac{k}{\epsilon}\log\log(n))$ measurements suffice to achieve guarantee (1), whereas it is known that any non-adaptive scheme requires $k = \Omega(\frac{k}{\epsilon}\log(\frac{n}{k}))$ measurements, provided $\epsilon > \sqrt{\frac{k\log n}{n}}$ (Theorem 4.4 of [17], see also [3]). Improving the sample complexity as much as possible is desired, as it might correspond to, e.g., the amount of radiation a hospital patient is exposed to, or the amont of time a patient must be present for diagnosis.

The ℓ_1/ℓ_1 problem was studied in [17], for which perhaps surprisingly, a better dependence on ϵ was obtained than is possible for ℓ_2/ℓ_2 schemes. Still, the power of adaptivity for the ℓ_1/ℓ_1 recovery problem over its non-adaptive counterpart has remained unclear. An $O(\frac{k}{\sqrt{\epsilon}}\log n\log^3(\frac{1}{\epsilon}))$ non-adaptive bound was shown in [17], while an adaptive lower bound of $\Omega(\frac{k}{\sqrt{\epsilon}}/\log\frac{k}{\sqrt{\epsilon}})$ was shown in [18]. Recently several works [20, 15] have looked at other values of p and q, even those for which $0 < p, q < 1$, which do not correspond to normed spaces. The power of adaptivity for such error measures is also unknown.

1.1 Our Results

Our work studies the problem of adaptive compressed sensing by providing affirmative answers to the above-mentioned open questions. We improve over the best known results for $p = q = 2$, and then provide novel adaptive compressed sensing guarantees for $0 < p = q < 2$ for every p and q. See Table 1 for a comparison of results.

For ℓ_1/ℓ_1, we design an adaptive algorithm which requires only $\mathcal{O}(\frac{k}{\sqrt{\epsilon}}\text{loglog}(n)\log^{\frac{5}{2}}(\frac{1}{\epsilon}))$ measurements for the ℓ_1/ℓ_1 problem. More generally, we study the ℓ_p/ℓ_p problem for $0 < p < 2$. One of our main theorems is the following.

▶ **Theorem 1** (ℓ_p/ℓ_p Recovery Upper Bound). *Let $x \in \mathbb{R}^n$ and $0 < p < 2$. There exists a randomized algorithm that performs $\mathcal{O}(\frac{k}{\epsilon^{p/2}}\text{loglog}(n)\,\text{poly}(\log(\frac{1}{\epsilon})))$ adaptive linear measurements*

Table 1 The sample complexity of adaptive compressed sensing. Results without any citation given correspond to our new results.

C, Guarantees	Upper Bounds	Rounds	Lower Bounds
$1+\epsilon$, ℓ_1/ℓ_1	$\mathcal{O}(\frac{k}{\sqrt{\epsilon}}\text{loglog}(n)\log^{\frac{5}{2}}(\frac{1}{\epsilon}))$	$\mathcal{O}(\text{loglog}(n))$	$\Omega(\frac{k}{\sqrt{\epsilon}\log(k/\sqrt{\epsilon})})$ [18]
$1+\epsilon$, ℓ_p/ℓ_p	$\mathcal{O}(\frac{k}{\epsilon^{p/2}}\text{loglog}(n)\,\text{poly}(\log(\frac{1}{\epsilon})))$	$\mathcal{O}(\text{loglog}(n))$	$\Omega(\frac{k}{\epsilon^{p/2}}\frac{1}{\log^2(k/\epsilon)})$
$\sqrt{\frac{1}{k}}$, ℓ_∞/ℓ_2	$\mathcal{O}(k\text{loglog}(n)+k\log(k))$	$\mathcal{O}(\text{loglog}(n))$	-
$1+\epsilon, \ell_2/\ell_2$	$\mathcal{O}(\frac{k}{\epsilon}\text{loglog}(\frac{n\epsilon}{k}))$ [10] $\mathcal{O}(k\text{loglog}(\frac{n}{k})+\frac{k}{\epsilon}\text{loglog}(\frac{1}{\epsilon}))$ $\mathcal{O}(\frac{k}{\epsilon}\text{loglog}(\frac{n\log(n\epsilon)}{k}))$	$\mathcal{O}(\log^*(k)\text{loglog}(\frac{n\epsilon}{k}))$ [10] $\mathcal{O}(\log^*(k)\text{loglog}(\frac{n}{k}))$ $\mathcal{O}(\text{loglog}(n\log(\frac{n\epsilon}{k}))$	$\Omega(\frac{k}{\epsilon}+\text{loglog}(n))$ [18]

on x in $\mathcal{O}(\text{loglog}(n))$ rounds, and with probability $2/3$, returns a vector $\hat{x}\in\mathbb{R}^n$ such that $\|x-\hat{x}\|_p \le (1+\epsilon)\|x_{-k}\|_p$.

Theorem 1 improves the previous sample complexity upper bound for the case of $C=1+\epsilon$ and $p=q=1$ from $\mathcal{O}(\frac{k}{\sqrt{\epsilon}}\log(n)\log^3(\frac{1}{\epsilon}))$ to $\mathcal{O}(\frac{k}{\sqrt{\epsilon}}\text{loglog}(n)\log^{\frac{5}{2}}(\frac{1}{\epsilon}))$. Compared with the non-adaptive $(1+\epsilon)$-approximate ℓ_1/ℓ_1 upper bound of $\mathcal{O}(\frac{k}{\sqrt{\epsilon}}\log(n)\log^3(\frac{1}{\epsilon}))$, we show that adaptivity exponentially improves the sample complexity w.r.t. the dependence on n over non-adaptive algorithms while retaining the improved dependence on ϵ of non-adaptive algorithms. Furthermore, Theorem 1 extends the working range of adaptive compressed sensing from $p=1$ to general values of $p\in(0,2)$.

We also state a complementary lower bound to formalize the hardness of the above problem.

▶ **Theorem 2** (ℓ_p/ℓ_p Recovery Lower Bound). *Fix $0<p<2$, any $(1+\epsilon)$-approximate ℓ_p/ℓ_p recovery scheme with sufficiently small constant failure probability must make $\Omega(\frac{k}{\epsilon^{p/2}}/\log^2(\frac{k}{\epsilon}))$ measurements.*

Theorem 2 shows that our upper bound in Theorem 1 is tight up to the $\log(k/\epsilon)$ factor.

We also study the case when $p\neq q$. In particular, we focus on the case when $p=\infty, q=2$ and $C=\sqrt{\frac{1}{k}}$, as in the following theorem.

▶ **Theorem 3** (ℓ_∞/ℓ_2 Recovery Upper Bound). *Let $x\in\mathbb{R}^n$. There exists a randomized algorithm that performs $\mathcal{O}(k\log(k)+k\text{loglog}(n))$ linear measurements on x in $\mathcal{O}(\text{loglog}(n))$ rounds, and with probability $1-1/\text{poly}(k)$ returns a vector $\hat{x}$ such that $\|x-\hat{x}\|_\infty^2\le\frac{1}{k}\|x_{-k}\|_2^2$, where $x_{-k}\in\mathbb{R}^n$ is the vector with the largest $n-k$ coordinates (in the sense of absolute value) being zeroed out.*

We also provide an improved result for $(1+\epsilon)$-approximate ℓ_2/ℓ_2 problems.

▶ **Theorem 4** (ℓ_2/ℓ_2 Sparse Recovery Upper Bounds). *Let $x\in\mathbb{R}^n$. There exists a randomized algorithm that*

- *uses $\mathcal{O}(\frac{k}{\epsilon}\text{loglog}(\frac{1}{\epsilon})+k\text{loglog}(\frac{n}{k}))$ linear measurements on x in $\mathcal{O}(\text{loglog}(\frac{n}{k})\cdot\log^*(k))$ rounds;*
- *uses $\mathcal{O}(\frac{k}{\epsilon}\text{loglog}(\frac{n\log(n\epsilon)}{k}))$ linear measurements on x in $\mathcal{O}(\text{loglog}(\epsilon n\log(\frac{n}{k})))$ rounds;*

and with constant probability returns a vector $\hat{x}$ such that $\|x-\hat{x}\|_2\le(1+\epsilon)\|x_{-k}\|_2$.

Previously the best known tradeoff was $\mathcal{O}(\frac{k}{\epsilon}\text{loglog}(\frac{n\epsilon}{k}))$ samples and $\mathcal{O}(\log^*(k)\text{loglog}(\frac{n\epsilon}{k}))$ rounds for $(1+\epsilon)$-approximation for the ℓ_2/ℓ_2 problem [10]. Our result improves both the sample complexity (the first result) and the number of rounds (the second result). We summarize our results in Table 1.

1.2 Our Techniques

ℓ_∞/ℓ_2 Sparse Recovery. Our ℓ_∞/ℓ_2 sparse recovery scheme hashes every $i \in [n]$ to poly(k) buckets, and then proceeds by finding all the buckets that have ℓ_2 mass at least $\Omega(\frac{1}{\sqrt{k}}\|x_{-\Omega(k)}\|_2)$. We then find a set of buckets that contain all heavy coordinates, which are isolated from each other due to hashing. Then, we run a 1-sparse recovery in each bucket in parallel in order to find all the heavy coordinate. However, since we have $\mathcal{O}(k)$ buckets, we cannot afford to take a union bound over all one-sparse recovery routines called. Instead, we show that most buckets succeed and hence we can substract from x the elements returned, and then run a standard CountSketch algorithm to recover everything else. This algorithm obtains an optimal $\mathcal{O}(\text{loglog}(n))$ number of rounds and $\mathcal{O}(k\log(k) + k\text{loglog}(n))$ number of measurements, while succeeding with probability at least $1 - 1/\text{poly}(k)$.

We proceed by showing an algorithm for ℓ_2/ℓ_2 sparse recovery with $\mathcal{O}(\frac{k}{\epsilon}\text{loglog}(n))$ measurements and $\mathcal{O}(\text{loglog}(n))$ rounds. This will be important for our more general ℓ_p/ℓ_p scheme, saving a $\log^*(k)$ factor from the number of rounds, achieving optimality with respect to this quantity. For this scheme, we utilize the ℓ_∞/ℓ_2 scheme we just developed, observing that for small $k < \mathcal{O}(\log(n))$, the measurement complexity is $\mathcal{O}(k\text{loglog}(n))$. The algorithm hashes to $k/(\epsilon\log(n))$ buckets, and in each bucket runs ℓ_∞/ℓ_2 with sparsity k/ϵ. The ℓ_∞/ℓ_2 algorithm in each bucket succeeds with probability $1 - 1/\text{polylog}(n))$; this fact allows us to argue that all but a $1/\text{polylog}(n)$ fraction of the buckets will succeed, and hence we can recover all but a $k/\text{polylog}(n))$ fraction of the heavy coordinates. The next step is to subtract these coordinates from our initial vector, and then run a standard ℓ_2/ℓ_2 algorithm with decreased sparsity.

ℓ_p/ℓ_p Sparse Recovery. Our ℓ_p/ℓ_p scheme, $0 < p < 2$, is based on carefully invoking several ℓ_2/ℓ_2 schemes with different parameters. We focus our discussion on $p = 1$, then mention extensions to general p. A main difficulty of adapting the ℓ_1/ℓ_1 scheme of [17] is that it relies upon an ℓ_∞/ℓ_2 scheme, and all known schemes, including ours, have at least a $k\log k$ dependence on the number of measurements, which is too large for our overall goal.

A key insight in [17] for ℓ_1/ℓ_1 is that since the output does not need to be exactly k-sparse, one can compensate for mistakes on approximating the top k entries of x by accurately outputting enough smaller entries. For example, if $k = 1$, consider two possible signals $x = (1, \epsilon, \ldots, \epsilon)$ and $x' = (1 + \epsilon, \epsilon, \ldots, \epsilon)$, where ϵ occurs $1/\epsilon$ times in both x and x'. One can show, using known lower bound techniques, that distinguishing x from x' requires $\Omega(1/\epsilon)$ measurements. Moreover, $x_1 = (1, 0, \ldots, 0)$ and $x'_1 = (1 + \epsilon, 0, \ldots, 0)$, and any 1-sparse approximation to x or x' must therefore distinguish x from x', and so requires $\Omega(1/\epsilon)$ measurements. An important insight though, is that if one does not require the output signal y to be 1-sparse, then one can output $(1, \epsilon, 0, \ldots, 0)$ in both cases, without actually distinguishing which case one is in!

As another example, suppose that $x = (1, \epsilon, \ldots, \epsilon)$ and $x' = (1 + \epsilon^c, \epsilon, \ldots, \epsilon)$ for some $0 < c < 1$. In this case, one can show that one needs $\Omega(1/\epsilon^c)$ measurements to distinguish x and x', and as before, to output an exactly 1-sparse signal providing a $(1 + \epsilon)$-approximation requires $\tilde{\Theta}(1/\epsilon^c)$ measurements. In this case if one outputs a signal y with $y_1 = 1$, one cannot simply find a single other coordinate ϵ to "make up" for the poor approximation on the first coordinate. However, if one were to output $1/\epsilon^{1-c}$ coordinates each of value ϵ, then the ϵ^c "mass" lost by poorly approximating the first coordinate would be compensated for by outputting $\epsilon \cdot 1/\epsilon^{1-c} = \epsilon^c$ mass on these remaining coordinates. It is not clear how to find such remaining coordinates though, since they are much smaller; however, if one randomly subsamples an ϵ^c fraction of coordinates, then roughly $1/\epsilon^{1-c}$ of the coordinates of value ϵ

survive and these could all be found with a number of measurements proportional to $1/\epsilon^{1-c}$. Balancing the two measurement complexities of $1/\epsilon^c$ and $1/\epsilon^{1-c}$ at $c = 1/2$ gives roughly the optimal $1/\epsilon^{1/2}$ dependence on ϵ in the number of measurements.

To extend this to the adaptive case, a recurring theme of the above examples is that the top k, while they need to be found, they do not need to be approximated very accurately. Indeed, they do need to be found, if, e.g., the top k entries of x were equal to an arbitrarily large value and the remaining entries were much smaller. We accomplish this by running an ℓ_2/ℓ_2 scheme with parameters $k' = \Theta(k)$ and $\epsilon' = \Theta(\sqrt{\epsilon})$, as well as an ℓ_2/ℓ_2 scheme with parameters $k' = \Theta(k/\sqrt{\epsilon})$ and $\epsilon' = \Theta(1)$ (up to logarithmic factors in $1/\epsilon$). Another theme is that the mass in the smaller coordinates we find to compensate for our poor approximation in the larger coordinates also does not need to be approximated very well, and we find this mass by subsampling many times and running an ℓ_2/ℓ_2 scheme with parameters $k' = \Theta(1)$ and $\epsilon' = \Theta(1)$. This technique is surprisingly general, and does not require the underlying error measure we are approximating to be a norm. It just uses scale-invariance and how its rate of growth compares to that of the ℓ_2-norm.

ℓ_2/ℓ_2 Sparse Recovery. Our last algorithm, which concerns ℓ_2/ℓ_2 sparse recovery, achieves $\mathcal{O}(k\text{loglog}(n) + \frac{k}{\epsilon}\text{loglog}(1/\epsilon))$ measurements, showing that ϵ does not need to multiply loglog(n). The key insight lies in first solving the 1-sparse recovery task with $\mathcal{O}(\text{loglog}(n) + \frac{1}{\epsilon}\text{loglog}(1/\epsilon))$ measurements, and then extending this to the general case. To achieve this, we hash to polylog$(1/\epsilon)$ buckets, then solve ℓ_2/ℓ_2 with constant sparsity on a new vector, where coordinate j equals the ℓ_2 norm of the jth bucket; this steps requires only $\mathcal{O}(\frac{1}{\epsilon}\text{loglog}(1/\epsilon))$ measurements. Now, we can run standard 1-sparse recovery in each of these buckets returned. Extending this idea to the general case follows by plugging this sub-routine in the iterative algorithm of [10], while ensuring that sub-sampling does not increase the number of measurements. For that we also need to sub-sample at a slower rate, slower roughly by a factor of ϵ.

Notation: For a vector $x \in \mathbb{R}^n$, we define $H_k(x)$ to be the set of its largest k coordinates in absolute value. For a set S, denote by x_S the vector with every coordinate $i \notin S$ being zeroed out. We also define $x_{-k} = x_{[n]\setminus H_k(x)}$ and $H_{k,\epsilon}(x) = \{i \in [n] : |x_i| \geq \frac{\epsilon}{k}\|x_{-k}\|_2^2\}$, where $[n]$ represents the set $\{1, 2, ..., n\}$. For a set S, let $|S|$ be the cardinality of S.

Due to space constraints, we defer the proof of Theorem 2 to the full version[1].

2 Adaptive ℓ_p/ℓ_p Recovery

This section is devoted to proving Theorem 1. Our algorithm for ℓ_p/ℓ_p recovery is in Algorithm 1.

Let $f = \epsilon^{p/2}$, $r = 2/(p\log(1/f))$ and $q = \max\{p - \frac{1}{2}, 0\} = (p - \frac{1}{2})^+$. We will invoke the following ℓ_2/ℓ_2 oracle frequently throughout the paper.

▶ **Oracle 1** (AdaptiveSparseRecovery$_{\ell_p/\ell_q}(x, k, \epsilon)$). *The oracle is fed with (x, k, ϵ) as input parameters, and outputs a set of coordinates $i \in [n]$ of size $\mathcal{O}(k)$ which corresponds to the support of vector $\hat{x}$, where $\hat{x}$ can be any vector for which $\|x-\hat{x}\|_p \leq (1+\epsilon)\min_{\mathcal{O}(k)\text{-sparse } x'} \|x-x'\|_q$.*

[1] see https://arxiv.org/pdf/1804.09673.pdf

Algorithm 1 Adaptive ℓ_p/ℓ_p Recovery.

1. $A \leftarrow \text{AdaptiveSparseRecovery}_{\ell_2/\ell_2}(x, 2k/f, 1/10)$.
2. $B \leftarrow \text{AdaptiveSparseRecovery}_{\ell_2/\ell_2}(x, 4k, f/r^2)$.
3. $S \leftarrow A \cup B$.
4. For $j = 1 : r$
5. Uniformly sample the entries of x with probability $2^{-j}f/k$ for $k/(2f(r+1)^q)$ times.
6. Run the adaptive $\text{AdaptiveSparseRecovery}_{\ell_2/\ell_2}(x, 2, 1/(4(r+1))^{\frac{2}{p}})$ algorithm on each of the $k/(2f(r+1)^q)$ subsamples to obtain sets $A_{j,1}, A_{j,2}, \ldots, A_{j,k/(2f(r+1)^q)}$.
7. Let $S_j \leftarrow \cup_{t=1}^{k/(2f(r+1)^q)} A_{j,t} \setminus \cup_{t=0}^{j-1} S_t$.
8. End For
9. Request the entries of x with coordinates $S_0, ..., S_r$.
Output: $\hat{x} = x_{S_0 \cup \cdots \cup S_r}$.

Existing algorithms can be applied to construct Oracle 1 for the ℓ_2/ℓ_2 case, such as [10]. Without loss of generality, we assume that the coordinates of x are ranked in decreasing value, i.e., $x_1 \geq x_2 \geq \cdots \geq x_n$.

▶ **Lemma 5.** *Suppose we subsample x with probability p and let y be the subsampled vector formed from x. Then with failure probability $e^{-\Omega(k)}$, $\|y_{-2k}\|_2 \leq \sqrt{2p}\left\|x_{-k/p}\right\|_2$.*

Proof. Let T be the set of coordinates in the subsample. Then $\mathbb{E}\left[\left|T \cap \left[\frac{3k}{2p}\right]\right|\right] = \frac{3k}{2}$. So by the Chernoff bound, $\Pr\left[\left|T \cap \left[\frac{3k}{2p}\right]\right| > 2k\right] \leq e^{-\Omega(k)}$. Thus $\left|T \cap \left[\frac{3k}{2p}\right]\right| \leq 2k$ holds with high probability. Let $Y_i = x_i^2$ if $i \in T$ $Y_i = 0$ if $i \in [n] \setminus T$. Then $\mathbb{E}\left[\sum_{i > \frac{3k}{2p}} Y_i\right] = p\left\|x_{-\frac{3k}{2p}}\right\|_2^2 \leq p\left\|x_{-k/p}\right\|_2^2$. Notice that there are at least $\frac{k}{2p}$ elements in $x_{-k/p}$ with absolute value larger than $\left|x_{\frac{3k}{2p}}\right|$. Thus for $i > \frac{3k}{2p}$, $Y_i \leq \left|x_{\frac{3k}{2p}}\right|^2 \leq \frac{2p}{k}\left\|x_{-k/p}\right\|_2^2$. Again by a Chernoff bound, $\Pr\left[\sum_{i > \frac{3k}{2p}} Y_i \geq \frac{4p}{3}\left\|x_{-k/p}\right\|_2^2\right] \leq e^{-\Omega(k)}$. Conditioned on the latter event not happening, $\|y_{-2k}\|_2^2 \leq \sum_{i > \frac{3k}{2p}} Y_i \leq \frac{4p}{3}\left\|x_{-k/p}\right\|_2^2 \leq 2p\left\|x_{-k/p}\right\|_2^2$. By a union bound, with failure probability $e^{-\Omega(k)}$, we have $\|y_{-2k}\|_2 \leq \sqrt{2p}\left\|x_{-k/p}\right\|_2$. ◀

▶ **Lemma 6.** *Let $\hat{x}$ be the output of the ℓ_2/ℓ_2 scheme on x with parameters $(k, \epsilon/2)$. Then with small constant failure probability, $\left\|x_{[k]}\right\|_p^p - \|\hat{x}\|_p^p \leq k^{1-\frac{p}{2}}\epsilon^{\frac{p}{2}}\|x_{-k}\|_2^p$.*

Proof. Notice that with small constant failure probability, the ℓ_2/ℓ_2 guarantee holds and we have

$$\left\|x_{[k]}\right\|_2^2 - \|\hat{x}\|_2^2 = \|x - \hat{x}\|_2^2 - \|x_{-k}\|_2^2 \leq (1+\epsilon)\|x_{-k}\|_2^2 - \|x_{-k}\|_2^2 = \epsilon\|x_{-k}\|_2^2.$$

Let $S \subset [n]$ be such that $x_S = \hat{x}$, and define $y = x_{[k]\setminus S}$, $z = x_{S\setminus[k]}$. Then if $\|y\|_p^p \leq k^{1-\frac{p}{2}}\epsilon^{\frac{p}{2}}\|x_{-k}\|_2^p$ we are done. Otherwise, let $1 \leq k' \leq k$ denote the size of $[k] \setminus S$, and define $c = \|y\|_2/\sqrt{k'}$.

$$\begin{aligned}\left\|x_{[k]}\right\|_p^p - \|\hat{x}\|_p^p = \|y\|_p^p - \|z\|_p^p &\leq k'^{1-\frac{p}{2}}\|y\|_2^p - \|z\|_p^p = \frac{\|y\|_2^2}{c^{2-p}} - \|z\|_p^p \\ &\leq \frac{\|y\|_2^2 - \|z\|_2^2}{c^{2-p}} = \frac{\left\|x_{[k]}\right\|_2^2 - \|\hat{x}\|_2^2}{c^{2-p}} \leq \frac{\epsilon\|x_{-k}\|_2^2}{c^{2-p}}.\end{aligned}$$

Since $c \geq \frac{\|y\|_p}{k'^{\frac{1}{p}}} \geq \frac{\|y\|_p}{k^{\frac{1}{p}}} \geq \sqrt{\frac{\epsilon}{k}} \|x_{-k}\|_2$, we have $\left\|x_{[k]}\right\|_p^p - \|\hat{x}\|_p^p \leq k^{\frac{2-p}{2}} \epsilon^{1-\frac{2-p}{2}} \|x_{-k}\|_2^{2-(2-p)} = k^{1-\frac{p}{2}} \epsilon^{\frac{p}{2}} \|x_{-k}\|_2^p$. ◀

▶ **Theorem 7.** *Fix $0 < p < 2$. For $x \in \mathbb{R}^n$, there exists a $(1+\epsilon)$-approximation algorithm that performs $\mathcal{O}(\frac{k}{\epsilon^{p/2}} \text{loglog}(n) \log^{\frac{2}{p}+1-(p-\frac{1}{2})^+}(\frac{1}{\epsilon}))$ adaptive linear measurements in $\mathcal{O}(\text{loglog}(n))$ rounds, and with probability at least $2/3$, we can find a vector $\hat{x} \in \mathbb{R}^n$ such that*

$$\|x - \hat{x}\|_p \leq (1+\epsilon) \|x_{-k}\|_p . \tag{2}$$

Proof. The algorithm is stated in Algorithm 1. We first consider the difference $\left\|x_{[k]}\right\|_p^p - \|x_{S_0}\|_p^p$.

Let $i^*(0)$ be the smallest integer such that for any $l > i^*(0)$, $|x_l| \leq \|x_{-2k/f}\|_2/\sqrt{k}$.

Case 1. $i^*(0) > 4k$

Then for all $k < j \leq 4k$, we have $|x_j| > \|x_{-2k/f}\|_2/\sqrt{k}$. Hence x_{S_0} must contain at least $1/2$ of these indices; if not, the total squared loss is at least $1/2 \cdot 3k \|x_{-2k/f}\|_2^2/k \geq (3/2)\|x_{-2k/f}\|_2^2$, a contradiction to $\epsilon' = 1/10$. It follows that $\|x_{S_0 \cap \{k+1,\ldots,4k\}}\|_p^p \geq \frac{3}{2}k \left[\frac{\|x_{-2k/f}\|_2}{\sqrt{k}}\right]^p = \frac{3}{2}k^{1-\frac{p}{2}} \|x_{-2k/f}\|_2^p$. On the other hand, $\left\|x_{[k]}\right\|_p^p - \|x_{S_0}\|_p^p$ is at most $1.1k^{1-\frac{p}{2}} \|x_{-2k/f}\|_2^p$, since by the ℓ_2/ℓ_2 guarantee

$$\|x_{[k]}\|_p^p - \|x_{S_0 \cap [k]}\|_p^p \leq k^{1-\frac{p}{2}} \|x_{[k]} - x_{S_0 \cap [k]}\|_2^p \leq k^{1-\frac{p}{2}} \|x - x_{S_0}\|_2^p \leq \frac{11}{10} k^{1-\frac{p}{2}} \|x_{-2k/f}\|_2^p.$$

It follows that

$$\begin{aligned} \|x_{[k]}\|_p^p - \|x_{S_0}\|_p^p &= \|x_{[k]}\|_p^p - \|x_{S_0 \cap [k]}\|_p^p - \|x_{S_0 \cap \{k+1,\ldots,4k\}}\|_p^p \\ &\leq \frac{11}{10} k^{1-\frac{p}{2}} \|x_{-2k/f}\|_2^p - \frac{3}{2} k^{1-\frac{p}{2}} \|x_{-2k/f}\|_2^p \leq 0. \end{aligned}$$

Case 2. $i^*(0) \leq 4k$, and $\sum_{j=i^*(0)+1}^{2k/f} x_j^2 \geq 4\|x_{-2k/f}\|_2^2$.

We claim that x_{S_0} must contain at least a $5/8$ fraction of coordinates in $\{i^*(0)+1, \ldots, 2k/f\}$; if not, then the cost for missing at least a $3/8$ fraction of the ℓ_2-norm of $x_{\{i^*(0)+1,\ldots,2k/f\}}$ will be at least $(3/2)\|x_{-2k/f}\|_2^2$, contradicting the ℓ_2/ℓ_2 guarantee. Since all coordinates x_j's for $j > i^*(0)$ have value at most $\|x_{-2k/f}\|_2/\sqrt{k}$, it follows that the p-norm of coordinates corresponding to $\{i^*(0)+1, \ldots, 2k/f\} \cap S_0$ is at least $\left\|x_{\{i^*(0)+1,\ldots,2k/f\} \cap S_0}\right\|_p^p \geq \frac{5}{2} k^{\frac{2-p}{2}} \frac{\|x_{-2k/f}\|_2^2}{\|x_{-2k/f}\|_2^{2-p}} = \frac{5}{2} k^{1-\frac{p}{2}} \|x_{-2k/f}\|_2^p$. Then

$$\begin{aligned} \|x_{[k]}\|_p^p - \|x_{S_0}\|_p^p &\leq \frac{11}{10} k^{1-\frac{p}{2}} \|x_{-2k/f}\|_2^p + k \left(\frac{\|x_{-2k/f}\|_2}{\sqrt{k}}\right)^p - \|x_{\{i^*(0)+1,\ldots,2k/f\} \cap S_0}\|_p^p \\ &\leq \frac{21}{10} k^{1-\frac{p}{2}} \|x_{-2k/f}\|_2^p - \frac{5}{2} k^{1-\frac{p}{2}} \|x_{-2k/f}\|_2^p \leq 0. \end{aligned}$$

Case 3. $i^*(0) \leq 4k$, and $\sum_{j=i^*(0)+1}^{2k/f} x_j^2 \leq 4\|x_{-2k/f}\|_2^2$.

With a little abuse of notation, let x_{S_0} denote the output of the ℓ_2/ℓ_2 with parameters $(4k, f/r^2)$. Notice that there are at most $8k$ non-zero elements in x_{S_0}, and $\|x_{-4k}\|_2^2 \leq \|x_{-i^*(0)}\|_2^2 = \sum_{j=i^*(0)+1}^{2k/f} x_j^2 + \|x_{-2k/f}\|_2^2 \leq 5\|x_{-2k/f}\|_2^2$. By Lemma 6, we have $\left\|x_{[k]}\right\|_p^p - \|x_{S_0}\|_p^p \leq \left\|x_{[4k]}\right\|_p^p - \|x_{S_0}\|_p^p \leq (4k)^{1-\frac{p}{2}} \frac{f^{\frac{p}{2}}}{r^p} \|x_{-4k}\|_2^p \leq \mathcal{O}\left(\frac{1}{r^p}\right) k^{1-\frac{p}{2}} f^{\frac{p}{2}} \|x_{-2k/f}\|_2^p$. According to the above three cases, we conclude that $\|x_{[k]}\|_p^p - \|x_{S_0}\|_p^p \leq \mathcal{O}\left(\frac{1}{r^p}\right) k^{1-\frac{p}{2}} f^{\frac{p}{2}} \|x_{-2k/f}\|_2^p$. Thus with failure probability at most $1/6$,

$$\|x - \hat{x}\|_p^p - \|x_{-k}\|_p^p = \|x_{[k]}\|_p^p - \sum_{j=0}^{r} \|x_{S_j}\|_p^p \leq \mathcal{O}\left(\frac{1}{r^p}\right) k^{1-\frac{p}{2}} f^{\frac{p}{2}} \|x_{-2k/f}\|_2^p - \sum_{j=1}^{r} \left\|x_{S_j}\right\|_p^p . \tag{3}$$

In order to convert the first term on the right hand side of (3) to a term related to the ℓ_p norm (which is a semi-norm if $0 < p < 1$), we need the following inequalities: for every u and s, by splitting into chunks of size s, we have

$$s^{1-\frac{p}{2}} \|u_{-2s}\|_2^p \le \|u_{-s}\|_p^p, \qquad \text{and} \qquad \left\|u_{\overline{[s]}\cap[2s]}\right\|_2 \le \sqrt{s}\,|u_s|\,.$$

Define $c = (r+1)^{\min\{p,1\}}$. This gives us that, for $0 < p < 2$ $\frac{1}{(r+1)^p} k^{1-\frac{p}{2}} f^{\frac{p}{2}} \left\|x_{-2k/f}\right\|_2^p \le \frac{k^{1-\frac{p}{2}} f^{\frac{p}{2}}}{c} \left\|x_{-2k/f^{1+\frac{2}{p}}}\right\|_2^p + \frac{k^{1-\frac{p}{2}} f^{\frac{p}{2}}}{c} \sum_{j=1}^{r} \left\|x_{\overline{[2^j k/f]}\cap[2^{j+1}k/f]}\right\|_2^p \le \frac{f^{(1-\frac{p}{2})(1+\frac{2}{p})+\frac{p}{2}}}{c} \left\|x_{-k/f^{1+\frac{2}{p}}}\right\|_p^p + \frac{1}{c}\sum_{j=1}^{r} k 2^{pj/2} \left|x_{2^j k/f}\right|^p$. Therefore,

$$\begin{aligned}\|\hat{x}-x\|_p^p - \|x_{-k}\|_p^p &\le \mathcal{O}\left(\frac{1}{c}\right) f^{\frac{2}{p}} \left\|x_{-k/f^{1+\frac{2}{p}}}\right\|_p^p + \sum_{j=1}^{r} \mathcal{O}\left(\frac{1}{c}\right) k 2^{pj/2} |x_{2^j k/f}|^p - \sum_{j=1}^{r} \|x_{S_j}\|_p^p \\ &\le \mathcal{O}\left(\frac{1}{c}\right) f^{\frac{2}{p}} \left\|x_{-k/f}\right\|_p^p + \sum_{j=1}^{r} \mathcal{O}\left(\frac{1}{c}\right) k 2^{pj/2} |x_{2^j k/f}|^p - \sum_{j=1}^{r} \|x_{S_j}\|_p^p. \end{aligned} \tag{4}$$

Let $y = x_T$ denote an independent subsample of x with probability $f/(2^j k)$, and $\hat{y}$ be the output of the ℓ_2/ℓ_2 algorithm with parameter $\mathrm{s}(2, 1/(4(r+1))^{\frac{2}{p}})$. Notice that $|S_j| \le 2k/(r+1)f$ by the adaptive ℓ_2/ℓ_2 guarantee. Define $Q = [2^j k/f] \setminus (S_0 \cup \cdots \cup S_{j-1})$. There are at least $2^j k/(2f)$ elements in Q, and every element in Q has absolute value at least $\left|x_{2^j k/f}\right|$. In each subsample, notice that $\mathbb{E}[|T \cap Q|] = \frac{1}{2}$. Thus with sufficiently small constant failure probability there exists at least 1 element in y with absolute value at least $|x_{2^j k/f}|$. On the other hand, by Lemma 6 and Lemma 5,

$$\left\|y_{[1]}\right\|_p^p - \|\hat{y}\|_p^p \le \left\|y_{[2]}\right\|_p^p - \|\hat{y}\|_p^p \le \frac{2^{1-\frac{p}{2}}}{4(r+1)} \|y_{-2}\|_2^p \le \frac{1}{2(r+1)} \left(\frac{f}{2^j k}\right)^{\frac{p}{2}} \left\|x_{-2^j k/f}\right\|_2^p, \tag{5}$$

with sufficiently small constant failure probability given by the union bound. For the $k/(2f(r+1)^q)$ independent copies of subsamples, by a Chernoff bound, a $1/4$ fraction of them will have the largest absolute value in Q and (5) will also hold, with the overall failure probability being $e^{-\Omega(k/(fr^q))}$. Therefore, since $k/f > 2^{pj/2}k$,

$$\begin{aligned}\left\|x_{S_j}\right\|_p^p &\ge \frac{2^{pj/2}k}{8(r+1)^q} \left[\left|x_{2^j k/f}\right|^p - \frac{1}{2(r+1)} \left(\frac{f}{2^j k}\right)^{\frac{p}{2}} \left\|x_{-2^j k/f}\right\|_2^p\right] \\ &\ge \frac{2^{pj/2}k}{8(r+1)^q} \left|x_{2^j k/f}\right|^p - \frac{k^{1-\frac{p}{2}} f^{\frac{p}{2}}}{16(r+1)^{q+1}} \left\|x_{-2k/f}\right\|_2^p, \end{aligned}$$

and by the fact that $0 < q < p < 2$,

$$\begin{aligned}\|x-\hat{x}\|_p^p - \|x_{-k}\|_p^p &\le \mathcal{O}(\frac{1}{r^p}) k^{1-\frac{p}{2}} f^{\frac{p}{2}} \|x_{-2k/f}\|_2^p - \sum_{j=1}^{r} \left\|x_{S_j}\right\|_p^p \\ &\le \left[\mathcal{O}\left(\frac{1}{r^p}\right) + \frac{r}{16(r+1)^{q+1}}\right] k^{1-\frac{p}{2}} f^{\frac{p}{2}} \|x_{-2k/f}\|_2^p - \sum_{j=1}^{r} \frac{2^{pj/2}k}{8(r+1)^q} \left|x_{2^j k/f}\right|^p \\ &\le \mathcal{O}\left(\frac{1}{c}\right) f^{\frac{2}{p}} \left\|x_{-k/f}\right\|_p^p + \left[\mathcal{O}\left(\frac{1}{c}\right) + \frac{1}{16(r+1)^q} - \frac{1}{8(r+1)^q}\right] \sum_{j=1}^{r} k 2^{pj/2} \left|x_{2^j k/f}\right|^p \\ &\le f^{\frac{2}{p}} \left\|x_{-k/f}\right\|_p^p \le \epsilon \|x_{-k}\|_p^p\,. \end{aligned}$$

The total number of measurements will be at most

$$\mathcal{O}\left(\frac{k}{f}\mathrm{loglog}(n) + \frac{4kr^2}{f}\mathrm{loglog}(n) + \frac{kr}{2fr^q} r^{\frac{2}{p}} \mathrm{loglog}(n)\right) = \mathcal{O}\left(\frac{k}{\epsilon^{\frac{p}{2}}} \mathrm{loglog}(n) \log^{\frac{2}{p}+1-(p-\frac{1}{2})^+}\left(\frac{1}{\epsilon}\right)\right),$$

while the total failure probability given by the union bound is $1/6 + e^{-\Omega(k/(fr^q))} < 1/3$, which completes the proof. ◀

3 ℓ_∞/ℓ_2 Adaptive Sparse Recovery

In this section, we will prove Theorem 3. Our algorithm first approximates $\|x_{-k}\|_2$. The goal is to compute a value V which is not much smaller than $\frac{1}{k}\|x_{-k}\|_2^2$, and also at least $\Omega(\frac{1}{k})\|x_{-\Omega(k)}\|_2^2$. This value will be used to filter out coordinates that are not large enough, while ensuring that heavy coordinates are included. We need the following lemma, which for example can be found in Section 4 of [13].

▶ **Lemma 8.** *Using* $\log(1/\delta)$ *non-adaptive measurements we can find with probability* $1-\delta$ *a value* V *such that* $\frac{1}{C_1 k}\|x_{-C_2 k}\|_2^2 \le V \le \frac{1}{k}\|x_{-k}\|_2^2$, *where* C_1, C_2 *are absolute constants larger than* 1.

We use the aforementioned lemma with $\Theta(\log k)$ measuremenents to obtain such a value V with probability $1-1/\text{poly}(k)$. Now let c be an absolute constant and let $g : [n] \to [k^c]$ be a random hash function. Then, with probability at least $1 - \frac{1}{\text{poly}(k)}$ we have that for every $i, j \in H_k(x)$, $g(i) \neq g(j)$. By running PartitionCountSketch $(x, 2C_1 k, \{g^{-1}(1), g^{-1}(2), \ldots, g^{-1}(k^c)\}$, we get back an estimate w_j for every $j \in [k^c]$; here C_1 is an absolute constant. Let γ' be an absolute constant to be chosen later. We set $S = \{j \in [k^c] : w_j^2 \ge \gamma' V\}$ and $T = \bigcup_{j \in S} g^{-1}(j)$. We prove the following lemma.

▶ **Lemma 9.** *Let* C' *be an absolute constant. With probability at least* $1 - 1/\text{poly}(k)$ *the following holds.*

1. $|S| = \mathcal{O}(k)$.
2. *Every* $j \in [k^c]$ *such that there exists* $i \in H_k(x) \cap g^{-1}(j)$, *will be present in* S.
3. *For every* $j \in S$, *there exists exactly one coordinate* $i \in g^{-1}(j)$ *with* $x_i^2 \ge \frac{1}{C' k}\|x_{-C_2 k}\|_2^2$.
4. *For every* $j \in S$, $\|x_{g^{-1}(j) \setminus H_k(x)}\|_2^2 \le \frac{1}{k^2}\|x_{-k}\|_2^2$.

Proof. Let C_0 be an absolute constant larger than 1. Note that with probability $1 - C_0^2 \cdot k^{6-c}$, all $i \in H_{C_0 k^3}(x)$ (and, hence, also in $H_{C_0 k^3, 1/k^3}(x)$) are isolated under g. Fix $j \in [k^c]$ and, for $i \in [n]$, define the random variable $Y_i = 1_{g(x_i)=j} x_i^2$. Now observe that

$$\mathbb{E}\left[\sum_{i \in g^{-1}(j) \setminus H_{C_0 k^3, 1/k^3}(x)} Y_i\right] = \frac{1}{k^c}\|x_{-C_0 k^3}\|_2^2.$$

Applying Bernstein's inequality to the variables Y_i with

$$K = \frac{1}{C_0 k^3}\|x_{-C_0 k^3}\|_2^2, \qquad \text{and} \qquad \sigma^2 < \frac{1}{k^{c+3}}\|x_{-C_0 k^3}\|_2^4,$$

we have that

$$\Pr\left[\sum_{i \in g^{-1}(j) \setminus H_{C_0 k^3, 1/k^3}(x)} x_i^2 \ge 1/k^2 \|x_{-C_0 k^2}\|_2^2\right] \le e^{-k},$$

where c is an absolute constant. This allows us to conclude that the above statement holds for all different k^c possible values j, by a union-bound. We now prove the bullets one by one. We remind the reader that PartitionCountSketch aproximates the value of every $\|x_{g^{-1}(j)}\|_2^2$ with a multiplicate error in $[1-\gamma, 1+\gamma]$ and additive error $\frac{1}{C_0 k}\|x_{-k}\|_2^2$.

1. Since there are at most $\frac{1}{\gamma'(1+\gamma)}C_2k + C_2k$ indices j with $(1+\gamma)\|x_{g^{-1}(j)}\|_2^2 \geq \frac{\gamma'}{k}\|x_{-k}\|_2^2 \geq \gamma' V$, the algorithm can output at most $\mathcal{O}(k)$ indices.
2. The estimate for such a j will be at least $(1-\gamma)\frac{1}{k}\|x_{-k}\|_2^2 - \frac{1}{2C_1k}\|x_{-C_2k}\|_2^2 \geq \gamma' V$, for some suitable choice of γ'. This implies that j will be included in S.
3. Because of the guarantee for V and the guarantee of PartitionCountSketch, we have that all j that are in S satisfy $(1+\gamma)\|x_{g^{-1}(j)}\|_2^2 + \frac{1}{k}\|x_{-2C_1k}\|_2^2 \geq \frac{\gamma'}{k}\|x_{-C_2k}\|_2^2$, and since

$$\sum_{i \in g^{-1}(j) \setminus H_{C_0k^3}(x)} x_i^2 \leq \frac{1}{k^2}\|x_{-k}\|_2^2,$$

   this implies that there exists $i \in H_{C_0k^3}(x) \cap g^{-1}(j)$. But since all $i \in H_{C_0k^3}(x)$ are perfectly hashed under g, this implies that this i should satisfy $x_i^2 \geq \frac{1}{C'k}\|x_{-C_2k}\|_2^2$, from which the claim follows.
4. Because elements in $H_{C_0k^3}(x)$ are perfectly hashed, we have that

$$\|x_{g^{-1}(j)\setminus H_k(x)}\|_2^2 = \|x_{g^{-1}(j)\setminus H_{C_0k^3}}(x)\|_2^2 \leq \frac{1}{k^2}\|x_{-k}\|_2^2$$

   for C_0 large enough. ◀

Given S, we proceed in the following way. For every $j \in S$, we run the algorithm guaranteed by Lemma 15 from the full version [2] to obtain an index i_j, using $\mathcal{O}(k\text{loglog}n)$ measurements. Then we observe directly x_{i_j} using another $\mathcal{O}(k)$ measurements, and form vector $z = x - x_{\{i_j\}_{j\in S}}$. We need the following lemma.

▶ **Lemma 10.** *With probability* $1 - 1/\text{poly}(k)$, $|H_k(x) \setminus \{i_j\}_{j\in S}| \leq \frac{k}{\log^2 n}$.

Proof. Let us consider the calls to the 1-sparse recovery routine in j for which there exists $i \in H_k(x) \cap g^{-1}(j)$. Since the 1-sparse recovery routine succeeds with probability $1 - 1/\text{poly}(\log n)$, then the probability that we have more than $\frac{k}{\log^2 n}$ calls that fail, is

$$\binom{k}{\frac{k}{\log^2 n}}\left(\frac{1}{\text{poly}(\log n)}\right)^{k/\log^2 n} \leq \frac{1}{\text{poly}(k)}.$$

This gives the proof of the lemma. ◀

For the last step of our algorithm, we run PartitionCountSketch$(z_T, k/\log(n), [n])$ to estimate the entries of z. We then find the coordinates with the largest $2k$ estimates, and observe them directly. Since

$$\frac{\log n}{k}\|(z_T)_{-k/\log n}\|_2^2 \leq \frac{\log n}{k} \cdot \frac{1}{k^2}\|x_{-k}\|_2^2 = \frac{\log n}{k^3}\|x_{-k}\|_2^2,$$

every coordinate will be estimated up to additive error $\frac{\log n}{k^3}\|x_{-k}\|_2^2$, which shows that every coordinate in $T \cap H_{k,1/k}(x)$ will be included in the top $2k$ coordinates. Putting everything together, we obtain the desired result.

[2] see https://arxiv.org/pdf/1804.09673.pdf

4 ℓ_2/ℓ_2 Adaptive Sparse Recovery in Optimal Rounds

In this section, we give an algorithm for ℓ_2/ℓ_2 compressed sensing using $\mathcal{O}(\text{loglog}n)$ rounds, instead of $\mathcal{O}(\log^* k \cdot \text{loglog}n)$ rounds. Specifically, we prove the first bullet of Theorem 4. We call this algorithm AdaptiveSparseRecovery$_{\ell_\infty/\ell_2}$.

We proceed with the design and the analysis of the algorithm. We note that for $k/\epsilon = \mathcal{O}(\log^5 n)$[3], ℓ_∞/ℓ_2 gives already the desired result. So, we focus on the case of $k/\epsilon = \Omega(\log^5 n)$. We pick a hash function $h : [n] \to [B]$, where $B = ck/(\epsilon \log n)$ for some constant c large enough. The following follows by an application of Bernstein's Inequality and the Chernoff Bound, similarly to ℓ_∞/ℓ_2.

▶ **Lemma 11.** *With probability* $1 - 1/\text{poly}(n)$*, the following holds:*

$$\forall j \in [B] : |H_{k/\epsilon}(x) \cap h^{-1}(j)| \le \log n, \qquad \text{and} \qquad \left| \sum_{i \in h^{-1}(j) \setminus H_{k/\epsilon}(x)} x_i^2 \right| \le \frac{\epsilon}{k} \|x_{-k}\|_2^2.$$

We now run the ℓ_∞/ℓ_2 algorithm for the previous section on vectors $x_{h^{-1}(1)}, x_{h^{-1}(2)}, \ldots, x_{h^{-1}(B)}$ with sparsity parameter $\mathcal{O}(\log n)$, to obtain vectors $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_B$. The number of rounds is $\mathcal{O}(\text{loglog}(n))$, since we can run the algorithm in every bucket in parallel. By the definition of the ℓ_∞/ℓ_2 algorithm, one can see that $|\text{supp}(\hat{x}_j)| \le \mathcal{O}(\log n)$. We set $S = \cup_{j \in B} |\text{supp}(x_j)|$, and observe that $|S| = ck/(\epsilon \log n) \cdot \mathcal{O}(\log n) = \mathcal{O}(k/\epsilon)$. The number of measurements equals $ck/(\epsilon \log n) \cdot \mathcal{O}(\log n \cdot \text{loglog}(n \log(n/k))) = \mathcal{O}((k/\epsilon) \cdot \text{loglog}(n \log(n/k)))$.

▶ **Lemma 12.** *With probability* $1 - 1/\text{poly}(n)$*, we have that* $|S \setminus H_{k/\epsilon}(x)| \le \frac{k}{\epsilon \log^2 n}$.

Proof. Since every call to ℓ_∞/ℓ_2 fails with probability $1/\text{poly}(\log n)$, the probability that we have more than a $\frac{1}{\log n}$ fraction of the calls that fail is at most

$$\binom{B}{B/\log^2 n} \left(\frac{1}{\log n}\right)^{B/\log n} \le (e \log^2 n)^{\log n} (\log n)^{-B/\log n} \le \frac{1}{\text{poly}(n)}.$$

This implies that S will contain all but at most $B/\log^2 n \cdot \log n = k/(\epsilon \log^2 n)$ coordinates $i \in H_k(x)$. ◀

We now observe x_S directly and form the vector $z = x - x_S$, for which $\|z_{-k/(\epsilon \log^2 n)}\|_2 \le \|x_{-k/\epsilon}\|_2$. We now run a standard ℓ_2/ℓ_2 algorithm that fails with probability $1/\text{poly}(n)$ to obtain a vector $\hat{z}$ that approximates z (for example PartitionCountSketch$(z, k/(\epsilon \log^2 n), [n])$ suffices). We then output $\hat{z} + x_S$, for which $\|\hat{z} + x_S - x\|_2 = \|\hat{z} - z\| \le (1+\epsilon)\|z_{-k/(\epsilon \log n)}\|_2 \le (1+\epsilon)\|x_{-k}\|_2$. The number of measurements of this step is $\mathcal{O}(\frac{1}{\epsilon} \frac{k}{\log^2 n} \cdot \log n) = o(\frac{k}{\epsilon})$. The total number of rounds is clearly $\mathcal{O}(\text{loglog}(n \log(\frac{n\epsilon}{k})))$.

5 ℓ_2/ℓ_2 with Improved Dependence on ϵ

In this section, we prove the second part of Theorem 4. We first need an improved algorithm for the 1-sparse recovery problem.

▶ **Lemma 13.** *Let* $x \in \mathbb{R}^n$*. There exists an algorithm* ImprovedOneSparseRecovery*, that uses* $\mathcal{O}(\text{loglog}n + \frac{1}{\epsilon}\text{loglog}(\frac{1}{\epsilon}))$ *measurements in* $\mathcal{O}(\text{loglog}(n))$ *rounds, and finds with sufficiently small constant probability an* $\mathcal{O}(1)$*-sparse vector* $\hat{x}$ *such that* $\|\hat{x} - x\|_2 \le (1 + \epsilon)\|x_{-1}\|_2$.

[3] the constant 5 is arbitrary

Proof. We pick a hash function $h : [n] \to [B]$, where $B = \lceil 1/\epsilon^h \rceil$ for a sufficiently large constant h. Observe that all elements of $H_{\sqrt{B}}(x)$ are perfectly hashed under h with constant probability, and, $\forall j \in [B]$, $\mathbb{E}\left[\left\|x_{h^{-1}(j)\setminus H_{\sqrt{B}}}(x)\right\|_2\right] \leq 1/B\|x_{-\sqrt{B}}\|_2$. As in the previous sections, invoking Bernstein's inequality we can get that with probability $1 - 1/\text{poly}(B)$, $\forall j \in [B]$, $\left\|x_{h^{-1}(j)\setminus H_{\sqrt{B}}(x)\|_2}\right\|_2^2 \leq \frac{c\log B}{B}\|x_{-\sqrt{B}}\|_2^2$, where c is some absolute constant, and the exponent in the failure probability is a function of c.

We now define the vector $z \in \mathbb{R}^B$, the j-th coordinate of which equals $z_j = \sum_{i\in h^{-1}(j)} \sigma_{i,j}x_i$. We shall invoke Khintchine inequality to obtain $\forall j$,

$$\Pr\left[\left|\sum_{i\in h^{-1}(j)\setminus H_{\sqrt{B}}(x)} \sigma_{i,j}x_i\right|^2 > \frac{c'}{\epsilon}\left\|x_{h^{-1}(j)\setminus H_{\sqrt{B}}(x)}\right\|_2^2\right] \leq e^{-\Omega(1/\epsilon^2)},$$

for some absolute constant c'. This allows us to take a union-bound over all $B = \lceil 1/\epsilon^h \rceil$ entries of z to conclude that there exists an absolute constant ζ such that $\forall j \in [B]$, $\left|\sum_{i\in h^{-1}(j)\setminus H_{\sqrt{B}}(x)} \sigma_{i,j}x_i\right|^2 \leq \frac{c'}{\epsilon}\|x_{h^{-1}(j)\setminus H_{\sqrt{B}}(x)}\|_2^2 < \zeta\epsilon\|x_{-1}\|_2^2$, by setting h large enough. Now, for every coordinate $j \in [B]$ for which $h^{-1}(j) \cap H_{1,\epsilon}(x) = i^*$ or some $i^* \in [n]$, we have that $|z_j| \geq \left||x_{i^*}| - \sqrt{\frac{c\log B}{B}\cdot\frac{c'}{\epsilon}}\|x_{-\sqrt{B}}\|_2\right| \geq (1-\zeta)\sqrt{\epsilon}\|x_{-1}\|_2$, whereas for every $j \in [B]$ such that $h^{-1}(j)\cap H_{1,\epsilon\zeta}(x) = \emptyset$ it holds that $|z_j| \leq 2\zeta\sqrt{\epsilon}\|x_{-1}\|_2$. We note that $H_{1,\epsilon}(x) \subset H_{\sqrt{B}}(x)$, and hence all elements of $H_{1,\epsilon}(x)$ are also perfectly hashed under h. Moreover, observe that $\mathbb{E}\|z_{-1}\|_2^2 \leq \|x_{-1}\|_2^2$, and hence by Markov's inequality, we have that $\|z_{-1}\|_2^2 \leq 10\|x_{-1}\|_2^2$ holds with probability $9/10$. We run the ℓ_2/ℓ_2 algorithm of Theorem 4 for vector z with the sparsity being set to 1, and obtain vector $\hat{z}$. We then set $S = \text{supp}(\hat{z})$. We now define $w = (|z_1|, |z_2|, \ldots)$, for which $\|w_{-1}\|_2 = \|z_{-1}\|_2$. Clearly, $\|z - z_S\|_2^2 \leq \|z - \hat{z}\|_2^2 \leq (1+\epsilon)\|z_{-1}\|_2^2 = (1+\epsilon)\|w_{-1}\|_2^2$. So $\|w - w_S\|_2^2 = \|z - z_S\|_2^2 \leq (1+\epsilon)\|w_{-1}\|_2^2$. We now prove that $\left\|x - x_{\cup_{j\in S}h^{-1}(j)}\right\|_2 \leq (1+\mathcal{O}(\epsilon))\|x_{-1}\|_2$. Let i^* be the largest coordinate in magnitude of x, and $j^* = h(i^*)$. If $j^* \in S$, then it follows easily that $\|x - x_{\cup_{j\in S}h^{-1}(j)}\|_2 \leq \|x_{-1}\|_2$. Otherwise, since $\sum_{j\neq j^*} w_j^2 = \|w_{-1}\|_2^2$, and $\sum_{j\notin S} w_j^2 \leq (1+\epsilon)\|w_{-1}\|_2^2$, it must be the case that $\left|w_{j^*}^2 - \|w_S\|_2^2\right| \leq \epsilon\|w_{-1}\|_2^2 \leq 10\epsilon\|x_{-1}\|_2^2$. The above inequality, translates to $\sum_{i\in h^{-1}(j^*)} x_i^2 \leq |S|\zeta\epsilon\|x_{-1}\|_2^2 + \zeta\epsilon\|x_{-1}\|_2^2 + 10\epsilon\|x_{-1}\|_2^2 + \sum_{j\in S}\sum_{i\in h^{-1}(j)} x_j^2 = \mathcal{O}(\epsilon)\|x_{-1}\|_2^2 + \sum_{j\in S}\sum_{i\in h^{-1}(j)} x_j^2$. This gives $\left\|x - x_{\cup_{j\in S}h^{-1}(j)}\right\|_2 = \sum_{i\in h^{-1}(j^*)} x_i^2 + \sum_{j\notin S\cup\{j^*\}}\sum_{i\in h^{-1}(j)} x_i^2 \leq \mathcal{O}(\epsilon)\|x_{-1}\|_2^2 + \mathcal{O}(1)\zeta\epsilon\|x_{-1}\|_2^2 + \sum_{j\in S}\sum_{i\in h^{-1}(j)} x_j^2 + \sum_{j\notin S\cup\{j^*\}}\sum_{i\in h^{-1}(j)} x_i^2 + \leq (1+\mathcal{O}(\epsilon))\|x_{-1}\|_2^2$.

Given S, we run the 1-sparse recovery routine on vectors x_j for $j \in S$, with a total of $\mathcal{O}(\text{loglog}n)$ measurements and $\mathcal{O}(\text{loglog}n)$ rounds. We then output $\{x_{i_j}\}_{j\in S}$. Let i_j be the index returned for $j \in S$ by the 1-sparse recovery routine. Since we have a constant number of calls to the 1-sparse recovery routine (because S is of constant size), all our 1-sparse recovery routines will succeed. We now have that $\|x - x_{\cup_{j\in S} i_j}\|_2 \leq \|x_{\bar{S}}\|_2 + \sum_{j\in S}\|x_{h^{-1}(j)} - x_{i_j}\|_2 \leq \|x_{\bar{S}}\|_2 + \sum_{j\in S}(1+\epsilon)\|x_{h^{-1}(j)\setminus H_1(x)}\|_1 \leq (1+\mathcal{O}(\epsilon))\|x_{-1}\|_2$. Rescaling ϵ, we get the desired result. ◀

The algorithm for general k is similar to [10], apart from the fact that we subsample at a slower rate, and also use our new 1-sparse recovery algorithm as a building block. In the algorithm below, R_r is the universe we are restricting our attention on at the rth round. Moreover, J is the set of coordinates that we have detected so far. We are now ready to prove Theorem 4.

Proof. The number of measurements is bounded in the exact same way as in Theorem 3.7 from [10].

Algorithm 2 Adaptive ℓ_2/ℓ_2 Sparse Recovery.

1. $R_0 \leftarrow [n]$.
2. $x_0 \leftarrow \vec{0}$.
3. $\delta_0 \leftarrow \delta/2, \epsilon_0 \leftarrow \epsilon/e, f_0 \leftarrow 1/32, k_0 \leftarrow k$.
4. $J \leftarrow \emptyset$.
5. **For** $r = 0$ to $\mathcal{O}(\log^* k)$ **do**
6. **For** $t = 0$ to $\Theta(k_r \log(1/(\delta_r f_r)))$ **do**
7. $S_t \leftarrow \textsc{Subsample}(x - x^{(r)}, R_r, 1/(C_0 k_r))$.
8. $J \leftarrow J \cup \textsc{ImprovedOneSparseRecovery}((x - x^{(r)})_{S_t})$.
9. **End For**
10. $R_{r+1} \leftarrow [n] \setminus J$.
11. $\delta_{r+1} \leftarrow \delta_r/8$.
12. $\epsilon_{r+1} \leftarrow \epsilon_r/2$.
13. $f_{r+1} \leftarrow 1/2^{1/(4^{i+r} f_r)}$.
14. $k_{r+1} \leftarrow f_r k_r$.
15. $R_{r+1} \leftarrow [n] \setminus J$.
16. **End For**
17. $\hat{x} \leftarrow x^{(r+1)}$.
18. Return $\hat{x}$.

We fix a round r and $i \in H_{k_r,\epsilon_r}(x^{(r)})$. Then the call to $\textsc{Subsample}(R_r, 1/(C_0 k_r))$ yields

$$\Pr\left[|H_{k_r,\epsilon_r}(x - x^{(r)}) \cap S_t| = \{i\}\right] \geq \frac{1}{C_0 k_r}, \quad \mathbb{E}\left[\|x_{S_t \setminus H_{k_r,\epsilon_i}(x^{(r)})}\|_2^2\right] = \frac{1}{C_0 k_r}\|x_{-k_r}\|_2^2.$$

Setting C_0 to be large enough and combining Markov's inequality with the guarantee of Lemma 13, we get that the probability that the call to $\textsc{ImprovedOneSparseRecovery}(x_{S_t})$ returns i is $\Theta(1/k_r)$. Because we repeat $k_r \log(1/(f_r\delta_r))$, the probability that i or a set S_i of size $\mathcal{O}(1)$ such that $\|x_{\{i\}} - x_{S_i}\|_2 \leq \epsilon_i \|x_{-k_r}\|_2^2$, is not added in J is at most $(1 - 1/k_r)^{k_r \log(1/(f_r\delta_r))} = f_r\delta_r$.

Given the above claim, the number of measurements is $\mathcal{O}((k\text{loglog}n + k/\epsilon\text{loglog}(1/\epsilon)\log(1/\delta))$ and the analysis of the iterative loop proceeds almost identically to Theorem 3.7 of [10]. ◀

References

1 Akram Aldroubi, Haichao Wang, and Kourosh Zarringhalam. Sequential adaptive compressed sampling via Huffman codes. *arXiv preprint arXiv:0810.4916*, 2008.
2 Pranjal Awasthi, Maria-Florina Balcan, Nika Haghtalab, and Hongyang Zhang. Learning and 1-bit compressed sensing under asymmetric noise. In *Annual Conference on Learning Theory*, pages 152–192, 2016.
3 Khanh Do Ba, Piotr Indyk, Eric Price, and David P. Woodruff. Lower bounds for sparse recovery. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1190–1197, 2010.
4 Rui M. Castro, Jarvis Haupt, Robert Nowak, and Gil M. Raz. Finding needles in noisy haystacks. In *International Conference on Acoustics, Speech and Signal Processing*, pages 5133–5136, 2008.
5 Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss. Approximate sparse recovery: optimizing time and measurements. *SIAM Journal on Computing*, 41(2):436–453, 2012.
6 Rishi Gupta, Piotr Indyk, Eric Price, and Yaron Rachlin. Compressive sensing with local geometric features. *International Journal of Computational Geometry & Applications*, 22(04):365–390, 2012.

7 Jarvis Haupt, Waheed U Bajwa, Michael Rabbat, and Robert Nowak. Compressed sensing for networked data. *IEEE Signal Processing Magazine*, 25(2):92–101, 2008.

8 Jarvis Haupt, Robert Nowak, and Rui Castro. Adaptive sensing for sparse signal recovery. In *Digital Signal Processing Workshop and IEEE Signal Processing Education Workshop*, pages 702–707, 2009.

9 Jarvis D. Haupt, Richard G. Baraniuk, Rui M. Castro, and Robert D. Nowak. Compressive distilled sensing: Sparse recovery using adaptivity in compressive measurements. In *Asilomar Conference on Signals, Systems and Computers*, pages 1551–1555, 2009.

10 Piotr Indyk, Eric Price, and David P. Woodruff. On the power of adaptivity in sparse recovery. In *Annual IEEE Symposium on Foundations of Computer Science*, pages 285–294, 2011.

11 Shihao Ji, Ya Xue, and Lawrence Carin. Bayesian compressive sensing. *IEEE Transactions on Signal Processing*, 56(6):2346–2356, 2008.

12 Raghunandan M. Kainkaryam, Angela Bruex, Anna C. Gilbert, John Schiefelbein, and Peter J. Woolf. poolmc: Smart pooling of mrna samples in microarray experiments. *BMC Bioinformatics*, 11:299, 2010.

13 Yi Li and Vasileios Nakos. Sublinear-time algorithms for compressive phase retrieval. *arXiv preprint arXiv:1709.02917*, 2017.

14 Dmitry M. Malioutov, Sujay Sanghavi, and Alan S. Willsky. Compressed sensing with sequential observations. In *International Conference on Acoustics, Speech and Signal Processing*, pages 3357–3360, 2008.

15 Tom Morgan and Jelani Nelson. A note on reductions between compressed sensing guarantees. *CoRR*, abs/1606.00757, 2016.

16 Shanmugavelayutham Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, 2005.

17 Eric Price and David P. Woodruff. (1+eps)-approximate sparse recovery. In *IEEE Symposium on Foundations of Computer Science*, pages 295–304, 2011.

18 Eric Price and David P. Woodruff. Lower bounds for adaptive sparse recovery. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 652–663, 2013.

19 Noam Shental, Amnon Amir, and Or Zuk. Rare-allele detection using compressed se(que)nsing. *CoRR*, abs/0909.0400, 2009.

20 Tasuku Soma and Yuichi Yoshida. Non-convex compressed sensing with the sum-of-squares method. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 570–579, 2016.

Approximate Low-Weight Check Codes and Circuit Lower Bounds for Noisy Ground States

Chinmay Nirkhe
Electrical Engineering and Computer Sciences, University of California, Berkeley
387 Soda Hall Berkeley, CA 94720, U.S.A.
https://people.eecs.berkeley.edu/~nirkhe/
nirkhe@cs.berkeley.edu

Umesh Vazirani
Electrical Engineering and Computer Sciences, University of California, Berkeley
387 Soda Hall Berkeley, CA 94720, U.S.A.
https://people.eecs.berkeley.edu/~vazirani/
vazirani@cs.berkeley.edu

Henry Yuen
Electrical Engineering and Computer Sciences, University of California, Berkeley
387 Soda Hall Berkeley, CA 94720, U.S.A.
https://www.henryyuen.net/
hyuen@cs.berkeley.edu

Abstract

The No Low-Energy Trivial States (NLTS) conjecture of Freedman and Hastings (Quantum Information and Computation 2014), which asserts the existence of local Hamiltonians whose low-energy states cannot be generated by constant-depth quantum circuits, identifies a fundamental obstacle to resolving the quantum PCP conjecture. Progress towards the NLTS conjecture was made by Eldar and Harrow (Foundations of Computer Science 2017), who proved a closely related theorem called No Low-Error Trivial States (NLETS). In this paper, we give a much simpler proof of the NLETS theorem and use the same technique to establish superpolynomial circuit size lower bounds for noisy ground states of local Hamiltonians (assuming $\mathsf{QCMA} \neq \mathsf{QMA}$), resolving an open question of Eldar and Harrow. We discuss the new light our results cast on the relationship between NLTS and NLETS.

Finally, our techniques imply the existence of *approximate quantum low-weight check (qLWC) codes* with linear rate, linear distance, and constant weight checks. These codes are similar to quantum LDPC codes except (1) each particle may participate in a large number of checks, and (2) errors only need to be corrected up to fidelity $1 - 1/\mathsf{poly}(n)$. This stands in contrast to the best-known stabilizer LDPC codes due to Freedman, Meyer, and Luo which achieve a distance of $O(\sqrt{n \log n})$.

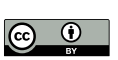
The principal technique used in our results is to leverage the Feynman-Kitaev clock construction to approximately embed a subspace of states defined by a circuit as the ground space of a local Hamiltonian.

2012 ACM Subject Classification Theory of computation → Quantum complexity theory

Keywords and phrases quantum pcps, local hamiltonians, error-correcting codes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.91

Related Version A full version of the paper is available at https://arxiv.org/abs/1802.07419.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 91; pp. 91:1–91:11

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding This work was supported by ARO Grant W911NF-12-1-0541 and NSF Grant CCF-1410022.

Acknowledgements We thank Dorit Aharanov, Adam Bouland, Elizabeth Crosson, Bill Fefferman, Lior Eldar, Zeph Landau, Ashwin Nayak, Nicholas Spooner, and Thomas Vidick for helpful discussions.

1 Introduction

The quantum PCP conjecture [4, 2] is a central open question in quantum complexity theory. To understand the statement, it is helpful to review the dictionary translating between classical constraint satisfaction problems (CSPs) and their quantum analogue, the local Hamiltonian problem. A classical CSP on n variables corresponds to a local Hamiltonian $H = H_1 + \cdots + H_m$ acting on n qubits[1]. A solution to the CSP corresponds to an n qubit quantum state, and the number of violated constraints corresponds to the energy (eigenvalue) of that quantum state. The NP-hardness of SAT corresponds to the QMA-hardness of deciding whether the H has minimum eigenvalue at most a or at least b for given a, b such that $b - a = 1/\mathsf{poly}(n)$. The quantum analogue of the PCP theorem, called the qPCP conjecture, asserts that the problem remains QMA-hard even when $b - a \geq cm = c\|H\|$.

Just as the classical PCP theorem connects coding theory to constraint satisfaction problems, it is natural to expect any resolution of the quantum PCP conjecture to rely on — and to reveal — deep connections between the theory of quantum error-correcting codes and ground states (i.e. states of minimum energy) of local Hamiltonians. Examples of quantum error-correcting codes realized as the ground spaces of local Hamiltonians already play a central role in our understanding of the physical phenomenon known as topological order [30]. Moreover, it has been suggested that the qPCP conjecture is closely related to one of the biggest open problems in quantum coding theory: whether quantum low density parity check (qLDPC) codes with linear rate and linear distance exist [23, 13, 33].

The difficulty of the qPCP conjecture motivated Freedman and Hastings to formulate a simpler goal called the *No Low-Energy Trivial States (NLTS) Conjecture* [22]. One way to put one's finger on the additional difficulty of qPCP (beyond the "standard" difficulty of proving a classical PCP theorem) is that solutions of QMA-hard problems are expected to have high description complexity. For example, if NP $\neq$ QMA, then ground states of local Hamiltonians do not have classically checkable polynomial-size descriptions. The NLTS conjecture isolates this aspect of high description complexity by asserting the existence of a family of local Hamiltonians $\{H^{(n)}\}_{n=1}^{\infty}$ where $H^{(n)}$ acts on n particles, such that low-energy states (of energy less than $c\|H\|$) cannot be generated by quantum circuits of constant depth. A much stronger version of the NLTS conjecture is a necessary consequence of the qPCP conjecture: assuming QCMA $\neq$ QMA,[2] low-energy states cannot be described even by polynomial-size quantum circuits. However, one of the advantages of the NLTS conjecture is that it does not involve complexity classes such as QMA, but rather focuses on the entanglement complexity that is intrinsic to low-energy states of local Hamiltonians.

[1] For normalization, we assume that the terms of a local Hamiltonian have spectral norm at most 1.

[2] For precise definitions of the complexity classes QCMA and QMA, we refer the reader to [20, 29]. Roughly speaking, QMA is the class of problems for which the solution is a quantum state that can be efficiently checked by a quantum computer. QCMA is the class of problems where the solution is a *classical* string that can be efficiently checked by a quantum computer.

Like the qPCP conjecture, the NLTS conjecture remains unresolved. In [19], Eldar and Harrow proposed a variant of the NLTS called *No Low-Error Trivial States (NLETS)*, which is itself a necessary consequence[3] of NLTS. The difference was that rather than considering low-energy states of H, they considered a notion of "local corruption error", what they call *ϵ-error states*: these are states that differ from the ground state in at most ϵn qubits. More precisely, σ is ϵ-error for a local Hamiltonian H if there exists a ground state ρ of H and a set S of at most ϵn qudits such that $\mathrm{Tr}_S(\rho) = \mathrm{Tr}_S(\sigma)$. Under this definition they were able to establish a family of Hamiltonians for which any ϵ-error state requires circuit depth of $\Omega(\log n)$. This was welcomed as very encouraging progress towards establishing NLTS, since NLETS could be regarded as a close proxy for NLTS, with a technical change in definition of distance under which to examine the robustness of the ground space.

In this paper, we start by giving a simple argument for the $\Omega(\log n)$ circuit depth lower bound of Eldar and Harrow; our lower bound holds even under a more general error model, which allows any probabilistic mixture of ϵ-error states (we call these states *noisy ground states*). Moreover, we can use the same techniques to answer their open question of whether one can obtain circuit size lower bounds on low-error states that go beyond logarithmic depth: specifically, we show that there exists a family of local Hamiltonians whose noisy ground states require superpolynomial-size circuits, assuming $\mathsf{QCMA} \neq \mathsf{QMA}$.

One way to view these results is that they provide further progress towards the NLTS conjecture and beyond. However, it is instructive to take a step back to consider more closely the basic difference between NLETS and NLTS. This lies in the different notion of approximation: in NLETS, approximation corresponds to local corruptions in ϵn sites, where n is the total number of particles, whereas in NLTS approximation corresponds to energy at most $\epsilon \|H\|$ (intuitively, at most ϵ fraction of the terms of the Hamiltonian are violated). An alternative perspective on our results is that they suggest these two notions of approximation are quite different. This view is reinforced by the fact that our $\Omega(\log n)$-circuit depth lower bounds on noisy ground states holds for a family of 1D Hamiltonians, whereas we know that NLTS and qPCP Hamiltonians cannot live on any constant-dimensional lattice [2]. This suggests that in the context of the qPCP and NLTS conjectures, the correct notion of distance is given by the energy or number of violated terms of the Hamiltonian.

On the other hand, the local corruption distance as defined by Eldar and Harrow for their NLETS result is the natural one that arises in quantum error correction: the distance of a code is *defined* by the maximum number of qubits of a codeword that can be erased while maintaining recoverability. We give a construction of a family of codes (inspired by the construction used in our noisy ground state lower bound) that we call *quantum low weight check (qLWC) codes*. The family of codes we consider are approximate error-correcting codes in the sense of [16, 11]. They are closely related to qLDPC codes, with the difference that they are not stabilizer codes and therefore the low weight checks are not Pauli operators. Specifically, we give a family of approximate qLWCs with linear distance and linear rate. Constructing qLDPC codes with similar parameters is a central open question in coding theory, with the best-known stabilizer LDPC codes due to Freedman, Meyer, and Luo which achieve a distance of $O(\sqrt{n \log n})$ [23].

What is common to the above results is the technique. We start with the observation that the complicated part of the Eldar and Harrow proof is constructing a local Hamiltonian whose ground states share some of the properties of the cat state $|\circledast_n\rangle = (|0\rangle^{\otimes n} + |1\rangle^{\otimes n})/\sqrt{2}$.

[3] The local Hamiltonian family must be of bounded-degree, meaning no particle participates in more than a constant number of Hamiltonian terms.

To do so, they constructed a local Hamiltonian corresponding to a quantum error-correcting code (based on the Tillich-Zemor hypergraph product construction) and showed that its ground states have non-expansion properties similar to those of the cat state [19]. Our starting point is the observation that the Tillich-Zemor construction is unnecessary and that one can make the cat state *approximately* a ground state of a local Hamiltonian in the following sense: we construct a Feynman-Kitaev clock Hamiltonian corresponding to the circuit that generates $|\boxtimes_n\rangle$ from $|0\rangle^{\otimes n}$.[4] The ground state of this Hamiltonian is the history state of this computation, and we directly argue that the circuit depth necessary to generate this history state is at least $\Omega(\log n)$. This same argument even allows us to lower bound the circuit depth of *approximate* noisy ground states (i.e. states that are close in trace distance to a noisy ground state).

The Feynman-Kitaev clock Hamiltonian plays a central role in our construction of qLWCs, with history states playing the role of codewords. The fact that such a construction yields an error-correcting code flies in the face of classical intuition. After all, it is the brittleness of the Cook-Levin tableau [15, 32] (the classical analogue of the history state) that motivates the elaborate classical PCP constructions [7, 6, 18]. The difference is that *time* is in superposition in a quantum history state. We do not yet understand the implications of this observation. For example, is it possible that it might lead to new ways of constructing qLDPC codes with super-efficient decoding procedures? There are precedents for such connections between computational phenomena and codes, most notably with the PCP theorem and the construction of locally testable and locally checkable codes.

Furthermore, while quantum error-correcting codes have typically provided a wealth of examples of interesting local Hamiltonians, our construction of qLWCs also suggest that a fruitful connection exists in the opposite direction: by considering techniques to construct local Hamiltonians (such as the Feynman-Kitaev clock construction), we can construct an interesting example of a quantum error-correcting code. We note that this reverse connection is starting to take hold in other areas of quantum information theory and physics: see [12, 28].

2 Summary of Results

Before we present our results, we motivate our definition of *noisy ground states*.

2.1 Noisy ground states

The NLETS Theorem and NLTS conjecture describe different ways in which the ground space entanglement is robust. The ground states of NLETS Hamiltonians are robust against local corruptions in ϵn sites, where n is the total number of particles. NLTS Hamiltonians are robust against low-energy excitations in the sense that all states with energy at most $\epsilon\|H\|$ retain nontrivial circuit complexity.

In this paper, we study another way that ground space entanglement can be robust. We focus on the concept of *noisy ground state*, which is a generalization of low-error states: an ϵ-noisy ground state σ of a local Hamiltonian H is a probabilistic mixture of ϵ-error states $\{\sigma_i\}$.

This notion of noisy ground state is naturally motivated by the following situation: consider a ground state ρ of H. On each particle independently apply the following process

[4] A similar construction of a clock Hamiltonian was also considered by Crosson and Bowen in the context of idealized adiabatic algorithms [17]. The construction is inspired by techniques of [14, 10].

$\mathcal{M}$: with probability ϵ, apply a noisy channel $\mathcal{N}$, and with probability $1-\epsilon$ apply the identity channel $\mathcal{I}$. The resulting state is

$$\begin{aligned}\mathcal{M}(\rho) &= \left((1-\epsilon)\mathcal{I} + \epsilon\mathcal{N}\right)^{\otimes n}(\rho) \\ &= \sum_{S\subseteq[n]} (1-\epsilon)^{n-|S|}\epsilon^{|S|}\mathcal{N}^S(\rho) \\ &\approx \sum_{S:|S|\leq 2\epsilon n} (1-\epsilon)^{n-|S|}\epsilon^{|S|}\mathcal{N}^S(\rho)\end{aligned} \tag{1}$$

where $\mathcal{N}^S$ denotes the tensor product of the map $\mathcal{N}$ acting on the particles indexed by S. The last approximate equality follows from the fact that with overwhelmingly large probability, $\mathcal{N}^S$ acts on at most $2\epsilon n$ particles. Notice that the expression on the right hand side is (up to normalization) a 2ϵ-noisy ground state, because when $|S| \leq 2\epsilon n$, the state $\mathcal{N}^S(\rho)$ is a 2ϵ-error state.

This justifies the name "noisy ground state", as the operation $\mathcal{M}$ is is a reasonable model of noise that occurs in physical processes (and is frequently considered in work on quantum fault-tolerance). Furthermore, we believe that our model arises naturally in the context of noisy adiabatic quantum computation.

As mentioned before, noisy ground states are a generalization of low-error states but are a special case of low-energy states: since low-error states are themselves low-energy states, a convex combination of them is also low-energy.

We prove several results about the robustness of entanglement in noisy ground states.

2.2 Logarithmic circuit depth lower bound

First, we generalize Eldar and Harrow's logarithmic circuit depth lower bound [19] to encompass noisy ground states. Furthermore, we present a family of Hamiltonians that is *one-dimensional*; in other words, the particles of the Hamiltonian are arranged on a line and the Hamiltonian terms act on neighboring particles.

We call this the *Logarithmic Noisy Ground States (LNGS) Theorem*[5].

▶ **Theorem 1** (Logarithmic lower bound). *There exists a family of 3-local Hamiltonians $\{H^{(n)}\}$ on a line, acting on particles of dimension 3, such that for all $n \in \mathbb{N}$, for all $0 \leq \epsilon < 1/48$, $0 \leq \delta < \frac{1}{8} - 6\epsilon$, the δ-approximate circuit depth of any ϵ-noisy ground state σ for $H^{(n)}$ is at least $\frac{1}{2}\log(n/2)$.*

Here, the δ-approximate circuit depth of ρ means the circuit depth needed to produce a state that is δ-close to ρ in trace distance.

Our proof of Theorem 1 is simple and self-contained. As a consequence of our simpler local Hamiltonian construction, we obtain improved parameters over those in [19]. Furthermore, as we will discuss below in Section 2.4, the fact that our LNGS Hamiltonian is one-dimensional gives a strong separation between NLETS/LNGS and NLTS Hamiltonians.

2.2.1 Superpolynomial circuit size lower bound

A question that was left open by [19] is whether one can obtain circuit lower bounds on low-error states that are better than logarithmic – say polynomial or even exponential. We show that there exists a family of local Hamiltonians whose noisy ground states require

[5] We pronounce this "Longs."

superpolynomial[6] size circuits, assuming QCMA $\neq$ QMA. Since low-error states are noisy ground states, this provides an answer to Eldar and Harrow's open question.

We call this the *Superpolynomial Noisy Ground States (SNGS) Theorem*[7].

▶ **Theorem 2** (Superpolynomial Noisy Ground States (SNGS)). *If* QCMA $\neq$ QMA*, then there exists* $q, \epsilon > 0$ *and a family of* 7*-local Hamiltonians* $\{H^{(n)}\}$ *acting on dimension-*q *qudits such that for all* $0 \leq \delta < 1/5$*, the* δ*-approximate circuit complexity of any family* $\{\sigma_n\}$ *of* ϵ*-noisy ground states for* $\{H^{(n)}\}$ *grows faster than any polynomial in* n*.*

We call such a family $\{H^{(n)}\}$ *SNGS Hamiltonians*. The following is a proof sketch. Let $L = (L_{yes}, L_{no})$ be the QMA-complete language consisting of descriptions of polynomial-size verifier circuits acting on a witness state and ancilla qubits. We convert each circuit $C \in L$, into a circuit C' where C' applies in order: (a) a unitary V to encode the state in an error-correcting code[8], (b) a collection of identity gates, (c) the unitary $V^\dagger$ to decode the state, and (d) the gate circuit C. The construction maintains that the circuits C' and C are equivalent. We then generate the Feynman-Kitaev clock Hamiltonian for C'. Let H_C be this Hamiltonian. The family of SNGS Hamiltonians is precisely $\{H_C : C \in L_{yes}\}$.

In order to prove that all noisy ground states of this Hamiltonian must have superpolynomial circuit size, we show that if there was a noisy ground state with a polynomial-size generating circuit, then the description of the generating circuit would suffice as a *classical* witness for the original QMA-complete problem. In the *yes* case, the construction of C' from C enforces that tracing out the time register of the noisy ground state will yield a state close to a convex combination of $\{\mathrm{Enc}(|\xi_i, 0\rangle)\}$ where $\mathrm{Enc}(\cdot)$ is the encoding function for the error-correcting code and $\{|\xi_i\rangle\}$, a collection of accepting witness. Therefore, given the description of the generating circuit for the noisy ground state, we can generate the noisy ground state and decode the original witness state. It suffices then to check the witness by running the original circuit C. The *no* case follows easily from the definition of L_{no}. This proves that $L \in$ QCMA, proving QCMA = QMA, contradicting the original assumption.

2.2.2 Semi-explicit SNGS Hamiltonians via oracle separations

It is an open question in quantum complexity theory of whether QCMA is equal to QMA. Aaronson and Kuperberg gave the first complexity-theoretic evidence that they are different by constructing a *quantum* oracle $\mathcal{O}$ such that $\mathsf{QCMA}^{\mathcal{O}} \subsetneq \mathsf{QMA}^{\mathcal{O}}$ [1]. Fefferman and Kimmel later showed that one can obtain the same oracle separation with *in-place* oracles $\mathcal{O}$, which are permutation matrices in the standard basis [20]. The separations of [1, 20] hold as long as the locality of the oracles $\mathcal{O}$ is $\omega(\log n)$ (i.e. superlogarithmic in the problem size).

We show that any oracle separation between QCMA and QMA can be leveraged to obtain a semi-explicit family of SNGS Hamiltonians:

▶ **Theorem 3.** *There exists* $q, \epsilon > 0$*, a function* $k(n) = O(\log^{1+\alpha} n)$ *for arbitrarily small* $\alpha > 0$ *and a family of* k*-local Hamiltonians* $\{H^{(n)}\}$ *acting on dimension-*q *qudits such that the following holds: The circuit complexity of any family* $\{\sigma_n\}$ *of* ϵ*-noisy ground states for* $\{H^{(n)}\}$ *grows faster than any polynomial in* n*. Furthermore, there is exactly one term in* $H^{(n)}$ *that is* $k(n)$*-local; all other terms are* 7*-local.*

[6] Here, "superpolynomial" refers to functions $f(n)$ that grow faster than any polynomial in n.
[7] We pronounce this "Songs".
[8] Such asymptotically good codes are known to exist (e.g, [8, 24]).

Unlike Theorem 2, the superpolynomial lower bound on the circuit complexity of noisy ground states does not require any complexity-theoretic assumption! The caveat is that this family is only known to exist via a counting argument; there is exactly *one* term of the Hamiltonian that has $\omega(\log n)$-locality and does not have an explicit description. However, however, all of other the terms of the local Hamiltonians are 7-local and have explicit descriptions.

The essential idea is to apply the proof of Theorem 2 to the $\mathsf{QMA}^{\mathcal{O}}$ verifier that decides a language L which is not in $\mathsf{QCMA}^{\mathcal{O}}$. In both [1, 20], this verifier only makes a single call to the oracle $\mathcal{O}$. Thus there is one term in the Feynman-Kitaev clock Hamiltonian corresponding to the propagation of that oracle call. Since we do not have an explicit description of a separating oracle $\mathcal{O}$, this Hamiltonian term is non-explicit.

2.3 Asymptotically good approximate low-weight check codes

The techniques from the previous sections also give rise to what we call *approximate quantum low-weight check (qLWC) codes*. These are closely related to quantum low-density parity check (qLDPC) codes, which are stabilizer codes where each parity check acts on a bounded number of particles, and each particle participates in a bounded number of parity checks. It is a long-standing open question of whether asymptotically good qLDPC codes exist (i.e. constant locality, constant rate, and constant relative distance). The qLDPC conjecture posits that such codes exist.

We show that if one relaxes the conditions of (a) each particle participating in a small number of constraints, and (b) that we can *exactly* recover from errors, we can obtain locally defined quantum error-correcting codes with such good parameters. First, we define our notion of approximate qLWC codes:

▶ **Definition 4** (Approximate qLWC code)**.** A local Hamiltonian $H = H_1 + \cdots + H_m$ acting on n dimension-q qudits is a $[[n, k, d]]_q$ *approximate quantum LWC code* with error δ and locality w iff each of the terms H_i act on at most w qudits and there exists encoding and decoding maps Enc, Dec such that

1. $\langle\Psi| H |\Psi\rangle = 0$ if and only if $|\Psi\rangle\langle\Psi| = \mathrm{Enc}(|\xi\rangle\langle\xi|)$ for some $|\xi\rangle \in (\mathbb{C}^q)^{\otimes k}$.
2. For all $|\phi\rangle \in (\mathbb{C}^q)^{\otimes k} \otimes \mathcal{R}$ where $\mathcal{R}$ is some purifying register, for all completely positive trace preserving maps $\mathcal{E}$ acting on at most $(d-1)/2$ qudits,

$$\|\mathrm{Dec} \circ \mathcal{E} \circ \mathrm{Enc}(|\phi\rangle\langle\phi|) - |\phi\rangle\langle\phi|\|_1 \leq \delta. \tag{2}$$

Here, the maps Enc, $\mathcal{E}$, and Dec do not act on register $\mathcal{R}$.

The first condition of the above definition enforces that the ground space of the Hamiltonian H of an approximate qLWC code is a q^k-dimensional codespace; it is the exactly the image of the encoding map Enc. The second condition corresponds to the approximate error-correcting condition, where we only require that the decoded state is *close* to the original state (i.e., we no longer insist that $\mathrm{Dec} \circ \mathcal{E} \circ \mathrm{Enc}$ is exactly the identity channel $\mathcal{I}$). Although there are few results on approximate quantum error-correcting codes, we do know that relaxing the exact decoding condition yields codes with properties that cannot be achieved using exact codes [31, 16, 11].

Our proof of Theorem 2 yields a construction of an approximate quantum LWC code with distance $\Omega(n)$.We believe this may be of independent interest.

▶ **Theorem 5** (Good approximate qLWC codes exist)**.** *For all error functions $\delta(n)$ there exist a family of $[[n,k,d]]_q$ approximate quantum LWC codes with the following parameters:*

$$\begin{aligned} \textit{Qudit dimension} \quad & q = O(1), \\ \textit{Error} \quad & \delta = \delta(n), \\ \textit{Locality} \quad & w = 3 + 2r, \\ \textit{Blocklength} \quad & n = O(rk), \\ \textit{Distance} \quad & d = \Omega(n/r) \end{aligned}$$

where

$$r = O\left(\frac{\log\left(1 + 4/\delta^2\right)}{\log n}\right) + 2. \tag{3}$$

Furthermore, the encoding and decoding maps for these codes are explicit and efficiently computable.

Observe that when $\delta(n) = 1/\mathsf{poly}(n)$, the parameter $r = O(1)$.

By comparison, the best-known qLDPC codes (of the stabilizer variety) with constant locality have distance bounded by $O(\sqrt{n \log n})$ [23]. Hastings constructs a qLDPC stabilizer code with constant locality that has distance $n^{1-\epsilon}$ for any $\epsilon > 0$, assuming a conjecture in high-dimensional geometry [26, 25]. Bacon, et al. were able to construct sparse subsystem codes (a generalization of stabilizer codes) with constant locality and distance $n^{1-o(1)}$ [9]. We note that, interestingly, the codes of [9] are constructed from fault-tolerant quantum circuits that implement a stabilizer code — this is similar to the way we construct our approximate qLWC codes!

2.4 Implications for NLTS, quantum PCP and quantum LDPC

Our investigation into noisy ground states and approximate low-weight check codes is motivated by a number of important open questions in quantum information theory: NLTS, quantum PCP, and quantum LDPC. We believe that our results help clarify the status of these open problems, and the relationships between them.

A separation between LNGS/SNGS and NLTS Hamiltonians.

First, our logarithmic circuit-depth lower bound for noisy ground states (Theorem 1) gives a strong separation between the notions of entanglement robustness in NLETS and NLTS: we showed that a one-dimensional local Hamiltonian is NLETS. However, it is easy to see that one-dimensional Hamiltonians (or any Hamiltonian on a constant-dimensional lattice) cannot be NLTS. To see this, consider taking a n-particle ground state $|\Psi\rangle$ of a 1D Hamiltonian H; divide up the n particles into contiguous chunks of length L. Let $\sigma = \rho_1 \otimes \rho_2 \otimes \cdots \otimes \rho_{n/L}$ where ρ_i is the reduced density matrix of $|\Psi\rangle$ on the i'th chunk. This state σ violates $O(n/L)$ terms of the Hamiltonian (since H is one-dimensional). Therefore, it is a ϵ-energy state of H for $L = \Theta(1/\epsilon)$. On the other hand, σ is a tensor product state that can be generated by $2^{O(1/\epsilon)}$-depth circuits, which is constant for constant ϵ. This indicates that the form of entanglement robustness as expressed in NLETS and in our LNGS/SNGS Hamiltonian constructions is much weaker than the entanglement robustness required by the NLTS conjecture and quantum PCP, where one has to look for Hamiltonians on high-dimensional geometries.

Quantum LDPC codes and the Quantum PCP conjecture.

Resolving the qPCP conjecture would likely involve a transformation from H to H' that (at the very least) has the property that exact ground states of H (or closeby states in trace distance) can be recovered from *low-energy* states of H'. It has been suggested that such a transformation would involve some kind of qLDPC code [22, 3, 26, 19]. In fact, it is believed that a special kind of qLDPC code, called a *quantum locally testable code (qLTC)*, is necessary [3]. However, the existence of qLTCs (or even qLDPC codes) with constant relative distance is a major open problem.

We believe our results on approximate quantum LWC codes present two take-home messages for the qPCP and qLDPC conjectures. First, it is important that a qPCP (or a qLTC) Hamiltonian be local, but it is *not* necessary that the Hamiltonian be bounded degree (meaning that each particle only participates in a few terms). The bounded degree condition is useful in the original context for qLDPCs, where an important motivation is to find fast decoding algorithms. In the context of qPCP/qLTC, however, decoding efficiency is not an immediate concern; thus resolving the qPCP conjecture *need not* resolve the qLDPC conjecture.

Second, we believe this gives evidence that considering codes other than stabilizer codes — such as approximate codes or subsystem codes — may be useful in the quest for both qPCP and qLDPC. Most work on qLDPC codes has focused on constructing CSS and stabilizer codes, but it may be fruitful to branch out beyond the CSS/stabilizer setting for the purposes of understanding the possibilities (or limits) of qPCP/qLDPC. For example, our qLWC codes are unconventional in a few ways: they are defined by non-commuting Hamiltonians, they only admit approximate recovery, and each particle participates in a large number of checks.

2.5 Open questions

We list a few open problems.

1. Are there SNGS Hamiltonians or (approximate) qLWC codes that are geometrically local (with respect to, say, the Euclidean metric)? Our construction of a one-dimensional NLGS Hamiltonian uses a simplification of a technique of Aharanov et. al. [5] of converting a quantum circuit into a two-dimensional local Hamiltonian. This technique works because of the specific structure of the circuit generating the $|\boxtimes\rangle$ state. In general, the transformation involves increasing the number of qudits by more than a constant factor. If this factor is $\Theta(n^\alpha)$, then the ground states are resilient to errors of size at most $n^{1-\alpha}$.
2. Is there a family of local Hamiltonians such that any *superposition* (not just convex combination) of low-error states have large circuit complexity? This notion is a generalization of a noisy state; such states have small *quantum Hamming distance* to the ground space. This is an interesting notion in the context of quantum locally testable codes (qLTCs) because low-energy states are equivalent to states with low quantum Hamming distance to the codespace (see [19] for definitions of quantum Hamming distance and qLTCs).
3. Are there applications of our qLWC constructions?
4. There has been a number of recent results about approximate quantum error-correcting codes in a variety of areas including many-body physics [12], the AdS/CFT correspondence [28], and quantum resource theories [27]. Could approximate error-correcting codes play a role in trying to resolve the qPCP and qLDPC conjectures?
5. Eldar and Harrow showed that quantum locally testable codes of the CSS type are NLTS [19]. Can this argument be extended to general qLTCs?

6. Is it possible for qLDPC codes (not necessarily stabilizer or exact error-correcting codes) to be defined as the codespace of a geometrically local Hamiltonian? There are a few no-go results that give limitations on codes living on lattices [13, 21], but they apply to special classes of codes such as stabilizer codes or locally-correctible codes. Our qLWC codes, by contrast, are neither.
7. Could the *combinatorial NLTS conjecture* be easier to prove than the NLTS conjecture? This conjecture posits that there exist a family of local Hamiltonians where states that have non-zero energy penalty on only a small constant fraction of Hamiltonian terms must have non-trivial circuit complexity.

References

1 Scott Aaronson and Greg Kuperberg. Quantum versus classical proofs and advice. In *Computational Complexity, 2007. CCC'07. Twenty-Second Annual IEEE Conference on*, pages 115–128. IEEE, 2007.

2 Dorit Aharonov, Itai Arad, and Thomas Vidick. Guest column: The quantum PCP conjecture. *SIGACT News*, 44(2):47–79, 2013. doi:10.1145/2491533.2491549.

3 Dorit Aharonov and Lior Eldar. Quantum locally testable codes. *SIAM Journal on Computing*, 44(5):1230–1262, 2015. doi:10.1137/140975498.

4 Dorit Aharonov and Tomer Naveh. Quantum NP - a survey, 2002. arXiv:arXiv:quant-ph/0210077.

5 Dorit Aharonov, Wim van Dam, Julia Kempe, Zeph Landau, Seth Lloyd, and Oded Regev. Adiabatic quantum computation is equivalent to standard quantum computation. *SIAM J. Comput.*, 37(1):166–194, 2007. doi:10.1137/S0097539705447323.

6 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998. doi:10.1145/278298.278306.

7 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *J. ACM*, 45(1):70–122, 1998. doi:10.1145/273865.273901.

8 Alexei Ashikhmin, Simon Litsyn, and Michael A Tsfasman. Asymptotically good quantum codes. *Physical Review A*, 63(3):032311, 2001.

9 Dave Bacon, Steven T Flammia, Aram W Harrow, and Jonathan Shi. Sparse quantum codes from quantum circuits. *IEEE Transactions on Information Theory*, 63(4):2464–2479, 2017.

10 Johannes Bausch and Elizabeth Crosson. Analysis and limitations of modified circuit-to-hamiltonian constructions, 2016. arXiv:arXiv:1609.08571.

11 Cédric Bény and Ognyan Oreshkov. General conditions for approximate quantum error correction and near-optimal recovery channels. *Physical review letters*, 104(12):120501, 2010.

12 Fernando GSL Brandao, Elizabeth Crosson, M Burak Şahinoğlu, and John Bowen. Quantum error correcting codes in eigenstates of translation-invariant spin chains. *arXiv preprint arXiv:1710.04631*, 2017.

13 Sergey Bravyi, David Poulin, and Barbara Terhal. Tradeoffs for reliable quantum information storage in 2d systems. *Physical review letters*, 104(5):050503, 2010.

14 Libor Caha, Zeph Landau, and Daniel Nagaj. The Feynman-Kitaev computer's clock: bias, gaps, idling and pulse tuning. *arXiv preprint arXiv:1712.07395*, 2017.

15 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM. doi:10.1145/800157.805047.

16 Claude Crépeau, Daniel Gottesman, and Adam Smith. Approximate quantum error-correcting codes and secret sharing schemes. In *Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'05, pages 285–301, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11426639_17.

17 Elizabeth Crosson and John F Bowen. Quantum ground state isoperimetric inequalities for the energy spectrum of local hamiltonians. *arXiv quant-ph*, 2017. arXiv:1703.10133.

18 Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3), jun 2007. doi:10.1145/1236457.1236459.

19 Lior Eldar and Aram Wettroth Harrow. Local hamiltonians whose ground states are hard to approximate. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 427–438, 2017. doi:10.1109/FOCS.2017.46.

20 Bill Fefferman and Shelby Kimmel. Quantum vs classical proofs and subset verification. *CoRR*, abs/1510.06750, 2015. arXiv:1510.06750.

21 Steven T. Flammia, Jeongwan Haah, Michael J. Kastoryano, and Isaac H. Kim. Limits on the storage of quantum information in a volume of space. *Quantum*, 1:4, apr 2017. doi:10.22331/q-2017-04-25-4.

22 Michael H. Freedman and Matthew B. Hastings. Quantum systems on non-k-hyperfinite complexes: a generalization of classical statistical mechanics on expander graphs. *Quantum Information & Computation*, 14(1-2):144–180, 2014. URL: http://www.rintonpress.com/xxqic14/qic-14-12/0144-0180.pdf.

23 Michael H Freedman, David A Meyer, and Feng Luo. Z2-systolic freedom and quantum codes. *Mathematics of quantum computation, Chapman & Hall/CRC*, pages 287–320, 2002.

24 Daniel Eric. Gottesman. *Stabilizer codes and quantum error correction*. PhD thesis, California Institute of Technology, Pasadena, California, 1997.

25 Matthew B Hastings. Weight reduction for quantum codes. *arXiv preprint arXiv:1611.03790*, 2016.

26 Matthew B. Hastings. Quantum codes from high-dimensional manifolds. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 25:1–25:26, 2017. doi:10.4230/LIPIcs.ITCS.2017.25.

27 Patrick Hayden and Geoffrey Penington. Approximate quantum error correction revisited: Introducing the alphabit. *arXiv preprint arXiv:1706.09434*, 2017.

28 Isaac H Kim and Michael J Kastoryano. Entanglement renormalization, quantum error correction, and bulk causality. *Journal of High Energy Physics*, 2017(4):40, 2017.

29 A.Y. Kitaev, A. Shen, and M.N. Vyalyi. *Classical and Quantum Computation*. Graduate studies in mathematics. American Mathematical Society, 2002.

30 A.Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003. doi:10.1016/S0003-4916(02)00018-0.

31 Debbie W Leung, Michael A Nielsen, Isaac L Chuang, and Yoshihisa Yamamoto. Approximate quantum error correction can lead to better codes. *Physical Review A*, 56(4):2567, 1997.

32 L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

33 Jean-Pierre Tillich and Gilles Zémor. Quantum ldpc codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Transactions on Information Theory*, 60(2):1193–1202, 2014.

Fully Dynamic MIS in Uniformly Sparse Graphs

Krzysztof Onak
IBM Research, TJ Watson Research Center, Yorktown Heights, New York, USA

Baruch Schieber
IBM Research, TJ Watson Research Center, Yorktown Heights, New York, USA

Shay Solomon[1]
IBM Research, TJ Watson Research Center, Yorktown Heights, New York, USA

Nicole Wein[2]
Massachusetts Institute of Technology, Cambridge, Massachusetts, USA

Abstract

We consider the problem of maintaining a maximal independent set (MIS) in a dynamic graph subject to edge insertions and deletions. Recently, Assadi, Onak, Schieber and Solomon (STOC 2018) showed that an MIS can be maintained in *sublinear* (in the dynamically changing number of edges) amortized update time. In this paper we significantly improve the update time for *uniformly sparse graphs*. Specifically, for graphs with arboricity α, the amortized update time of our algorithm is $O(\alpha^2 \cdot \log^2 n)$, where n is the number of vertices. For low arboricity graphs, which include, for example, minor-free graphs as well as some classes of "real world" graphs, our update time is polylogarithmic. Our update time improves the result of Assadi *et al.* for all graphs with arboricity bounded by $m^{3/8-\epsilon}$, for any constant $\epsilon > 0$. This covers much of the range of possible values for arboricity, as the arboricity of a general graph cannot exceed $m^{1/2}$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Theory of computation → Graph algorithms analysis, Theory of computation → Dynamic graph algorithms

Keywords and phrases dynamic graph algorithms, independent set, sparse graphs, graph arboricity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.92

Related Version A full version of the paper is available at [23].

1 Introduction

The importance of the maximal independent set (MIS) problem is hard to overstate. In general, MIS algorithms constitute a useful subroutine for locally breaking symmetry between several choices. The MIS problem has intimate connections to a plethora of fundamental combinatorial optimization problems such as maximum matching, minimum vertex cover, and graph coloring. As a prime example, MIS is often used in the context of graph coloring, as all vertices in an independent set can be assigned the same color. As another important example, one can compute a large matching (approximating the maximum matching to within a factor arbitrarily close to 1) by applying maximal independent sets of longer and longer augmenting paths as observed by Hopcroft and Karp [11]. The seminal papers of Luby [18] and Linial [17] discuss additional applications of MIS. A non-exhaustive list of further direct and indirect

[1] Supported by the IBM Herman Goldstine Postdoctoral Fellowship.
[2] Supported by an NSF Graduate Fellowship.

applications of MIS includes resource allocation [25], leader election [7], the construction of network backbones [16, 12], and sublinear-time approximation algorithms [22].

In the 1980s, questions concerning the computational complexity of the MIS problem spurred a line of research that led to the celebrated parallel algorithms of Luby [18], and Alon, Babai, and Itai [1]. These algorithms find an MIS in $O(\log n)$ rounds without global coordination. More recently, Fischer and Noever [8] gave an even simpler greedy algorithm that considers vertices in random order and takes $O(\log n)$ rounds with high probability (see also an earlier result of Blelloch, Fineman, and Shun [4]).

In this work we continue the study of the MIS problem by considering the *dynamic setting*, where the underlying graph is not fixed, but rather evolves over time via edge updates. Formally, a *dynamic graph* is a graph sequence $\mathcal{G} = (G_0, G_1, \ldots, G_M)$ on n fixed vertices, where the initial graph is $G_0 = (V, \emptyset)$ and each graph $G_i = (V, E_i)$ is obtained from the previous graph G_{i-1} in the sequence by either adding or deleting a single edge. The basic goal in this context is to maintain an MIS in time significantly faster than it takes to recompute it from scratch following every edge update.

In STOC'18, Assadi, Onak, Schieber, and Solomon [2] gave the first sub-linear (amortized) update time fully dynamic algorithm for maintaining a MIS. Their amortized update time in $\min\{m^{3/4}, \Delta\}$, where m is the (dynamically changing) number of edges, and Δ is a fixed bound on the maximum degree of the graph. For graphs of high maximum degree, the update time of the algorithm of [2] decreases as the graph becomes sparser.

1.1 Our contribution

We focus on graphs that are "uniformly sparse" or "sparse everywhere", as opposed to the previous work by Assadi et al. [2] that considers unrestricted sparse graphs. We aim to improve the update time of [2] as a function of the "uniform sparsity" of the graph. This fundamental property of graphs has been studied in various contexts and names over the years, one of which is via the notion of *arboricity* [19, 20, 24]:

▶ **Definition 1.1.** The arboricity α of a graph $G = (V, E)$ is defined as $\alpha = \max_{U \subset V} \lceil \frac{|E(U)|}{|U|-1} \rceil$, where $E(U) = \{(u, v) \in E \mid u, v \in U\}$.

Thus a graph has bounded arboricity if every induced subgraph has bounded density. The family of low arboricity graphs contains, among others, bounded-degree graphs, all minor-closed graph classes (e.g., planar graphs, graphs with bounded treewidth), and randomly generated preferential attachment graphs. Moreover, it is believed that many real-world graphs such as the world wide web graph and social networks also have low arboricity [9].

A dynamic graph of *arboricity* α is a dynamic graph such that all graphs G_i have arboricity bounded by α. We prove the following result.

▶ **Theorem 1.2.** *For any dynamic n-vertex graph of arboricity α, an MIS can be maintained deterministically in $O(\alpha^2 \log^2 n)$ amortized update time.*

Theorem 1.2 improves the result of Assadi et al. for all graphs with arboricity bounded by $m^{3/8-\epsilon}$, for any constant $\epsilon > 0$. This covers much of the range of possible values for arboricity, as the arboricity of a general graph cannot exceed $\sqrt{m}$. Furthermore, we obtain polylogarithmic update time for graphs of polylogarithmic arboricity; in particular, for the family of constant arboricity graphs the update time is $O(\log^2 n)$.

1.2 Our and previous techniques

1.2.1 The dynamic edge orientation problem

Our algorithm utilizes two properties of arboricity α graphs: a) every subgraph contains a vertex of degree at most 2α, and b) there exists an orientation of the edges so that every vertex has out-degree at most α. The first property follows from Definition 1.1 and the second property is due to an alternate definition [20].

Brodal and Fagerberg [5] initiated the study of the dynamic edge orientation problem and gave an algorithm that maintains an $O(\alpha)$ out-degree orientation in amortized $O(\alpha + \log n)$ time; our algorithm uses the algorithm of [5] as a *black box*. (Refer to [14, 10, 13, 3] for additional results on the dynamic edge orientation problem.)

1.2.2 A comparison to Assadi et al.

As noted already by Assadi et al. [2], a central obstacle in maintaining an MIS in a dynamic graph is the maintenance of a detailed 2-hop neighborhood of a vertex. We need this information to update the MIS. Consider the "hard" case that an edge is added to the graph and as a result, one of its endpoints v is removed from the MIS. In this case, to maintain the maximality of the MIS we need to identify and add to the MIS all neighbors of v that are not adjacent to a vertex in the MIS. This means that for each of v's neighbors, we need to know whether it has a neighbor the MIS (other than v). Alas, dynamically maintaining the 2-neighborhood of each vertex *explicitly* is prohibitive.

To overcome this hurdle, we build upon the approach of Assadi et al. [2] and maintain an *incomplete* 2-hop neighborhood. Consequently, we may err by adding vertices to the MIS even though they are adjacent to MIS vertices. To fix this, we need to remove vertices from the MIS. The important property that we maintain, following [2], is that the number of added vertices is significantly higher than the number of removed vertices. Like Assadi et al., we use a potential function defined as the number of vertices not in the MIS. To amortize the update time, if an update takes a long time, then the size of the MIS increases substantially as a result. On the other hand, the size of the MIS can only decrease by one in each update. Our algorithm deviates significantly from Assadi et al. in several respects, described next.

I: An underlying bounded out-degree orientation. We apply the algorithm of [5] for efficiently maintaining a bounded out-degree orientation. Given such an orientation, we can maintain 2-hop neighborhoods with respect to *outgoing edges* explicitly, which helps significantly in the maintenance of the MIS. More specifically, the usage of a bounded out-degree orientation enables us to reduce the problem to a single nontrivial case.

II: An intricate "chain reaction". To handle the nontrivial case efficiently, we develop an intricate "chain reaction" process, initiated by adding vertices to the MIS that violate the independence property, which then forces the removal of other vertices from the MIS, which in turn forces the addition of other vertices, and so forth. Such a process may take a long time. However, a novel partition of a subset of the in-neighborhood of each vertex in the MIS into a logarithmic number of "buckets" (see below) together with a careful analysis allow for limiting this chain reaction to a logarithmic number of phases; upper bounding the number of phases is crucial for achieving a low update time. We note that such a careful analysis was not required in [2], where the chain reaction was handled by a simple recursive treatment.

III: A precise bucketing scheme. In order to achieve a sufficient increase in the size of the MIS, we need to carefully choose which vertices to add to the MIS so as to guarantee that at every step of the aforementioned chain reaction, even steps far in the future, we will be able to add enough vertices to the MIS. We achieve this by maintaining a precise *bucketing* of a carefully maintained subset of the in-neighborhood of each vertex, where vertices in larger-indexed buckets are capable of setting off longer and more fruitful chain reactions. At the beginning of the chain reaction, we add vertices in the larger-indexed buckets to the MIS, and gradually allow for vertices in smaller-indexed buckets to be added.

IV: A tentative set of MIS vertices. In contrast to the algorithm of Assadi et al., here we cannot iteratively choose which vertices to add to the MIS. Instead, we need to build a *tentative* set of vertices to add to the MIS, and only later prune this set to choose which vertices to add. If we do not carefully select vertices added to the MIS, we may not increase the size of the MIS sufficiently in order to amortize the cost of this process. To make sure that the size of the tentative set remains sufficiently large after the pruning, we make critical use of the first property of low arboricity graphs mentioned in Section 1.2.1. More details can be found at the beginning of Section 4.1.

1.2.3 A comparison to other previous work

Censor-Hillel, Haramaty, and Karnin [6] consider the problem of maintaining MIS in the dynamic distributed setting. They show that there is a randomized algorithm that requires only a constant number of rounds in expectation to update the maintained MIS, as long as the sequence of graph updates does not depend on the algorithm's randomness. This assumption is often referred to as *oblivious adversary*. As noted by Censor-Hillel et al., it is unclear whether their algorithm can be implemented with low total work in the centralized setting. This shortcoming is addressed by Assadi et al. [2] and by the current paper.

Kowalik and Kurowski [15] employ a dynamic bounded out-degree orientation to answer shortest path queries in (unweighted) planar graphs. Specifically, given a planar graph and a constant k, Kowalik and Kurowski maintain a data structure that can check in constant time whether two vertices are at distance at most k and if so produce a path of such length. This data structure is fully dynamic with polylogarithmic amortized update time. Like our data structure, their data structure maintains information on the 2-hop neighborhoods of the vertices, however, the nature of the 2-hop neighborhood information necessary for the two problems is different. For answering shortest path queries, one needs to maintain the *complete* 2-hop neighborhood. Whereas, to maintain an MIS, we only need to maintain *partial* information about the 2-hop neighborhood, however, the information that we store must be more detailed in the sense that for each 2-hop neighbor that we store, we need to know which 1-hop neighbors it is adjacent to. This is necessary because when we remove a vertex v from the MIS, we need to know which of v's neighbors have no neighbors in the MIS in order to know which vertices we need to add to the MIS.

1.3 Dynamic MIS vs. dynamic maximal matching

In the maximal matching problem, the goal is to compute a matching that cannot be extended by adding another edge. The problem is equivalent to finding an MIS in the line graph of the input graph. However, despite this very close relationship between the MIS and maximal matching problems, (efficiently) *maintaining* an MIS appears to be inherently harder than maintaining a maximal matching. As a first potential evidence, one may notice that there

is a significant gap in the performance of the naive algorithms for these dynamic problems. For the maximal matching problem, the naive algorithm works in $O(\Delta)$ time. For the MIS problem, the naive algorithm has to inspect not only neighbors of a vertex v that is being removed from MIS, but also their neighbors (i.e. its 2-hop neighborhood) in order to find out which neighbors need to be added to the MIS; as a result the update time is $O(m)$. Furthermore, the worst-case number of MIS changes (by any algorithm) may be as high as $\Omega(\Delta)$ [6, 2], whereas the worst-case number of changes to the maximal matching maintained by the naive algorithm is $O(1)$. Also, the available body of work on the dynamic MIS problem is significantly sparser than for the dynamic maximal matching problem.

It is therefore plausible that it may be hard to obtain, for MIS, a bound better than the best bounds known for the dynamic maximal matching problem. In particular, the state-of-the-art dynamic deterministic algorithm for maintaining a maximal matching has an update time of $O(\sqrt{m})$ [21], even in the amortized sense. Hence, in order to obtain update time bounds sub-polynomial in m, one may have to exploit the structure of the graph, and bounded arboricity graphs are a natural candidate. A maximal matching can be maintained in graphs of arboricity bounded by α with amortized update time $O(\alpha + \sqrt{\alpha \lg n})$ [21, 10]; as long as the arboricity is polylogarithmic in n, the amortized update time for maintaining a maximal matching is polylogarithmic. In this work we show that essentially the same picture applies to the seemingly harder problem of dynamic MIS.

2 Algorithm overview

Using a bounded out-degree orientation of the edges, a very simple algorithm suffices to handle edge updates that fall into certain cases. The nontrivial case occurs when we remove a vertex v from the MIS and need to determine which vertices in v's in-neighborhood have no neighbors in the MIS, and thus need to be added to the MIS. The in-neighborhood of v could be very large and it would be costly to spend even constant time per in-neighbor of v. Furthermore, it would be costly to maintain a data structure that stores for each vertex in the MIS which of its in-neighbors have no other neighbors in the MIS. Suppose we stored such a data structure for a vertex v. Then the removal of a vertex u from the MIS could cause the entirety of the common neighborhood of u and v to change their status in v's data structure. If this common neighborhood is large, then this operation is costly.

To address this issue, our algorithm does not even attempt to determine the exact set of neighbors of v that need to be added to the MIS. Instead, we maintain *partial* information about which vertices will need to be added to the MIS. Then, when we are unsure about whether we need to add a specific vertex to the MIS, we simply add it to the MIS and remove its conflicting neighbors from the MIS, which triggers a chain reaction of changes to the MIS. Despite the fact that this chain reaction may take a long time to resolve, we obtain an amortized time bound by using a potential function: the number of vertices not in the MIS. That is, we ensure that if we spend a lot of time processing an edge update, then the size of the MIS increases substantially as a result.

The core of our algorithm is to carefully choose which vertices to add to the MIS at each step of the chain reaction to ensure that the size of the MIS increases sufficiently. To accomplish this, we store an intricate data structure for each vertex which includes a partition of a subset of its in-neighborhood into numbered buckets. The key idea is to ensure that whenever we remove a vertex from the MIS, it has at least one full bucket of vertices, which we add to the MIS.

When we remove a vertex from the MIS whose top bucket is full of vertices, we begin the chain reaction by adding these vertices to the MIS (and removing the conflicting vertices from the MIS). In each subsequent step of the chain reaction, we process more vertices, and for each processed vertex, we add to the MIS the set of vertices in its *topmost full bucket*. To guarantee that every vertex that we process has at least one full bucket, we utilize an invariant (the "Main Invariant") which says that for all i, when we process a vertex whose bucket i is full, then in the next iteration of the chain reaction we will only process vertices whose bucket $i-1$ is full. This implies that if the number of iterations in the chain reaction is at most the number of buckets, then we only process vertices with at least one full bucket. To bound the number of iterations of the chain reaction, we prove that the number of processed vertices *doubles* at every iteration. This way, there cannot be more than a logarithmic number of iterations. Thus, by choosing the number of buckets to be logarithmic, we only process vertices with at least one full bucket, which results in the desired increase in the size of the MIS.

3 Algorithm setup

Let $\mathcal{M}$ be the MIS that we maintain. Our algorithm uses a dynamic edge orientation algorithm as a black box.

For each vertex v, let $N(v)$ denote the neighborhood of v, let $N^+(v)$ denote the out-neighborhood of v, and let $N^-(v)$ denote the in-neighborhood of v.

3.1 The trivial cases

For certain cases of edge updates, there is a simple algorithm to update the $\mathcal{M}$. Here, we introduce this simple algorithm and then describe the case that this algorithm does not cover.

▶ **Definition 3.1.** We say that a vertex v is *resolved* if either v is in $\mathcal{M}$ or a vertex in $N^-(v)$ is in $\mathcal{M}$. Otherwise we say that v is *unresolved*.

The data structure is simply that each vertex v stores a partition of its in-neighborhood into resolved vertices and unresolved vertices. To maintain this data structure, whenever a vertex v enters or exits $\mathcal{M}$, v notifies its 2-hop out-neighborhood.

DELETE(U,V):
- It cannot be the case that both u and v are in $\mathcal{M}$ since $\mathcal{M}$ is an independent set.
- If neither u nor v is in $\mathcal{M}$ then both must already have neighbors in $\mathcal{M}$ and we do nothing.
- If $u \in \mathcal{M}$ and $v \notin \mathcal{M}$, then we may need to add v to $\mathcal{M}$. If v is resolved, we do not add v to $\mathcal{M}$. Otherwise, we scan $N^+(v)$ and if no vertex in $N^+(v)$ is in $\mathcal{M}$, we add v to $\mathcal{M}$.

INSERT(U,V):
- If it is not the case that both u and v are in $\mathcal{M}$, then we do nothing and $\mathcal{M}$ remains maximal.
- If both u and v are in $\mathcal{M}$ we remove v from $\mathcal{M}$. Now, some of v's neighbors may need to be added to $\mathcal{M}$, specifically, those with no neighbors in $\mathcal{M}$. For each unresolved vertex $w \in N^+(v)$, we scan $N^+(w)$ and if $N^+(w) \cap \mathcal{M} = \emptyset$, then we add w to $\mathcal{M}$. For each resolved vertex $w \in N^-(v)$, we know not to add w to $\mathcal{M}$. On the other hand, for each unresolved vertex $w \in N^-(v)$, we do not know whether to add w to $\mathcal{M}$ and it could be costly to scan $N^+(w)$ for all such w. This simple algorithm does not handle the case where v has many unresolved in-neighbors.

In summary, the nontrivial case occurs when we delete a vertex v from $\mathcal{M}$ and v has many unresolved in-neighbors.

3.2 Data structure

As in the trivial cases, each vertex v maintains a partition of $N^-(v)$ into resolved vertices and unresolved vertices. In addition, we further refine the set of unresolved vertices. One important subset of the unresolved vertices in $N^-(v)$ is the *active set of* v, denoted A_v. As motivated in the algorithm overview, A_v is partitioned into b buckets $A_v(1), \ldots, A_v(b)$ each of size at most s. We will set b and s so that $b = \Theta(\log n)$ and $s = \Theta(\alpha)$.

The purpose of maintaining A_v is to handle the event that v is removed from $\mathcal{M}$. When v is removed from $\mathcal{M}$, we use the partition of A_v into buckets to carefully choose which neighbors of v to add to $\mathcal{M}$ to begin a chain reaction of changes to $\mathcal{M}$. For the rest of the vertices in A_v, we scan through them and update the data structure to reflect the fact that $v \notin \mathcal{M}$. This scan of A_v is why it is important that each active set is small (size $O(\alpha \log n)$).

One important property of active sets is that each vertex is in the active set of at most one vertex. For each vertex v, let $a(v)$ denote the vertex whose active set contains v. Let $B(v)$ denote the bucket of $A_{a(v)}$ that contains v.

For each vertex v the data structure maintains the following partition of $N^-(v)$:

- Z_v is the set of resolved vertices in $N^-(v)$.
- A_v (*the active set*) is a subset of the unresolved vertices in $N^-(v)$ partitioned into $b = \Theta(\log n)$ buckets $A_v(1), ..., A_v(b)$ each of size at most $s = \Theta(\alpha)$. A_v is empty if $v \notin \mathcal{M}$.
- P_v (*the passive set*) is the set of unresolved vertices in $N^-(v)$ in the active set of some vertex other than v. P_v is partitioned into b buckets $P_v(1), ..., P_v(b)$ such that each vertex $u \in P_v$ is in the set $P_v(i)$ if and only if $B(u) = i$.
- R_v (*the residual set*) is the set of unresolved vertices in $N^-(v)$ not in the active set of any vertex.

We note that while Z_v depends only on $\mathcal{M}$ and the orientation of the edges, the other three sets depend on internal choices made by the algorithm. In particular, for each vertex v, the algorithm picks at most one vertex $a(v)$ for which $v \in A_{a(v)}$ and this choice uniquely determines for every vertex $u \in N^+(v)$, which set (A_u, P_u, or R_u) v belongs to.

We now outline the purpose of the passive set and the residual set. Suppose a vertex v is removed from $\mathcal{M}$. We do not need to worry about the vertices in P_v because we know that all of these vertices are in the active set of a vertex in $\mathcal{M}$ and thus none of them need to be added to $\mathcal{M}$. On the other hand, we do not know whether the vertices in R_v need to be added to $\mathcal{M}$. We cannot afford to scan through them all and we cannot risk not adding them since this might cause $\mathcal{M}$ to not be maximal. Thus, we add them all to $\mathcal{M}$ and set off a chain reaction of changes to $\mathcal{M}$. That is, even though our analysis requires that we carefully choose which vertices of A_v to add to $\mathcal{M}$ during the chain reaction, it suffices to simply add every vertex in R_v to $\mathcal{M}$ (except for those with edges to other vertices we are adding to $\mathcal{M}$).

3.3 Invariants

We maintain several invariants of the data structure. The invariant most central to the overall argument is the Main Invariant (Invariant 6), whose purpose is outlined in the algorithm overview. The first four invariants follow from the definitions.

▶ **Invariant 1.** (Resolved Invariant). *For all resolved vertices v, for all $u \in N^+(v)$, v is in Z_u.*

▶ **Invariant 2.** (Orientation Invariant). *For all v, $Z_v \cup A_v \cup P_v \cup R_v = N^-(v)$.*

▶ **Invariant 3.** (Empty Active Set Invariant). *For all $v \notin \mathcal{M}$, A_v is empty.*

▶ **Invariant 4.** (Consistency Invariant).

- *If v is resolved, then for all vertices $u \in N^+(v)$, $v \notin A_u$.*
- *If v is unresolved then v is in A_u for at most one vertex $u \in N^+(v)$.*
- *If v is in the active set of some vertex u, then for all $w \in N^+(v) \setminus \{u\}$, v is in $P_w(i)$ where i is such that $B(v) = i$.*
- *If v is in the residual set for some vertex u, then for all $w \in N^+(v)$, v is in R_w.*

The next invariant says that the active set of a vertex is filled from lowest bucket to highest bucket and only then is the residual set filled.

▶ **Definition 3.2.** We say that a bucket $A_v(i)$ is *full* if $|A_v(i)| = s$. We say A_v is *full* if all b of its buckets are full.

▶ **Invariant 5.** (Full Invariant). *For all vertices v and all $i < b$, if $A_v(i)$ is not full then $A_v(i+1)$ is empty. Also, if A_v is not full then R_v is empty.*

The next invariant, the Main Invariant, says that if we were to move v from $A_{a(v)}$ to the active set of a different vertex u by placing v in the lowest non-full bucket of A_u, then $B(v)$ would not decrease.

▶ **Invariant 6.** (Main Invariant). *For all v, if $B(v) = i > 1$ then for all $u \in N^+(v) \cap \mathcal{M}$, $A_u(i-1)$ is full.*

4 Algorithm

The algorithm works in four phases.

1. Update $\mathcal{M}$.
2. Update the data structure.
3. Run a black-box edge orientation algorithm.
4. Update the data structure.

The data structure is completely static during phases 1 and 3.

4.1 Updating $\mathcal{M}$

When an edge (u, v) is deleted, we run the procedure DELETE(U,V) specified in the trivial cases section. When an edge (u, v) is inserted and it is not the case that both u and v are in $\mathcal{M}$, then we do nothing and $\mathcal{M}$ remains maximal. In the case that both u and v are in $\mathcal{M}$, we need to remove either u or v from $\mathcal{M}$ which may trigger many changes to $\mathcal{M}$.

The procedure of updating $\mathcal{M}$ happens in two stages. In the first stage, we iteratively build two sets of vertices, S^+ and S^-. Intuitively, S^+ is a subset of vertices that we intend to add to $\mathcal{M}$ and S^- is the set of vertices that we intend to delete from $\mathcal{M}$. The aforementioned chain reaction of changes to $\mathcal{M}$ is captured in the construction of S^+ and S^-. In the second stage we make changes to $\mathcal{M}$ according to S^+ and S^-. In particular, the set of vertices that we add to $\mathcal{M}$ contains a large subset of S^+ as well as some vertices not in S^+ and the set of vertices that we remove from $\mathcal{M}$ is a subset of S^-. In accordance with our goal of increasing

the size of $\mathcal{M}$ substantially, we ensure that S^+ is much larger than S^-. The most interesting part of updating $\mathcal{M}$ is captured in the first stage. We defer the description of the second stage to the full version [23].

Why is it important to build S^+ before choosing which vertices to add to $\mathcal{M}$? The answer is that it is important that we add a *large* subset of S^+ to $\mathcal{M}$ since our goal is to increase the size of $\mathcal{M}$ substantially. We find this large subset of S^+ by finding a large MIS in the graph induced by S^+, which exists (and can be found in linear time) because the graph has bounded arboricity (see the full version for details [23]). Suppose that instead of iteratively building S^+, we tried to iteratively add vertices directly to $\mathcal{M}$ in a greedy fashion. This could result in only very few vertices successfully being added to $\mathcal{M}$. For example, if we begin by adding the center of a star graph to $\mathcal{M}$ and subsequently try to add the leaves of the star, we will not succeed in adding any of the leaves to $\mathcal{M}$. On the other hand, if we first add the vertices of the star to S^+ then we can find a large MIS in the star (the leaves) to add it to $\mathcal{M}$.

For the rest of this section we consider an edge insertion (u, v).

4.1.1 Stage 1: Constructing S^+ and S^-

A key property of the construction is that S^+ is considerably larger than S^-:

▶ **Lemma 4.1.** *If* $|S^-| > 1$ *then* $|S^+| \geq 4\alpha|S^-|$.

After constructing S^+ and S^- we will add at least $\frac{|S^+|}{2\alpha}$ vertices to $\mathcal{M}$ and remove at most $|S^-|$ vertices from $\mathcal{M}$. Thus, Lemma 4.1 implies that $\mathcal{M}$ increases by $\Omega(\frac{|S^+|}{\alpha})$.

To construct S^+ and S^-, we define a recursive procedure Process(w) which adds at least one full bucket of A_w to S^+. A key idea in the analysis is to guarantee that for every call to Process(w), A_w indeed has at least one full bucket.

Algorithm description

We say that a vertex $w \in S^-$ has been *processed* if Process(w) has been called and otherwise we say that w is *unprocessed*. We maintain a partition of S^- into the processed set and the unprocessed set and we maintain a partition of the set of unprocessed vertices w into two sets based on whether A_w is full or not. We also maintain a queue $\mathcal{Q}$ of vertices to process, which is initially empty. Recall that (u, v) is the inserted edge and both u and v are in $\mathcal{M}$. The algorithm is as follows.

First, we add v to S^-. Then, if A_v is not full, we terminate the construction of S^+ and S^-. Otherwise, we call Process(v).

Process(w):

1. If A_w is full, then add all vertices in $A_w(b) \cup R_w$ to S^+. If A_w is not full, then let i be the largest full bucket of A_w and add all vertices in $A_w(i)$ and $A_w(i+1)$ to S^+. We will claim that such an i exists (Lemma 4.2).
2. For all vertices x added to S^+ in this call to Process, we add $N^+(x) \cap \mathcal{M}$ to S^-.
3. If S^- contains an unprocessed vertex x with full A_x, we call Process(x).

When a call to Process terminates, including the recursive calls, we check whether Lemma 4.1 is satisfied (that is, whether $|S^+| \geq 4\alpha|S^-|$), and if so, we terminate. Otherwise, if $\mathcal{Q}$ is not empty, we let w be the next vertex in $\mathcal{Q}$ and call Process(w). If $\mathcal{Q}$ is empty we enqueue a new batch of vertices to $\mathcal{Q}$. This batch consists of the set of all unprocessed vertices in S^-. We will claim that such vertices exist (Lemma 4.2).

▶ Remark. The reason we terminate without calling PROCESS(v) if A_v is not full (i.e. R_v is empty) is because R_v is the only set for which we cannot afford to determine whether or not each vertex has another neighbor in $\mathcal{M}$ (besides v): We know that each vertex $w \in Z_v \cup P_v$ has another neighbor in $\mathcal{M}$, and the set A_v is small enough to scan. For the same reason, step 3 of PROCESS is necessary because it ensures that for every vertex w in S^-, all vertices in R_w are in S^+. If this weren't the case and we removed a vertex w in S^- from $\mathcal{M}$, we might be left in the "hard case" of needing to deal with R_w.

Lemma 4.1 follows from the algorithm specification: either the algorithm terminates immediately with $S^- = \{v\}$ or the algorithm terminates according to the termination condition, which is that Lemma 4.1 is satisfied.

Several steps in the algorithm (Step 2 of PROCESS(w) and the last sentence of the algorithm specification) rely on Lemma 4.2:

▶ **Lemma 4.2.**
1. *If we call* PROCESS*(w), then A_w has at least one full bucket.*
2. *Every batch of vertices that we enqueue to $\mathcal{Q}$ is nonempty.*

Proof of Lemma 4.2

Let epoch 1 denote the period of time until the first batch of vertices has been enqueued to $\mathcal{Q}$. For all $i > 1$, let epoch i denote the period of time from the end of epoch $i-1$ to when the i^{th} batch of vertices has been enqueued to $\mathcal{Q}$.

To prove Lemma 4.2, we prove a collection of lemmas that together show that (i) Lemma 4.2 holds for all calls to PROCESS before epoch b ends (recall that b is the number of buckets) and (ii) the algorithm terminates before the end of epoch b.

For all i, let p_i and u_i be the number of processed and unprocessed vertices in S^- respectively, when epoch i ends. Let S_i^+ and S_i^- be the sets S^+ and S^- respectively when epoch i ends. Recall that s is the size of a full bucket. Let $s = 8\alpha$ and let $b = \log_2 n + 1$.

▶ **Lemma 4.3.** *For all $1 \le j \le b$, every time we call* PROCESS*(w) during epoch j, $A_w(b-j+1)$ is full.*

Proof. We proceed by induction on j.
Base case. If $j = 1$ then the algorithm only calls PROCESS(w) on vertices w with full A_w and thus full $A_w(b)$.
Inductive hypothesis. Suppose that during epoch j, all of the processed vertices have full $A_w(b-j+1)$.
Inductive step. We will show that during epoch $j+1$, all of the processed vertices have full $A_w(b-j)$. We first note that during PROCESS(w), the algorithm only adds the vertices in the topmost full bucket of A_w to S^+. Thus, the inductive hypothesis implies that at the end of epoch j for all vertices $x \in S^+$, $B(x) \ge b-j+1$.

Then, by the Main Invariant, at the end of epoch j, for all $x \in S^+$ and all $y \in N^+(x) \cap \mathcal{M}$, $A_y(b-j)$ is full. By construction, the only vertices in S^- other than v are those in $N^+(x) \cap \mathcal{M}$ for some $x \in S^+$. Thus, at the end of epoch j, for all vertices $y \in S^-$, $A_y(b-j)$ is full. During epoch $j+1$, the set of vertices that we process consists only of vertices w that are either in S^- at the end of epoch j or have full A_w. We have shown that all of these vertices w have full $A_w(b-j)$. ◀

▶ **Lemma 4.4.** *For all $1 \le j \le b$, $|S_j^+| \ge p_j s$.*

Proof. By Lemma 4.3, for all calls to PROCESS(w) until the end of epoch j, A_w has at least one full bucket. During each call to PROCESS(w), the algorithm adds at least one full bucket (of size s) of A_w to S^+. By the Consistency Invariant, (i) every vertex is in the active set of at most one vertex and (ii) if a vertex w appears in the active set of some vertex, then w is not in the residual set of any vertex. The only vertices added to S^+ are those in some active set or some residual set, so every vertex in some active set that is added to S^+, is added at most once. Thus, for each processed vertex, there are at least s distinct vertices in S^+. ◀

▶ **Lemma 4.5.** *For all $1 \leq j \leq b$, $p_j < u_j$. That is, there are more unprocessed vertices than processed vertices.*

Proof. At the end of epoch j, Lemma 4.1 is not satisfied because if it were then the algorithm would have terminated. That is, $|S_j^+| < 4\alpha|S_j^-|$. Combining this with Lemma 4.4 and the fact that $p_j + u_j = |S_j^-|$, we have $p_j s < 4\alpha(p_j + u_j)$. Choosing $s = 8\alpha$ completes the proof. ◀

▶ **Lemma 4.6.** *For all $1 \leq j \leq b$, $p_j > 2p_{j-1}$. That is, the number of processed vertices more than doubles during each epoch.*

Proof. At the end of epoch $j-1$, we add all unprocessed vertices to $\mathcal{Q}$. As a result of calling PROCESS on each vertex in $\mathcal{Q}$, the number of processed vertices increases by u_{j-1} by the end of epoch j. That is, $p_j \geq p_{j-1} + u_{j-1}$. By Lemma 4.5, $p_{j-1} < u_{j-1}$, so $p_j > 2p_{j-1}$. ◀

We apply these lemmas to complete the proof of Lemma 4.2:

1. In epoch 1 we process at least one vertex, so $p_1 \geq 1$. By Lemma 4.6, $p_j > 2p_{j-1}$. Thus, $p_j \geq 2^{j-1}$. If $j = b = \log_2 n + 1$, then $p_j > n$, a contradiction. Thus, the algorithm never reaches the end of epoch b. Then, by Lemma 4.3, every time we call PROCESS(w), A_w has at least one full bucket.
2. Suppose by way of contradiction that we enqueue no vertices to $\mathcal{Q}$ at the end of some epoch $1 \leq j \leq b$. Then, $u_j = 0$. By Lemma 4.5, $p_j < u_j$, so $p_j < 0$, a contradiction.

5 Analysis

In this section we present the most interesting part of the analysis and defer the rest of the analysis to the full version [23]. When a vertex v is added to $\mathcal{M}$, A_v is empty and needs to be populated in order to satisfy the Main Invariant. This process is the bottleneck of the runtime and we analyze it here.

We begin by analyzing the runtime of two basic processes that happen while updating the data structure: adding a vertex to some active set (Lemma 5.1) and removing a vertex from some active set (Lemma 5.2).

Recall that our algorithm uses a dynamic edge orientation algorithm as a black box. Let T be the amortized update time of this algorithm and let D be the out-degree of the orientation. Ultimately, we will apply the algorithm of Brodal and Fagerberg [5] to get $D = O(\alpha)$ and $T = O(\alpha + \log n)$.

▶ **Lemma 5.1.** *Suppose vertex v is not in any active set. Adding v to some active set and updating the data structure accordingly takes time $O(D)$.*

Proof. When we add a vertex v to some $A_u(i)$, for all $w \in N^+(v)$ this could causes a violation to the Consistency Invariant. To remedy this, it suffices to remove v from whichever set it was previously in with respect to w (which is not A_w) and add it to $P_w(i)$. ◀

▶ **Lemma 5.2.** *Removing a vertex v from some active set A_u and updating the data structure accordingly takes time $O(D \log n)$.*

Proof. When we remove a vertex v from some $A_u(i)$, this leaves bucket $A_u(i)$ not full so the Full Invariant might be violated. To remedy this, we move a vertex, the *replacement vertex*, from a higher bucket or the residual set to $A_u(i)$. That is, the replacement vertex w is chosen to be any arbitrary vertex from $R_u \cup A_u(i+1) \cup \cdots \cup A_u(b) \cup P_u(i+1) \cup \cdots \cup P_u(b)$. We can choose a vertex from this set in constant time by maintaining a list of all non-empty $P_x(i)$ and $A_x(i)$ for each vertex x. If w is chosen from P_u, we remove w from $A_{a(w)}$ before adding w to $A_u(i)$.

The removal of w from its previous bucket in its previous active set may leave this bucket not full, so again the Full Invariant might be violated and again we remedy this as described above, which sets off a chain reaction. The chain reaction terminates when either there does not exist a viable replacement vertex or until the replacement vertex comes from the residual set. Since the number of the bucket that we choose the replacement vertex from increases at every step of this process, the length of this chain reaction is at most b.[3]

For each vertex v that we add to an active set, we have already removed v from its previous active set, so Lemma 5.1 applies. Overall, we move at most b vertices to a new bucket and by Lemma 5.1, for each of these b vertices we spend time $O(D)$. Thus, the runtime is $O(bD) = O(D \log n)$. ◀

▶ **Lemma 5.3.** *The time to update the data structure in response to a violation of the Main Invariant triggered the addition of a single vertex to $\mathcal{M}$ is $O(D\alpha \log^2 n)$.*

Proof. To satisfy the Main Invariant, we need to populate A_v. We fill A_v in order from bucket 1 to bucket b. First, we add the vertices in R_v until either A_v is full or R_v becomes empty. If R_v becomes empty, then we start adding the vertices of $P_v(i)$ in order from $i = b$ to $i = 1$; however, we only add vertex u to A_v if this causes $B(u)$ to decrease. Once we reach a vertex u in P_v where moving u to the lowest numbered non-full bucket of A_v does not cause $B(u)$ to decrease, then we stop populating A_v. Each time we add a vertex u to A_v from P_v, we first remove u from $A_{a(u)}$ and apply Lemma 5.2. Also, each time we add a vertex to A_v, we apply Lemma 5.1. We note that this method of populating A_v is consistent with the Main Invariant.

We add at most $sb = O(\alpha \log n)$ vertices to A_v and for each one we could apply Lemmas 5.2 and 5.1 in succession. Thus, the total time is $O(\alpha D \log^2 n)$. ◀

References

1 Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986. doi:10.1016/0196-6774(86)90019-2.

2 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proc. 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, 2018.

[3] We note that the length of the chain reaction can be shortened by choosing the replacement vertex from the highest possible bucket. However, we could be forced to choose from bucket $i+1$ (if all buckets higher than $A_u(i+1)$ or $P_u(i+1)$ are empty).

3 Edvin Berglin and Gerth Stølting Brodal. A simple greedy algorithm for dynamic graph orientation. In *Proc. 28th International Symposium on Algorithms and Computation, ISAAC 2017, December 9-12, 2017, Phuket, Thailand*, pages 12:1–12:12, 2017. doi:10.4230/LIPIcs.ISAAC.2017.12.

4 Guy E. Blelloch, Jeremy T. Fineman, and Julian Shun. Greedy sequential maximal independent set and matching are parallel on average. In *Proc. 24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2012, Pittsburgh, PA, USA, June 25-27, 2012*, pages 308–317, 2012. doi:10.1145/2312005.2312058.

5 Gerth Stølting Brodal and Rolf Fagerberg. Dynamic representations of sparse graphs. In *Proc. 6th International Workshop on Algorithms and Data Structures WADS*, pages 342–351. Springer-Verlag, 1999.

6 Keren Censor-Hillel, Elad Haramaty, and Zohar S. Karnin. Optimal dynamic distributed MIS. In *Proc. ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 217–226, 2016. doi:10.1145/2933057.2933083.

7 Sebastian Daum, Seth Gilbert, Fabian Kuhn, and Calvin C. Newport. Leader election in shared spectrum radio networks. In *Proc. ACM Symposium on Principles of Distributed Computing, PODC 2012, Funchal, Madeira, Portugal, July 16-18, 2012*, pages 215–224, 2012. doi:10.1145/2332432.2332470.

8 Manuela Fischer and Andreas Noever. Tight analysis of parallel randomized greedy MIS. In *Proc. 29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2152–2160, 2018. doi:10.1137/1.9781611975031.140.

9 Gaurav Goel and Jens Gustedt. Bounded arboricity to determine the local structure of sparse graphs. In *Proc. Graph-Theoretic Concepts in Computer Science, 32nd International Workshop, WG*, pages 159–167, 2006. doi:10.1007/11917496_15.

10 Meng He, Ganggui Tang, and Norbert Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In *Proc. Algorithms and Computation - 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014*, pages 128–140, 2014. doi:10.1007/978-3-319-13075-0_11.

11 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019.

12 Tomasz Jurdzinski and Dariusz R. Kowalski. Distributed backbone structure for algorithms in the SINR model of wireless networks. In *Proc. Distributed Computing - 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012*, pages 106–120, 2012. doi:10.1007/978-3-642-33651-5_8.

13 Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *Proc. Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Part II*, pages 532–543, 2014. doi:10.1007/978-3-662-43951-7_45.

14 Lukasz Kowalik. Adjacency queries in dynamic sparse graphs. *Inf. Process. Lett.*, 102(5):191–195, 2007. doi:10.1016/j.ipl.2006.12.006.

15 Lukasz Kowalik and Maciej Kurowski. Short path queries in planar graphs in constant time. In *Proc. 35th Annual ACM Symposium on Theory of Computing, STOC*, pages 143–148, 2003.

16 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Initializing newly deployed ad hoc and sensor networks. In *Proc. 10th Annual International Conference on Mobile Computing and Networking, MOBICOM 2004, Philadelphia, PA, USA, September 26 - October 1, 2004*, pages 260–274, 2004. doi:10.1145/1023720.1023746.

17 Nathan Linial. Distributive graph algorithms-global solutions from local data. In *Proc. 28th Annual Symposium on Foundations of Computer Science, FOCS 1987, Los Angeles, California, USA, 27-29 October 1987*, pages 331–335, 1987. doi:10.1109/SFCS.1987.20.

18 Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986. doi:10.1137/0215074.

19 Crispin St.J. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *J. London Math. Soc.*, 36(1):445—450, 1961.

20 Crispin St.J. Nash-Williams. Decomposition of finite graphs into forests. *J. London Math. Soc.*, 39(1):12, 1964.

21 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Proc. Symposium on Theory of Computing Conference, STOC 2013, Palo Alto, CA, USA, June 1-4, 2013*, pages 745–754, 2013. doi:10.1145/2488608.2488703.

22 Huy N. Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 327–336, 2008. doi:10.1109/FOCS.2008.81.

23 Krzysztof Onak, Baruch Schieber, Shay Solomon, and Nicole Wein. Fully dynamic mis in uniformly sparse graphs. *arXiv CoRR*, 2018.

24 William T. Tutte. On the problem of decomposing a graph into n connected factors. *J. London Math. Soc.*, 36(1):221–230, 1961.

25 Dongxiao Yu, Yuexuan Wang, Qiang-Sheng Hua, and Francis C. M. Lau. Distributed (Δ+1)-coloring in the physical model. *Theor. Comput. Sci.*, 553:37–56, 2014. doi:10.1016/j.tcs.2014.05.016.

Strictly Balancing Matrices in Polynomial Time Using Osborne's Iteration

Rafail Ostrovsky[1]
Department of Computer Science, University of California Los Angeles, USA
rafail@cs.ucla.edu

Yuval Rabani[2]
The Rachel and Selim Benin School of Computer Science and Engineering, The Hebrew University of Jerusalem, Israel
yrabani@cs.huji.ac.il

Arman Yousefi[3]
Department of Computer Science, University of California Los Angeles, USA
armany@cs.ucla.edu

Abstract

Osborne's iteration is a method for balancing $n \times n$ matrices which is widely used in linear algebra packages, as balancing preserves eigenvalues and stabilizes their numeral computation. The iteration can be implemented in any norm over $\mathbb{R}^n$, but it is normally used in the L_2 norm. The choice of norm not only affects the desired balance condition, but also defines the iterated balancing step itself.

In this paper we focus on Osborne's iteration in any L_p norm, where $p < \infty$. We design a specific implementation of Osborne's iteration in any L_p norm that converges to a strictly ϵ-balanced matrix in $\tilde{O}(\epsilon^{-2}n^9K)$ iterations, where K measures, roughly, the *number of bits* required to represent the entries of the input matrix.

This is the first result that proves a variant of Osborne's iteration in the L_2 norm (or any L_p norm, $p < \infty$) strictly balances matrices in polynomial time. This is a substantial improvement over our recent result (in SODA 2017) that showed weak balancing in L_p norms. Previously, Schulman and Sinclair (STOC 2015) showed strict balancing of another variant of Osborne's iteration in the L_∞ norm. Their result does not imply any bounds on strict balancing in other norms.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Numerical Linear Algebra, Optimization

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.93

[1] Research supported in part by NSF grant 1619348, DARPA, US-Israel BSF grant 2012366, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. The views expressed are those of the authors and do not reflect position of the Department of Defense or the U.S. Government.
[2] Research supported in part by ISF grant 956-15, by BSF grant 2012333, and by I-CORE Algo.
[3] Research supported in part by NSF grant 1619348, DARPA, US-Israel BSF grant 2012366.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 93; pp. 93:1–93:11

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

1.1 Problem Statement and Motivation

This paper analyzes the convergence properties of Osborne's celebrated iteration [10] for balancing matrices. Given a norm $\|\cdot\|$ in $\mathbb{R}^n$, an $n \times n$ matrix A is balanced if and only if for all i, the i-th row of A and the i-th column of A have the same norm. The problem of balancing a matrix A is to compute a diagonal matrix D such that DAD^{-1} is balanced. The main motivation behind this problem is that balancing a matrix does not affect its eigenvalues, and balancing matrices in the L_2 norm increases the numerical stability of eigenvalue computations [10, 9]. Balancing also has a positive impact on the computational time needed for computing eigenvalues ([9, section 1.4.3]). In practice, it is sufficient to get a good approximation to the balancing problem. For $\alpha \geq 1$, a matrix $B = DAD^{-1}$ is an α-approximation to the problem of balancing A if and only if for all i, the ratio between the maximum and minimum of the norms of the i-th row and column is bounded by α. It is desirable to achieve $\alpha = 1 + \epsilon$ for some small $\epsilon > 0$. A matrix B that satisfies this relaxed balancing condition is also said to be strictly ϵ-balanced.

Osborne's iteration attempts to compute the diagonal matrix D by repeatedly choosing an index i and balancing the i-th row and column (this multiplies the i-th diagonal entry of D appropriately). Osborne proposed this iteration in the L_2 norm, and suggested round-robin choice of index to balance. However, other papers consider the iteration in other norms and propose alternative choices of index to balance [12, 15, 11]. Notice that a change of norm not only changes the target balance condition, but also changes the iteration itself, as in each step a row-column pair is balanced in the given norm. An implementation of Osborne's iteration is used in most numerical algebra packages, including MATLAB, LAPACK, and EISPACK, and is empirically efficient (see [9, 16] for further background). The main theoretical question about Osborne's iteration is its rate of convergence. How many rounds of the iteration are provably sufficient to get a strictly ϵ-balanced matrix?

1.2 Our Results

We consider Osborne's iteration in L_p norms for finite p. We design a new simple choice of the iteration (i.e., a rule to choose the next index to balance), and we prove that this variant provides a polynomial time approximation scheme to the balancing problem. More specifically, we show that in the L_1 norm, our implementation converges to a strictly ϵ-balanced matrix in $O\left(\epsilon^{-2} n^9 \log(wn/\epsilon) \log w / \log n\right)$ iterations, where $\log w$ is a lower bound on the number of bits required to represent the entries of A (exact definitions await Section 2). The time complexity of these iterations is $O\left(\epsilon^{-2} n^{10} \log(wn/\epsilon) \log w\right)$ arithmetic operations over $O(n \log(w/\epsilon))$-bit numbers. This result implies similar bounds for any L_p norm where p is fixed, and in particular the important case of $p = 2$. This is because applying Osborne's iteration in the L_p norm to $A = (a_{ij})_{n \times n}$ is equivalent to applying the iteration in the L_1 norm to $(a_{ij}^p)_{n \times n}$. Of course, the bit representation complexity of the matrix, and thus the bound on the number of iterations, grows by a factor of p.

Our results give the first theoretical analysis that indicates that Osborne's iteration in the L_2 norm, or any L_p norm for finite p, is indeed efficient in the worst case. This partially resolves the question that has been open since 1960 because we actually analyze a variant of Osborne's iteration which is different from the original iteration in the way it picks the next index to balance. Previously, Schulman and Sinclair [15] analyzed yet another variant of Osborne's iteration and answered this question only for the L_∞ norm. Concerning the

convergence rate for the L_p norms discussed here, we recently published a result [11] that considers a much weaker notion of approximation. The previous result only shows the rate of convergence to a matrix that is approximately balanced in an average sense. The matrix might still have row-column pairs that are highly unbalanced. The implementations in the common numerical linear algebra packages use as a stopping condition the strict notion of balancing, and not this weaker notion. We discuss previous work in greater detail below.

1.3 Previous Work

Osborne [10] studied the L_2 norm version of matrix balancing, proved the uniqueness of the L_2 solution, designed the iterative algorithm discussed above, and proved that it converges in the limit to a balanced matrix (without bounding the convergence rate). Parlett and Reinsch [12] generalized Osborne's iteration to other norms. Their implementation is the one widely used in practice (see Chen [3, Section 3.1], also the book [13, Chapter 11] and the code in [1]). Grad [6] proved convergence in the limit for the L_1 version (again without bounding the running time), and Hartfiel [7] showed that the L_1 solution is unique. Eaves et al. [5] gave a characterization of balanceable matrices. Kalantari et al. [8] gave an algorithm for ϵ-balancing a matrix in the L_1 norm. The algorithm reduces the problem to unconstrained convex optimization and uses the ellipsoid algorithm to approximate the optimal solution. This generates a weakly ϵ-balanced matrix, which satisfies the following definition. Given $\epsilon > 0$, a matrix $A = (a_{ij})_{n\times n}$ is weakly ϵ-balanced if and only if $\sqrt{\sum_{i=1}^{n}(\|a_{\cdot,i}\| - \|a_{i,\cdot}\|)^2} \leq \epsilon \cdot \sum_{i,j} |a_{i,j}|$. Compare this with the stronger condition of being strictly ϵ-balanced, which we use in this paper, and numerical linear algebra packages use as a stopping condition. This condition requires that for every $i \in \{1, 2, \ldots, n\}$, $\max\{\|a_{\cdot,i}\|, \|a_{i,\cdot}\|\} \leq (1+\epsilon) \cdot \min\{\|a_{\cdot,i}\|, \|a_{i,\cdot}\|\}$. In L_∞, Schneider and Schneider [14] gave a polynomial time algorithm that exactly balances a matrix. The algorithm does not use Osborne's iteration. Its running time was improved by Young et al. [17]. Both algorithms rely on iterating over computing a minimum mean cycle in a weighted strongly connected digraph, then contracting the cycle. Schulman and Sinclair [15] were the first to provide a quantitative bound on the running time of Osborne's iteration. They proposed a carefully designed implementation of Osborne's iteration in the L_∞ norm that strictly ϵ-balances an $n \times n$ matrix A in $O(n^3 \log(\varrho n/\epsilon))$ iterations, where ϱ measures the initial L_∞ imbalance of A. Their proof is an intricate case analysis. Following that work, in [11] we showed that several implementations of Osborne's iteration in L_p norms, including the original implementation, converge to a weakly ϵ-balanced matrix in polynomial time (which, in fact, can be either nearly linear in number of non-zero entries of matrix A or nearly linear in $1/\epsilon$). Very recently, Cohen et al. [4] gave an interior point algorithm that weakly ϵ-balances matrices in the L_1 norm in time $\tilde{O}(m^{3/2} \log(1/\epsilon))$, where m is the number of non-zero entries of A. Their results also apply to matrix scaling, a problem similar to matrix balancing, for which Allen-Zhu et al. [2] independently gave similar results. Notice that in the results of [4], due to the logarithmic dependence on ϵ one can choose ϵ that is exponentially small in the input representation, and thus get an algorithm for strict balancing with better worst case running time than our claimed analysis of Osborne's iteration. We note, however, that the purpose of our analysis is not to present a new algorithm for matrix balancing, but rather to provide some worst case guarantee for the time it takes some implementation of Osborne's iteration to achieve strict balancing. Osborne's iteration and strict balancing are the heuristic and stopping condition used in practice to balance matrices. We also note that by the lower bound proved in our previous paper [11], Osborne's iteration does not guarantee convergence in time polynomial in $\log(1/\epsilon)$ (the lower bound is $\Omega(1/\sqrt{\epsilon})$).

1.4 Our Contribution

The result of [11] is derived by observing that any implementation of Osborne's iteration can be interpreted as an application of coordinate descent to optimize the convex function from [8]. This is the starting point of this paper, but to make the approach guarantee strict balancing, we need to revise substantially previous implementations using novel algorithmic ideas. Nevertheless, the implementation itself remains a rather simple rule for choosing the next index to balance.

The main difficulty with respect to previous work is the following. The convergence rate of coordinate descent can be bounded effectively as long as there is a choice of coordinate (i.e., index) for which the drop in the objective function in a single step is non-negligible compared with the current objective value. But if this is not the case, then one can argue only about the balance of each index relative to the sum of norms of all rows and columns. Indices that have relatively heavy weight (row norm + column norm) will indeed be balanced at this point. However, light-weight indices may be highly unbalanced. The naive remedy to this problem is to work down by scales. After balancing the matrix globally, heavy-weight indices are balanced, approximately, so they can be left alone, deactivated. Now there are light-weight indices that have become heavy-weight with respect to the remaining active nodes, so we can continue balancing the active indices until the relatively heavy-weight among them become approximately balanced, and so forth. The problem with the naive solution is that balancing the active indices shifts the weights of both active and inactive indices, and they move out of their initial scale. This movement need not complicate the balancing rule—we can keep balancing far-from-balanced nodes in the current scale. However, if the scale sets of indices keep changing, it is harder to argue that the process converges. Shifting between scales is precisely what our algorithm and proof deal with. Light-weight indices that have become heavy-weight are easy to handle. They can keep being active. Heavy-weight indices that have become light-weight cannot continue to be inactive, because they are no longer guaranteed to be approximately balanced. Thus, in order to analyze convergence effectively, we need to bound the number (and global effect on weight) of these reactivation events.

2 Preliminaries

The input is a real square matrix $A = (a_{ij})_{n\times n}$. We denote the i-th row of such a matrix by $a_{i,\cdot}$ and the i-th column by $a_{\cdot,i}$. We also use the notation $[n] = \{1, 2, \ldots, n\}$. The matrix A is *balanced* in the L_p norm iff $\|a_{\cdot,i}\|_p = \|a_{i,\cdot}\|_p$ for every index $i \in [n]$. Since the condition for being balanced depends neither on the signs of the entries of A nor on the diagonal values, we will assume without loss of generality that A is non-negative with zeroes on the diagonal.

An invertible diagonal matrix $D = \mathtt{diag}(d_1, \cdots, d_n)$ *balances* A in the L_p norm iff DAD^{-1} is balanced in the L_p norm. A matrix A is *balanceable* iff there exists an invertible diagonal matrix D that balances A. Balancing a matrix $A = (a_{ij})_{n\times n}$ in the L_p norm is equivalent to balancing the matrix $(a_{ij}^p)_{n\times n}$ in the L_1 norm. Therefore, for the rest of the paper we focus on balancing matrices in the L_1 norm.

We use $a_{\min}$ to denote the minimum non-zero entry of A. We also define $w = \frac{1}{a_{\min}} \cdot \sum_{ij} a_{ij}$.

▶ **Definition 1.** Given $\epsilon > 0$ and an $n \times n$ matrix A, we say that the index i of A (where $i \in [n]$) is *ϵ-balanced* iff

$$\frac{\max\{\|a_{\cdot,i}\|_1, \|a_{i,\cdot}\|_1\}}{\min\{\|a_{\cdot,i}\|_1, \|a_{i,\cdot}\|_1\}} \leq 1 + \epsilon.$$

We say that A is *strictly ϵ-balanced* iff every index i of A is ϵ-balanced.

Any implementation of Osborne's iteration can be thought of as computing vectors $\mathbf{x^{(t)}} \in \mathbb{R}^n$ for $t = 1, 2, \ldots$, where iteration t is applied to the matrix $(a_{ij}^{(t)}) = DAD^{-1}$ for $D = \mathtt{diag}(e^{x_1^{(t)}}, e^{x_2^{(t)}}, \ldots, e^{x_n^{(t)}})$. Thus, for all i, j, $a_{ij}^{(t)} = a_{ij} \cdot e^{x_i^{(t)} - x_j^{(t)}}$. Initially, $\mathbf{x}^{(1)} = (0, 0, \ldots, 0)$. A balancing step of the iteration chooses an index i, then sets $x_i^{(t+1)} = x_i^{(t)} + \frac{1}{2} \cdot \left(\ln \|a_{.,i}^{(t)}\|_1 - \ln \|a_{i,.}^{(t)}\|_1 \right)$, and for all $j \neq i$, keeps $x_j^{(t+1)} = x_j^{(t)}$. For $\mathbf{x} \in \mathbb{R}^n$, we denote the sum of entries of the matrix DAD^{-1} for $D = \mathtt{diag}(e^{x_1}, e^{x_2}, \ldots, e^{x_n})$ by $f(\mathbf{x}) = f_A(\mathbf{x}) = \sum_{ij} a_{ij} \cdot e^{x_i - x_j}$. For any $n \times n$ non-negative matrix $B = (b_{ij})$, we denote by G_B the weighted directed graph with node set $\{1, 2, \ldots, n\}$, arc set $\{(i, j) : b_{ij} > 0\}$, where an arc (i, j) has weight b_{ij}. We will assume henceforth that the undirected version of G_A is connected, otherwise we can handle each connected component separately. We quote a few useful lemmas. The references contain the proofs.

▶ **Lemma 2** (Theorem 1 in Kalantari et al. [8]). *The input matrix A is balanceable if and only if G_A is strongly connected. Moreover, DAD^{-1} is balanced in the L_1 norm if and only if $D = \mathtt{diag}(e^{x_1^*}, e^{x_2^*}, \ldots, e^{x_n^*})$, where $\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_n^*)$ minimizes $f(\mathbf{x})$ over $\mathbf{x} \in \mathbb{R}^n$.*

Notice that f is a convex function and the gradient $\nabla f(\mathbf{x})$ of f at $\mathbf{x}$ is given by

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \sum_{j=1}^{n} a_{ij} \cdot e^{x_i - x_j} - \sum_{j=1}^{n} a_{ji} \cdot e^{x_j - x_i},$$

the difference between the total weight of arcs leaving node i and the total weights of arcs going into node i in the graph of DAD^{-1} for $D = \mathtt{diag}(e^{x_1}, e^{x_2}, \ldots, e^{x_n})$. If DAD^{-1} is balanced then the arc weights $a_{ij} \cdot e^{x_i - x_j}$ form a valid circulation in the graph G_A, since the gradient has to be 0. Some properties of f are given in the following lemma.

▶ **Lemma 3** (Lemmas 2.1 and 2.2 in Ostrovsky et al. [11]). *If $\mathbf{x}'$ is derived from $\mathbf{x}$ by balancing index i of a matrix $B = (b_{ij})_{n \times n}$, then $f(\mathbf{x}) - f(\mathbf{x}') = (\sqrt{\|b_{.,i}\|_1} - \sqrt{\|b_{i,.}\|_1})^2$. Also, for all $\mathbf{x} \in \mathbb{R}^n$, $f(\mathbf{x}) - f(\mathbf{x}^*) \leq \frac{n}{2} \cdot \|\nabla f(\mathbf{x})\|_1$.*

We also need the following absolute bounds on the arc weights.

▶ **Lemma 4** (Lemma 3.2 in Ostrovsky et al. [11]). *Suppose that a matrix B is derived from a matrix A through a sequence of balancing operations. Then, for every arc (i, j) of G_B,*

$$\left(\frac{a_{\min}}{\sum_{ij} a_{ij}} \right)^n \cdot \sum_{ij} a_{ij} \leq b_{ij} \leq \sum_{ij} a_{ij}.$$

(Notice that the arcs of G_B are identical to the arcs of G_A.)

Finally, we prove the following global condition on indices being ϵ-balanced.

▶ **Lemma 5.** *Consider a matrix $B = DAD^{-1} = (b_{ij})_{n \times n}$, where $D = \mathtt{diag}(e^{x_1}, e^{x_2}, \ldots, e^{x_n})$, that was derived from A by a sequence of zero or more balancing operations. Let $\epsilon \in (0, 1/2]$, and put $\epsilon' = \frac{\epsilon^2}{64n^4}$. Suppose that $\|\nabla f_A(\vec{0})\|_1 \leq \epsilon' \cdot f_A(\vec{0})$. Then, for every $i \in [n]$ we have the following implication. If $\|b_{.,i}\|_1 + \|b_{i,.}\|_1 \geq \frac{1}{8n^3} \cdot f_A(\mathbf{x})$, then index i is ϵ-balanced in B.*

Proof. We will show the contrapositive claim that if a node is not ϵ-balanced then it must have low weight (both with respect to B). Let i be an index that is not ϵ-balanced in B. Without loss of generality we may assume that the in-weight is larger than the out-weight,

so $\|b_{.,i}\|_1/\|b_{i,.}\|_1 > 1+\epsilon$. Consider what would happen if we balance index i in B, yielding a vector $\mathbf{x}'$ that differs from $\mathbf{x}$ only in the i-th coordinate.

$$\begin{aligned} f_A(\mathbf{x}) - f_A(\mathbf{x}') &= \left(\sqrt{\|b_{.,i}\|_1} - \sqrt{\|b_{i,.}\|_1}\right)^2 \\ &> \|b_{.,i}\|_1 \cdot \left(1 - \sqrt{\frac{1}{1+\epsilon}}\right)^2 \\ &> \frac{\epsilon^2}{16} \cdot (\|b_{.,i}\|_1 + \|b_{i,.}\|_1)\,, \end{aligned} \tag{1}$$

where the equation follows from Lemma 3 and the last inequality uses the fact that $\epsilon \leq \frac{1}{2}$. On the other hand, we have

$$\begin{aligned} f_A(\mathbf{x}) - f_A(\mathbf{x}') &\leq f_A(\vec{0}) - f(\mathbf{x}^*) \\ &\leq \frac{n}{2} \cdot \|\nabla f_A(\vec{0})\|_1 \\ &\leq \frac{n}{2} \cdot \epsilon' \cdot f_A(\vec{0}) \\ &= \frac{\epsilon^2}{128n^3} \cdot f_A(\vec{0}). \end{aligned} \tag{2}$$

where the first inequality follows from the the fact that every balancing step decreases f_A, the second inequality follows from Lemma 3, the third inequality follows from the assumption on $f_A(\vec{0})$, and the last equation follows from the choice of ϵ'. Combining the bounds on $f_A(\mathbf{x}) - f_A(\mathbf{x}')$ in Equations (1) and (2) gives

$$\|b_{.,i}\|_1 + \|b_{i,.}\|_1 < \frac{1}{8n^3} \cdot f_A(\vec{0}),$$

and this completes the proof. ◀

3 Strict Balancing

In this section we present a variant of Osborne's iteration and prove that it converges in polynomial time to a strictly ϵ-balanced matrix. The algorithm, a procedure named StrictBalance, is defined in pseudocode labeled Algorithm 1 on page 7. Lemma 5 above motivates the main idea of contracting heavy nodes in step 14 of StrictBalance.

Our main theorem is

▶ **Theorem 6.** *StrictBalance(A, ϵ) returns a strictly ϵ-balanced matrix $B = DAD^{-1}$ after at most $O\left(\epsilon^{-2}n^9 \log(wn/\epsilon) \log w/\log n\right)$ balancing steps, using $O\left(\epsilon^{-2}n^{10}\log(wn/\epsilon)\log w\right)$ arithmetic operations over $O(n\log(w/\epsilon))$-bit numbers.*

The proof of Theorem 6 uses a few arguments, given in the following lemmas. A *phase* of StrictBalance is one iteration of the outer while loop. Notice that in the beginning of this loop the variable s indexes the phase number (i.e., $s-1$ phases were completed thus far). Also in the beginning of the inner while loop the variable t indexes the total iteration number from all phases (i.e., $t-1$ balancing operations from all phases were completed thus far).

We identify outer loop iteration s with an interval $[t_s, t_{s+1}) = \{t_s, t_s+1, \ldots, t_{s+1}-1\}$ of the inner loop iterations executed during phase s. We denote by $\mathcal{B}_{s,t}$ the value of $\mathcal{B}_s$ in the beginning of the inner while loop iteration number t (dubbed time t). If $t \in [t_j, t_{j+1})$, then $\mathcal{B}_{s,t}$ is defined only for $s \leq j$. We also use $G^{(\mathcal{B}_{s,t})}$ to denote the graph that is obtained by contracting the nodes of set $\mathcal{B}_{s,t}$ in graph G_A. Also $f^{(\mathcal{B}_{s,t})}$ is the function corresponding to

Algorithm 1 StrictBalance(A, ϵ)

Input: Matrix $A \in \mathbb{R}^{n\times n}, \epsilon$
Output: A strictly ϵ-balanced matrix

1: $\mathcal{B}_1 = \varnothing$, $\tau_1 = 0$, $s = 1$, $\epsilon' = \epsilon^2/64n^4$, $\mathbf{x^{(1)}} = (0, \ldots, 0)$, $t = 1$
2: **while** $\mathcal{B}_s \neq [n]$ and there is $i \in [n]$ that is not ϵ-balanced **do**
3: Define $f^{(\mathcal{B}_s)} : \mathbb{R}^n \to \mathbb{R}$, $f^{(\mathcal{B}_s)}(\mathbf{x}) = \sum_{i,j:i\notin\mathcal{B}_s \text{ or } j\notin\mathcal{B}_s} a_{ij}e^{x_i-x_j}$
4: **while** $\frac{\|\nabla f^{(\mathcal{B}_s)}(\mathbf{x^{(t)}})\|_1}{f^{(\mathcal{B}_s)}(\mathbf{x^{(t)}})} > \epsilon'$ **do**
5: Pick $i = \arg\max_{i\notin\mathcal{B}_s} \left\{ \left(\sqrt{\|a^{(t)}_{.,i}\|_1} - \sqrt{\|a^{(t)}_{i,.}\|_1} \right)^2 \right\}$
6: Balance ith node: $\mathbf{x^{(t+1)}} = \mathbf{x^{(t)}} + \alpha_t \mathbf{e}_i$, where $\alpha_t = \ln \sqrt{\|a^{(t)}_{.,i}\|_1 / \|a^{(t)}_{i,.}\|_1}$
7: $t \leftarrow t + 1$
8: **if** $s > 1$ and $\|a^{(t)}_{.,i}\|_1 + \|a^{(t)}_{i,.}\|_1 < \tau_s$ for some $i \in \mathcal{B}_s \setminus \mathcal{B}_{s-1}$ **then**
9: $\mathcal{B}_s = \mathcal{B}_s \setminus \{i \notin \mathcal{B}_{s-1} : \|a^{(t)}_{.,i}\|_1 + \|a^{(t)}_{i,.}\|_1 < \tau_s\}$
10: Redefine $f^{(\mathcal{B}_s)} : \mathbb{R}^n \to \mathbb{R}$, $f^{(\mathcal{B}_s)}(\mathbf{x}) = \sum_{i,j:i\notin\mathcal{B}_s \text{ or } j\notin\mathcal{B}_s} a_{ij}e^{x_i-x_j}$
11: **end if**
12: **end while**
13: $\tau_{s+1} = \frac{1}{4n^3} f^{(\mathcal{B}_s)}(\mathbf{x^{(t)}})$
14: $\mathcal{B}_{s+1} = \mathcal{B}_s \cup \left\{ i : \|a^{(t)}_{.,i}\|_1 + \|a^{(t)}_{i,.}\|_1 \geq \tau_{s+1} \right\}$
15: $s \leftarrow s + 1$
16: **end while**
17: **return** the resulting matrix

graph $G^{(\mathcal{B}_{s,t})}$ and $f^{(\mathcal{B}_{s,t})}(\mathbf{x^{(t)}})$ denotes the sum of weights of arcs of graph $G^{(\mathcal{B}_{s,t})}$ at time t. If set B_s is unchanged during an interval and there is no confusion, we may use $G^{(\mathcal{B}_s)}$ instead of $G^{(\mathcal{B}_{s,t})}$. Particularly we use $f^{(\mathcal{B}_s)}(\mathbf{x^{(t)}})$ instead of $f^{(\mathcal{B}_{s,t})}(\mathbf{x^{(t)}})$. We refer to the quantity $\|a^{(t)}_{.,i}\|_1 + \|a^{(t)}_{i,.}\|_1$ as the *weight* of node i at time t.

▶ **Lemma 7.** *For every phase $s \geq 1$, for every $t \geq t_{s+1}$, $\mathcal{B}_{s,t} = \mathcal{B}_{s,t_{s+1}}$.*

Proof. The claim follows easily from the fact that any iteration $t \geq t_{s+1}$ belongs to a phase $s' > s$, so $\mathcal{B}_{s,t_{s+1}} \cap (\mathcal{B}_{s',t} \setminus \mathcal{B}_{s'-1,t}) = \emptyset$, and by line 8 and 9 of StrictBalance none of the nodes in $\mathcal{B}_{s,t_{s+1}}$ will be removed. ◀

▶ **Lemma 8.** *For all $s > 1$, for all $t \in [t_s, t_{s+1})$, $f^{(\mathcal{B}_{s,t})}(\mathbf{x^{(t)}}) \leq (n - |\mathcal{B}_{s,t}|) \cdot \tau_s$.*

Proof. Let $t_s = t_{s,1} < t_{s,2} < t_{s,3} < \cdots < t_{s,\ell_s}$ denote the time steps before which $\mathcal{B}_s$ changes during phase s. For simplicity, we abuse notation and use $\mathcal{B}_{s,j}$ instead of $\mathcal{B}_{s,t_{s,j}}$. Clearly $\mathcal{B}_{s,1} \supseteq \mathcal{B}_{s,2} \ldots \supseteq \mathcal{B}_{s,\ell_s}$, because we only remove nodes from $\mathcal{B}_s$ once it is set. Fix $s > 1$. We prove this lemma by induction on $r \in \{1, 2, \ldots, \ell_s\}$. For the basis, let $r = 1$. Clearly, by

the way the algorithm sets $\mathcal{B}_s$ before time $t_{s,1}$, all nodes with weight $\geq \tau_s$ are in $\mathcal{B}_s$, and therefore every node $i \notin \mathcal{B}_s$ has weight at most τ_s, so the lemma follows. Now, assume that the lemma is true for every $t \leq t_{s,r}$, we show that the lemma holds for every $t \leq t_{s,r+1}$. If $t \in [t_{s,r}, t_{s,r+1})$, then $\mathcal{B}_{s,t} = \mathcal{B}_{s,t_{s,r}}$, and we have:

$$f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t})}) \leq f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t_{s,r}})}) \leq \left(n - \left|\mathcal{B}_{s,t_{s,r}}\right|\right) \cdot \tau_s = (n - |\mathcal{B}_{s,t}|) \cdot \tau_s.$$

The first inequality holds because balancing operations from time $t_{s,r}$ to time t only reduce the value of $f^{(\mathcal{B}_s)}$, and the second inequality holds by the induction hypothesis.

Just before iteration $t = t_{s,r+1}$, the set $\mathcal{B}_s$ changes, and one or more nodes are removed from it. However, every removed node has weight at most τ_s, and its removal does not change the weights of the other nodes in $[n] \setminus \mathcal{B}_s$. Therefore, if k nodes are removed from $\mathcal{B}_s$,

$$f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t_{s,r+1}})}) \leq \left(n - \left|\mathcal{B}_{s,t_{s,r}}\right|\right) \cdot \tau_s + k \cdot \tau_s = \left(n - \left|\mathcal{B}_{s,t_{s,r+1}}\right|\right) \cdot \tau_s.$$

This completes the proof. ◀

▶ **Corollary 9.** *For all $s > 1$, $f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t_{s+1}})}) \leq \frac{1}{4n^2} \cdot f^{(\mathcal{B}_{s-1})}(\mathbf{x}^{(\mathbf{t_s})})$. If $s > 2$, then $\tau_s \leq \frac{\tau_{s-1}}{4n^2}$.*

Proof. Notice that

$$f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t_{s+1}})}) \leq n \cdot \tau_s = \frac{1}{4n^2} \cdot f^{(\mathcal{B}_{s-1})}(\mathbf{x}^{(\mathbf{t_s})}),$$

where the inequality follows from Lemma 8, and the equation follows from line 13 of StrictBalance. This proves the first assertion. As for the second assertion, notice that if $s > 2$ then $s - 1 > 1$, so using line 13 of StrictBalance and Lemma 8 again,

$$\tau_s = \frac{1}{4n^3} \cdot f^{(\mathcal{B}_{s-1})}(\mathbf{x}^{(\mathbf{t_s})}) \leq \frac{1}{4n^3} \cdot n\tau_{s-1} = \frac{1}{4n^2} \cdot \tau_{s-1},$$

as stipulated. ◀

▶ **Lemma 10.** *For every phase $s > 1$, for every $t \geq t_s$, all the nodes in $\mathcal{B}_{s,t}$ have weight $\geq \tau_s/2$ and are ϵ-balanced at time t.*

Proof. Fix $s > 1$ and let $i \in \mathcal{B}_{s,t}$. Without loss of generality $i \notin \mathcal{B}_{s-1,t}$, otherwise we can replace s with $s - 1$. (Recall that $\mathcal{B}_1 = \emptyset$ at all times.) Also note that it must be the case that $i \in \mathcal{B}_{s,t_s}$, because $\mathcal{B}_s$ does not accumulate additional nodes after being created. If $t \in [t_s, t_{s+1}]$, then lines 13-14 and 8-9 of StrictBalance guarantee that if $i \in \mathcal{B}_{s,t} \setminus \mathcal{B}_{s-1,t}$, then its weight at time t is at least τ_s.

Otherwise, consider $t > t_{s+1}$ and let $s' > s$ be the phase containing t. Consider a phase $j > s$. By Lemma 8 the total weight of $f^{(\mathcal{B}_j)}$ during phase j is at most $n\tau_j$, and $f^{(\mathcal{B}_j)}$ never drops below 0. So, the total weight that a node $i \in \mathcal{B}_j$ can lose (which is at most the total weight that $f^{(\mathcal{B}_j)}$ can lose) is at most $n\tau_j$. By Corollary 9, for every $j > s$, $\tau_{j+1} \leq \frac{\tau_j}{4n^2}$. Now, suppose that t is an iteration in phase $s' > s$. Then, the weight of i at time t is at least

$$\tau_s - \sum_{j=s+1}^{s'} n\tau_j \geq \tau_s \cdot \left(1 - n \cdot \sum_{k=1}^{s'-s} (2n)^{-2k}\right) \geq \frac{\tau_s}{2}.$$

Thus we have established that at any time $t \geq t_s$, if $i \in \mathcal{B}_{s,t}$ then its weight is at least $\frac{\tau_s}{2} = \frac{1}{8n^3} f^{(\mathcal{B}_{s-1})}(\mathbf{x}^{(\mathbf{t_s})})$. By line 4 of StrictBalance, $\|\nabla f^{(\mathcal{B}_{s-1,t_s})}(\mathbf{x}^{(\mathbf{t_s})})\|_1 \leq \epsilon' \cdot f^{(\mathcal{B}_{s-1,t_s})}(\mathbf{x}^{(\mathbf{t_s})})$. By Lemma 7, $\mathcal{B}_{s-1}$ does not change in the interval $[t_s, t]$. Therefore, we conclude from Lemma 5 that i is ϵ-balanced at time t. ◀

▶ **Lemma 11.** *Suppose that $t < t'$ satisfies $[t, t') \subseteq [t_s, t_{s+1})$, and furthermore, during the iterations in the interval $[t, t')$ the set $\mathcal{B}_s$ does not change (it could change after balancing step $t' - 1$). Then, the length of the interval*

$$t' - t = O\left(\epsilon^{-2} n^7 \log(wn/\epsilon)\right).$$

Proof. Rename the nodes so that $\mathcal{B}_{s,t} = \mathcal{B}_{s,t'-1} = \{p, p+1, \ldots, n\}$. The assumption that $\mathcal{B}_s$ does not change during the interval $[t, t')$ means that the weights of all the nodes $p, p+1, \ldots, n$ remain at least τ_s for the duration of this interval. During the interval $[t, t')$, the graph $G^{(\mathcal{B}_s)}$ (which remains fixed) is obtained by contracting the nodes $p, p+1, \ldots, n$ in G_A. So $G^{(\mathcal{B}_s)}$ has p nodes $1, 2, \ldots, p-1, p$, where the last node p is the contracted node. In each iteration in the interval $[t, t')$, one of the nodes $1, 2, \ldots, p-1$ is balanced. Consider some time step $t'' \in [t, t')$, and let I_i and O_i, respectively, denote the current sums of weights of the arcs of $G^{(\mathcal{B}_s)}$ into and out of node i, respectively. Let $j \in [p-1]$ be the node that maximizes $\frac{(I_j - O_j)^2}{I_j + O_j}$. We have

$$\begin{aligned}
f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}'')}) - f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}''+\mathbf{1})}) &= \max_{i \in [p-1]} \left(\sqrt{I_i} - \sqrt{O_i}\right)^2 \geq \left(\sqrt{I_j} - \sqrt{O_j}\right)^2 \\
&\geq \frac{(I_j - O_j)^2}{2(I_j + O_j)} \geq \frac{\sum_{i=1}^{p-1} (I_i - O_i)^2}{2 \sum_{i=1}^{p-1} (I_i + O_i)} \\
&\geq \frac{\left(\sum_{i=1}^{p-1} |I_i - O_i|\right)^2}{2n \sum_{i=1}^{p} (I_i + O_i)} \geq \frac{\left(\sum_{i=1}^{p} |I_i - O_i|\right)^2}{8n \sum_{i=1}^{p} (I_i + O_i)} \\
&= \frac{1}{16n} \cdot \frac{\|\nabla f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}'')})\|_1^2}{f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}'')})}. \qquad (3)
\end{aligned}$$

The first equation follows from the choice of i in line 5 StrictBalance, and Lemma 3. The third inequality follows from an averaging argument and the choice of j. The fourth inequality uses Cauchy-Schwarz. The last inequality holds because $\sum_{i=1}^{p} (I_i - O_i) = 0$, so $|I_p - O_p| = \left|\sum_{i=1}^{p-1} (I_i - O_i)\right| \leq \sum_{i=1}^{p-1} |I_i - O_i|$, and therefore $\sum_{i=1}^{p} |I_i - O_i| \leq 2 \sum_{i=1}^{p-1} |I_i - O_i|$.

Since the interval $[t, t')$ is contained in phase s, the stopping condition for the phase does not hold, so

$$\frac{\|\nabla f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}'')})\|_1}{f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}'')})} > \epsilon' = \frac{\epsilon^2}{64n^4}.$$

Therefore,

$$\begin{aligned}
f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}'')}) - f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}''+\mathbf{1})}) &\geq \frac{1}{16n} \cdot \frac{\|\nabla f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}'')}\|_1^2}{f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}'')})} \\
&> \frac{\epsilon'}{16n} \cdot \|\nabla f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}'')})\|_1 \\
&\geq \frac{\epsilon'}{8n^2} \cdot (f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}'')}) - f^{(\mathcal{B}_s)}(\mathbf{x}^*)),
\end{aligned}$$

where the last inequality follows from Lemma 3. Rearranging the terms gives

$$f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}''+\mathbf{1})}) - f^{(\mathcal{B}_s)}(\mathbf{x}^*) \leq \left(1 - \frac{\epsilon'}{8n^2}\right) \cdot (f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}'')}) - f^{(\mathcal{B}_s)}(\mathbf{x}^*)).$$

Iterating for T step yields

$$f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t}+\mathbf{T})}) - f^{(\mathcal{B}_s)}(\mathbf{x}^*) \leq \left(1 - \frac{\epsilon'}{8n^2}\right)^T \cdot (f^{(\mathcal{B}_s)}(\mathbf{x}^{(\mathbf{t})}) - f^{(\mathcal{B}_s)}(\mathbf{x}^*)).$$

Now, by Lemma 4, we have that $f^{(\mathcal{B}_s)}(\mathbf{x^{(t)}}) - f^{(\mathcal{B}_s)}(\mathbf{x}^*) \le f^{(\mathcal{B}_s)}(\mathbf{x^{(t)}}) \le \sum_{i,j=1}^{n} a_{ij}$, and for all t'', $f^{(\mathcal{B}_s)}(\mathbf{x^{(t'')}}) \ge \frac{1}{w^n}\sum_{i,j=1}^{n} a_{ij}$. Therefore, if $t' - t \ge \frac{8n^2}{\epsilon'} \cdot \ln\left(16nw^n/(\epsilon')^2\right) + 1$, then

$$f^{(\mathcal{B}_s)}(\mathbf{x^{(t'-1)}}) - f^{(\mathcal{B}_s)}(\mathbf{x}^*) \le \left(\frac{\epsilon'}{4\sqrt{n}}\right)^2 \cdot \frac{1}{w^n} \cdot \sum_{i,j=1}^{n} a_{ij} \le \left(\frac{\epsilon'}{4\sqrt{n}}\right)^2 \cdot f^{(\mathcal{B}_s)}(\mathbf{x^{(t'-1)}}).$$

Therefore,

$$\begin{aligned} \frac{1}{16n} \cdot \frac{\|\nabla f^{(\mathcal{B}_s)}(\mathbf{x^{(t'-1)}})\|_1^2}{(f^{(\mathcal{B}_s)}(\mathbf{x^{(t'-1)}}))^2} &\le \frac{f^{(\mathcal{B}_s)}(\mathbf{x^{(t'-1)}}) - f^{(\mathcal{B}_s)}(\mathbf{x^{(t')}})}{f^{(\mathcal{B})}(\mathbf{x^{(t'-1)}})} \\ &\le \frac{f^{(\mathcal{B}_s)}(\mathbf{x^{(t'-1)}}) - f^{(\mathcal{B}_s)}(\mathbf{x}^*)}{f^{(\mathcal{B})}(\mathbf{x^{(t'-1)}})} \le \left(\frac{\epsilon'}{4\sqrt{n}}\right)^2, \end{aligned}$$

where the first inequality follows from (3), and the second inequality holds because $f^{(\mathcal{B}_s)}(\mathbf{x}^*) \le f^{(\mathcal{B}_s)}(\mathbf{x^{(t'-1)}})$. We get that $\frac{\|\nabla f^{(\mathcal{B})}(\mathbf{x^{(t'-1)}})\|_1}{f^{(\mathcal{B})}(\mathbf{x^{(t'-1)}})} \le \epsilon'$, in contradiction to our assumption that the phase does not end before the start of iteration t'. ◀

▶ **Corollary 12.** *In any phase, the number of balancing steps is at most* $O\left(\epsilon^{-2}n^8\log(wn/\epsilon)\right)$.

Proof. In the beginning of phase s the set $\mathcal{B}_s$ contains at most $n-1$ nodes. Partition the phase into intervals $[t, t')$ where $\mathcal{B}_s$ does not change during an interval, but does change between intervals. By Lemma 11, each interval consists of at most $O\left(\epsilon^{-2}n^7\log(wn/\epsilon)\right)$ balancing steps. Since nodes that are removed from $\mathcal{B}_s$ between intervals are never returned to $\mathcal{B}_s$, the number of such intervals is at most $n-1$. Hence, the total number of balancing steps in the phase is at most $O\left(\epsilon^{-2}n^8\log(wn/\epsilon)\right)$. ◀

▶ **Lemma 13.** *The total number of phases of the algorithm is* $O(n\log w/\log n)$.

Proof. Let $s > 2$ be a phase of the algorithm and $t \in [t_s, t_{s+1})$. By Lemma 4, $f^{(\mathcal{B}_{s,t})}(\mathbf{x^{(t)}}) \ge \frac{1}{w^n}\cdot\sum_{ij} a_{ij}$. On the other hand, by Lemma 8 and Corollary 9, $\tau_s \le \frac{1}{(4n^2)^{s-2}}\cdot\tau_2 \le \frac{1}{(4n^2)^{s-2}}\cdot\sum_{ij} a_{ij}$, and $f^{(\mathcal{B}_{s,t})}(\mathbf{x^{(t)}}) \le n\tau_s$. Combining these gives $\frac{1}{w^n}\cdot\sum_{ij} a_{ij} \le n\tau_s \le \frac{n}{(4n^2)^{s-2}}\cdot\sum_{ij} a_{ij}$ which implies that $s \le \frac{\log(nw^n)}{\log(4n^2)} + 2$. ◀

Proof of Theorem 6. By Lemma 13, for some $s = O(n\log w/\log n)$, StrictBalance terminates, so $\mathcal{B}_{s,t_s} = [n]$. By Corollary 12, the number of balancing steps in a phase is at most $O\left(\epsilon^{-2}n^8\log(wn/\epsilon)\right)$. Therefore, the total number of balancing steps is at most $O\left(\epsilon^{-2}n^9\log(wn/\epsilon)\log w/\log n\right)$. These steps require at most $O\left(\epsilon^{-2}n^{10}\log(wn/\epsilon)\log w\right)$ arithmetic operations. When the algorithm terminates at time t_s, all the nodes are in $\mathcal{B}_{s,t_s}$, and by Lemma 10 they are all ϵ-balanced, so the matrix is strictly ϵ-balanced.

Thus, assuming exact arithmetics with infinite precision we have shown that the algorithm converges to a strictly ϵ-balanced matrix in the claimed number of arithmetic operations. To show that the algorithm still works if all numbers are represented with only $O(n\log(w/\epsilon))$ bits, we apply an analysis similar to the one in section 3 of [11]. ◀

References

1 EISPACK implementation. URL: `http://www.netlib.org/eispack/balanc.f`.

2 Zeyuan Allen-Zhu, Yuanzhi Li, Rafael Oliveira, and Avi Wigderson. Much faster algorithms for matrix scaling. In *Proc. of the 58th Ann. IEEE Symp. on Foundations of Computer Science*, 2017.

3 T.-Y. Chen. Balancing sparse matrices for computing eigenvalues. Master's thesis, UC Berkeley, May 1998.

4 Michael B. Cohen, Aleksander Madry, Dimitris Tsipras, and Adrian Vladu. Matrix scaling and balancing via box constrained newton's method and interior point methods. In *Proc. of the 58th Ann. IEEE Symp. on Foundations of Computer Science*, 2017.

5 B. C. Eaves, A. J. Hoffman, U. G. Rothblum, and H. Schneider. Line-sum-symmetric scalings of square nonnegative matrices. In *Mathematical Programming Essays in Honor of George B. Dantzig Part II*, pages 124–141. Springer, 1985.

6 J. Grad. Matrix balancing. *The Computer Journal*, 14(3):280–284, 1971.

7 D. J. Hartfiel. Concerning diagonal similarity of irreducible matrices. In *Proceedings of the American Mathematical Society*, pages 419–425, 1971.

8 B. Kalantari, L. Khachiyan, and A. Shokoufandeh. On the complexity of matrix balancing. *SIAM Journal on Matrix Analysis and Applications*, 118(2):450–463, 1997.

9 D. Kressner. *Numerical methods for general and structured eigenvalue problems*. Princeton University Press, 2005.

10 E. E. Osborne. On pre-conditioning of matrices. *Journal of the ACM (JACM)*, 7(4):338–345, 1960.

11 Rafail Ostrovsky, Yuval Rabani, and Arman Yousefi. Matrix balancing in lp norms: Bounding the convergence rate of osborne's iteration. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 154–169, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics. URL: `http://dl.acm.org/citation.cfm?id=3039686.3039697`.

12 B. N. Parlett and C. Reinsch. Balancing a matrix for calculation of eigenvalues and eigenvectors. *Numerische Mathematik*, 13(4):293–304, 1969.

13 W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing, 3rd Edition*. Cambridge University Press, 2007.

14 H. Schneider and M. H. Schneider. Max-balancing weighted directed graphs and matrix scaling. *Mathematics of Operations Research*, 16(1):208–222, 1991. `doi:10.1287/moor.16.1.208`.

15 L. J. Schulman and A. Sinclair. Analysis of a classical matrix preconditioning algorithm. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 831–840, 2015.

16 L. N. Trefethen and M. Embree. *Spectra and pseudospectra: The behavior of nonnormal matrices and operators*. Springer, 2005.

17 N. E. Young, R. E. Tarjan, and J. B. Orlin. Faster parametric shortest path and minimum-balance algorithms. *Networks*, 21(2):205–221, 1991. `doi:10.1002/net.3230210206`.

Parameterized Algorithms for Zero Extension and Metric Labelling Problems

Felix Reidl
Royal Holloway, University of London, TW20 0EX, UK
Felix.Reidl@rhul.ac.uk

Magnus Wahlström
Royal Holloway, University of London, TW20 0EX, UK
Magnus.Wahlstrom@rhul.ac.uk

Abstract

We consider the problems Zero Extension and Metric Labelling under the paradigm of parameterized complexity. These are natural, well-studied problems with important applications, but have previously not received much attention from this area.

Depending on the chosen cost function μ, we find that different algorithmic approaches can be applied to design FPT-algorithms: for arbitrary μ we parameterize by the number of edges that cross the cut (not the cost) and show how to solve Zero Extension in time $O(|D|^{O(k^2)}n^4 \log n)$ using randomized contractions. We improve this running time with respect to both parameter and input size to $O(|D|^{O(k)}m)$ in the case where μ is a metric. We further show that the problem admits a polynomial *sparsifier*, that is, a kernel of size $O(k^{|D|+1})$ that is *independent of the metric* μ.

With the stronger condition that μ is described by the distances of leaves in a tree, we parameterize by a *gap parameter* $(q-p)$ between the cost of a true solution q and a 'discrete relaxation' p and achieve a running time of $O(|D|^{q-p}|T|m + |T|\phi(n,m))$ where T is the size of the tree over which μ is defined and $\phi(n,m)$ is the running time of a max-flow computation. We achieve a similar result for the more general Metric Labelling, while also allowing μ to be the distance metric between an arbitrary subset of nodes in a tree using tools from the theory of VCSPs. We expect the methods used in the latter result to have further applications.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases FPT, VCSP, cut problem, gap parameter

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.94

Related Version A full version of the paper is available at [27], `http://arxiv.org/abs/1802.06026`.

Funding Supported by EPSRC grant EP/P007228/1

1 Introduction

The task of extending a partial labelling of data points to a full data set while minimizing an error function is a natural step for many scientific and engineering tasks. For the particular case of data imposed with a binary relationship, we find the problems Zero Extension and Metric Labelling to be well-suited for optimization in image processing [1], social network classification [24], or sentiment analysis [25]. The problems are as follow. For Zero Extension, we are given a graph G and a partial labelling $\tau\colon S \to D$, for terminals $S \subseteq V(G)$, and a cost function $\mu\colon D \times D \to \mathbb{R}^+$. Our task is to compute a labelling $\lambda\colon V(G) \to D$ which agrees with τ on S, subject to the following cost: for each edge $uv \in G$ we pay $\mu(\lambda(u), \lambda(v))$. In Metric Labelling, we are given G, μ as above, and a *labelling cost* $\sigma\colon V(G) \times D \to \mathbb{R}^+$.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 94; pp. 94:1–94:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Again we are asked to compute a labelling λ and in addition to the above edge-costs we now also pay $\sigma(v, \lambda(v))$. This model allows us to emulate terminals by making the cost $\sigma(v, \lambda(v))$ prohibitive for all but the required label $\lambda(v)$. Both problems are generalizations of Multiway Cut (we simply let μ be identically one for all distinct pairs), which has garnered considerable attention from the FPT community and formed a crystallization nucleus for the very fruitful research of cut-based problems (see e.g. [20, 17, 3, 11, 22]).
We apply most of these tools in the following, but we wish to highlight the use of tools and relaxations from *Valued CSPs* (VCSPs) for designing FPT algorithms under gap parameters. VCSPs are a general framework for expressing optimisation problems, via the specification of a set Γ of *cost functions* (aka *constraint language*). Many important problems correspond to VCSP for a specific language, including every choice of a specific metric for the problems above. Thapper and Živný [30] characterized the languages Γ for which the resulting VCSP is tractable.

The use of a tractable VCSP as a *discrete relaxation* of an NP-hard optimisation problem has led to powerful FPT algorithms [11] (see also related improvements [12, 32]). In this paper, we advance this research in two ways. First, previous approaches have required the relaxation to have a *persistence* property, which allows an optimum to be found by sequentially fixing variables. Here, we relax this condition to a weaker *domain consistency* property. Second, we use a folklore result from VCSP research to restrict the behaviour of an instance's optimal solutions in order to facilitate the proof that the domain consistency property holds for the relevant VCSPs. See Section 5 for details.

Related work. Zero Extension and Metric Labelling have been researched primarily from the perspective of efficient and approximation algorithms (see [19] for an overview and hardness results). Kleinberg and Tardos [29] introduced Metric Labelling and provided a $O(\log |S| \log \log |S|)$ approximation. A result by Fakcharoenphol *et al.* regarding embedding general metrics into tree metrics [7] improves the ratio of this algorithm to $O(\log |S|)$ and a lower bound of $O((\log |S|)^{1/2-\varepsilon})$ was proved by Chuzhoy and Naor [4]. Karzanov [14] introduced Zero Extension with the specific case of μ being a graph metric, that is, equal to the distance metric of some graph H. His central question—for which graphs H the problem is tractable—was recently fully answered by Hirai [9]. Picard and Ratliff earlier showed that an equivalent problem is tractable on trees [26]. Fakcharoenphol *et al.* showed that the problem can be approximated to within a factor of $O(\log |S| / \log \log |S|)$ [6]. Karloff *et al.* used the approach by Chuzhoy and Naor to show that no factor of $O((\log |S|)^{1/4-\varepsilon})$ for any $\varepsilon > 0$ is possible unless NP $\subseteq$ QP [13]. More recently, Hirai and Pap [8, 10] studied the problem from a more structural angle and we make use of their duality result.

Our results. We study both problems from the perspective of parameterized complexity. As the choice of metric has a strong effect on the problem complexity, we give a range of results, from the more generally applicable to the algorithmically stronger, both in terms of running time and parameterization. When μ is a general cost function or a metric, we will parametrize not by the *cost* of a solution but by the number of *crossing edges*, i.e., bichromatic edges under a labelling λ. This lets us consider μ with zero-cost pairs. For general cost functions, we employ the technique of randomized contractions [3] and prove:

▶ **Theorem 1** ($\star$[1]). Zero Extension *can be solved in time* $O(|D|^{O(k^2)} n^4 \log n)$ *where* k *is a given upper bound on the number of crossing edges in the solution.*

[1] Results marked by $\star$ are found in the full version of the paper [27]

When μ is a metric, we are able to give a linear-time FPT algorithm, while also improving the dependency on the parameter, using important separators [20]:

▶ **Theorem 2.** Zero Extension *with metric cost functions can be solved in time* $O(|D|^{O(k)} \cdot m)$ *where* k *is a given upper bound on the number of crossing edges in the solution.*

For the general metric setting, we also have our most surprising result, demonstrating that Zero Extension admits a *sparsifier*; that is, we prove that it admits a polynomial kernel *independent of the metric* μ. This result crucially builds on the technique of *representative sets* [17, 21, 18]. The exact formulation of the result is somewhat technical and we defer it to Section 4.2, but roughly, we obtain a kernel of size $O(k^{|S|+1})$, independent of μ, where k is again the number of crossing edges. This result is a direct, seemingly far-reaching generalization of the polynomial kernel for s-Multiway Cut [17].

Next, we consider the case when $\mu\colon D \times D \to \mathbb{Z}^+$ is induced by the distance in a tree T with $D \subset V(T)$. Here, relaxing the problem to allow all labels $V(T)$ as vertex values defines a tractable discrete relaxation, in the sense discussed above. In particular, we can compute a *relaxed solution cost* p in polynomial time which lower-bounds the optimal integral solution q. Using techniques from VCSP, we design a gap-parameter algorithm:

▶ **Theorem 3.** *Let* $I = (G, \tau, \mu, q)$ *be an instance of* Zero Extension *where* μ *is an induced tree metric on a set of labels* D *in a tree* T*, and let* $\hat{I} = (G, \tau, \hat{\mu}, q)$ *be the relaxed instance. Let* $p = \mathbf{cost}(\hat{I})$*. Then we can solve* I *in time* $O(|D|^{q-p}|T||D|nm)$*.*

For the further restriction when μ corresponds to the distances of the leaves D of a tree T, we obtain an algorithm with a slightly better polynomial dependence. Moreover, it uses only elementary operations like computing cuts and flows:

▶ **Theorem 4.** *Let* $I = (G, \tau, \mu, q)$ *be an instance of* Zero Extension *where* μ *is a leaf metric on a set of labels* D *in a tree* T*, and let* $\hat{I} = (G, \tau, \hat{\mu}, q)$ *be the relaxed instance. Let* $p = \mathbf{cost}(\hat{I})$*. Then we can solve* I *in time* $O(|D|^{q-p}|T|m + |T|\phi(n, m))$*, where* ϕ *is the time needed to run a max-flow algorithm.*

Finally, we apply the VCSP toolkit to Metric Labelling and obtain a similar gap algorithm (see Section 5 for undefined terms).

▶ **Theorem 5.** *Let* $I = (G, \sigma, \mu, q)$ *be an instance of* Metric Labelling *where* μ *is an induced tree metric for a tree* T *and a set of nodes* $D \subseteq V(T)$*, and where every unary cost* $\sigma(v, \cdot)$ *admits an interpolation on* T*. Let* $\hat{I} = (G, \hat{\sigma}, \hat{\mu}, q)$ *be the relaxed instance, and let* $p = \mathbf{cost}(\hat{I})$*. Then the instance* I *can be solved in time* $O^*(|D|^{q-p})$*. In particular, this applies for any* σ *if* D *is the set of leaves of* T*.*

2 Preliminaries

For a graph $G = (V, E)$ we will use $n_G = |V|$ and $m_G = |E|$ to denote the number of vertices and edges, respectively. For two disjoint vertex sets $A, B \subseteq V$ we write $E(A, B)$ to denote the edges that have one endpoint in A and the other in B. We write d_G for the distance-metric induced by G, that is, $\mathrm{d}_G(u, v)$ is the length of a shortest path between vertices $u, v \in V(G)$. We denote by $N_G(v)$ and $N_G[v]$ the open and closed neighbourhood of a vertex. For a vertex set $S \subseteq V(G)$ we write $\delta_G(S)$ to denote the set of edges with exactly one endpoint in S. We omit the subscript G, if clear from the context, in all these notations.

Let T be a tree and $xy \in T$ an edge, then we use the notation T_x to denote the component of $T - xy$ that contains x. We call a sequence of nodes $x_1 x_2 \dots x_p$ in T a *monotone sequence*

if $x_1 \leqslant_P x_2 \leqslant_P \ldots \leqslant_P x_p$ where P is a path in T and $\leqslant_P$ is the linear order induced by P. Note that $x_i = x_{i+1}$ is explicitly allowed. For two nodes $x, y \in T$ we will denote the unique x-y-path in T by $T[x, y]$. For a vertex set S, an *S-path packing* is a collection of edge-disjoint paths $\mathcal{P}$ that connect pairs of vertices in S. We will also consider *half-integral* path packings where every edge is allowed to be used by at most two paths.

Let D be a set of labels. For a graph G we call a function $\tau \colon S \to D$ for $S \subseteq V(G)$ a *partial labelling* and a function $\lambda \colon V(G) \to D$ a *labelling*. The labelling λ is an *extension* of τ if λ and τ agree on S, that is, for every vertex $u \in S$ we have that $\lambda(u) = \tau(u)$. Given a graph G and a labelling λ we call an edge $uv \in E(G)$ *crossing* if $\lambda(u) \neq \lambda(v)$. A *$\tau$-path packing* is a collection $\mathcal{P}$ of edge-disjoint paths such that every path $P \in \mathcal{P}$ connects two vertices that receive distinct labels under τ (and both are labelled).

A *cost function* over D is a symmetric positive function $\mu \colon D \times D \to \mathbb{R}^+$. We call it *simple* if $\mu(x, x) = 0$. A cost function is a *metric* if it is simple and further obeys the triangle inequality; it is a *tree metric* if it corresponds to the distance metric of a tree. We derive an *induced tree metric* from a tree metric by restricting its domain to a subset D of the nodes of the underlying tree. A *leaf metric* is an induced tree metric where D is the set of leaves of the tree. Given a cost function μ, we define the cost of a labelling λ of a graph G as $\mathbf{cost}_\mu(\lambda, G) = \sum_{uv \in G} \mu(\lambda(u), \lambda(v))$.

3 Cost functions: Randomized Contractions

We apply the framework by Chitnis *et al.* [3] to show that the general case of Zero Extension is in FPT when parameterized by the number of crossing edges. Note that crossing edges could incur an arbitrary cost, including zero. The stronger parameterization of only counting the number of crossing edges at non-zero cost makes for an intractable problem: With zero-cost edges, we can express the problem H-Retraction for reflexive graphs H, which asks us to find a retraction of a graph G into a fixed graph H. This problem is already NP-complete for H being the reflexive 4-cycle [31] and thus Zero Extension is paraNP-complete for $k = 0$ when parameterized by the number of non-zero crossing edges or the total cost.

A (σ, κ)-*good* separation is a partition (L, R) of $V(G)$ such that $|L|, |R| > \sigma$, $|E(L, R)| \leqslant \kappa$, and both $G[L]$ and $G[R]$ are connected. There exists an algorithm that finds a (σ, κ)-good separation in time $O((\sigma+\kappa)^{O(\min(\sigma,\kappa))} n^3 \log n)$ (Lemma 2.2 in [3]) or concludes that the graph is (σ, κ)-*connected*, that is, no such separation exists. The following lemma is a slight reformulation of Lemma 1.1 in [3] which in turn is based on splitters as defined by Naor *et al.* [23]:

▶ **Lemma 6** (Edge splitter). *Given a set E of size m and integers $0 \leqslant a, b \leqslant m$ one can in time $O((a+b)^{O(\min\{a,b\})} m \log m)$ construct a set family $\mathcal{F}$ over E of size at most $O((a+b)^{O(\min\{a,b\})} \log m)$ with the following property: for any disjoint sets $A, B \subseteq E$ with $|A| \leqslant a$ and $|B| \leqslant b$ there exists a set $H \in \mathcal{F}$ with $A \subseteq H$ and $B \cap H = \emptyset$.*

We first prove that Zero Extension can be solved on such highly connected instances and then apply the 'recursive understanding' framework to handle graphs with good separations.

▶ **Lemma 7.** *Let G be (σ, k)-connected for some $\sigma > k$. Then we can find an optimal solution in time $O((|D| + 2\sigma k + k)^{O(k)}(n+m) \log n)$.*

Proof sketch. Let $\lambda \in \mathbf{opt}(I)$ and let E_λ be the crossing edges with endpoints $V(E_\lambda)$. Let $C_0, C_1, \ldots, C_\ell$ be the connected components of $G - E_\lambda$ with C_0 being the largest one. Since G is (σ, k)-connected, we know that $\ell \leqslant k$ and that all components $C_1, \ldots, C_\ell$ have size at most σ (*cf.* Lemma 3.6 in [3]). We will assume that $|C_0| > \sigma$, otherwise $|V(G)| \leqslant \sigma k$ vertices and we find E_λ by brute-force.

We proceed by colouring $E(G)$. Such a colouring is *successful* if 1) E_λ is red, 2) each C_i, $i \geqslant 1$, contains a blue spanning tree, and 3) each vertex $u \in C_0 \cap V(E_\lambda)$ is contained in a blue tree of size $\geqslant \sigma + 1$. It is easy to verify that we need to correctly colour a set $B \subseteq E(G)$, $|B| \leqslant (\sigma - 1)\ell + \sigma k \leqslant 2\sigma k$ edges blue while colouring a set $R \subseteq E(G)$, $|R| \leqslant k$, edges red. We construct an edge-splitter $\mathcal{F}$ with $a = 2\sigma k$ and $b = k$ according to Lemma 6 of size $O((2\sigma k + k)^{O(k)} \log m)$. By construction, at least one colouring in $\mathcal{F}$ will be successful.

Fix a successful colouring. Let G_B be the graph on the blue edges. Call a component of G_B *small* if it contains $\leqslant \sigma$ vertices and *big* otherwise. Our task is to recover $C_0, C_1, \ldots, C_\ell$. Every C_i, $i \geqslant 1$ is small in G_B and all components reachable from C_i via red edges must either be a solution component C_j, $j \geqslant 1$, *or* a big component in G_B. Thus, we can 'discover' the sets $C_1, \ldots, C_\ell$ by marking small components that contain a terminal and then successively mark small components with red edges into already marked components. Afterwards we identify the crossing edges E_λ and λ. The total running time to identify E_λ is $(2\sigma k + k)^{O(k)}(n + m) \log n$. Given E_λ, the final step is to find an optimal assignment. We simply try all possible assignments for non-terminal components in time $O(|D|^\ell k) = O(|D|^k k)$ and the claimed running time follows. ◀

With the well-connected cases handled, the theorem follows by a straightforward application of recursive understandings [3].

▶ **Theorem 1** ($\star$[2]). Zero Extension *can be solved in time* $O(|D|^{O(k^2)} n^4 \log n)$ *where* k *is a given upper bound on the number of crossing edges in the solution.*

4 General metrics: Pushing separators

We now consider the more restricted, but reasonable case that μ is a metric, observing the triangle inequality. We find that this allows a 'greedy' operation of *pushing* in a solution λ, which allows both the design of a faster algorithm (Section 4.1) and the computation of a *metric sparsifier* (Section 4.2). Throughout the section, let $I = (G = (V, E), \tau, \mu, q)$ be an instance of Zero Extension for an arbitrary metric μ, let S be the set of terminals of G, and let D be the set of labels. We assume that the following reductions have been performed on G: For every label ℓ used by τ there is a terminal t_ℓ, and every vertex v such that $\tau(v) = \ell$ has been identified with this terminal t_ℓ.

We first prove a useful lemma. Let $\lambda \colon V \to D$ be an extension of τ, and let $U = \lambda^{-1}(\ell)$ for some $\ell \in D$. By *pushing from* ℓ *in* λ we refer to the operation of relabelling vertices to grow the set U "as large as possible", without increasing the number of crossing edges. Formally, this refers to the following operation: Let C be the furthest min-cut between vertex sets U and $S - t_\ell$, respectively S if there is no terminal t_ℓ (cf. [20, Lemma 3]); let U' be the vertices reachable from U in $G - C$; and let λ' be the labelling where $\lambda'(v) = \ell$ for $v \in U'$ and $\lambda'(v) = \lambda(v)$ otherwise. Clearly, λ' is an extension of τ. We show that as long as μ is a metric (observing the triangle inequality), pushing does not increase the cost of the solution.

▶ **Lemma 8** (Pushing Lemma). *For any* τ*-extension* λ *and every label* $\ell \in D$*, pushing from* ℓ *in* λ *yields a* τ*-extension* λ' *with* $\mathbf{cost}_\mu(\lambda', G) \leqslant \mathbf{cost}_\mu(\lambda, G)$.

Proof sketch. Since C is a min-cut there is a set of paths that begin in $\delta(U)$, saturate C, and end in terminals $S - t_\ell$. By the triangle inequality, the cost incurred by λ on these paths is at least as large as that incurred by λ', and no other edge increases its cost in λ'. ◀

[2] Results marked by $\star$ are found in the full version of the paper [27]

An immediate consequence of the above lemma is the following reduction rule.

▶ **Corollary 9** (⋆). *We can reduce to the case where for every terminal $t_\ell \in S$, $\delta(t_\ell)$ is the unique $(t_\ell, S - t_\ell)$-min cut in G.*

4.1 An FPT algorithm

We now show that Zero Extension is FPT for a metric μ parameterized by $k + |D|$, where k is a bound on the number of crossing edges of an optimum λ. The algorithm uses Lemma 8 to guess a solution λ using the classical technique of *important separators*, pioneered by Marx [20] as an important technique for FPT algorithms solving cut problems. Our algorithm roughly follows the algorithm for Multiway Cut of Chen, Liu and Lu [2], with two complications. First, unlike in Multiway Cut, there may be crossing edges in λ that are not reachable by a terminal, which makes the branching more expensive; second, even after all crossing edges have been found, it still remains to find an optimal labelling λ.

Since these complications can be present or absent for different metrics μ, we describe the algorithm in stages, where the first stage identifies all crossing edges reachable from a terminal, the second stage identifies the remaining crossing edges, and the third stage finds an assignment λ. For specific metrics μ, it may then be possible to speed this up by skipping some steps. In summary, we show the following.

▶ **Theorem 2.** Zero Extension *with metric cost functions can be solved in time $O(|D|^{O(k)} \cdot m)$ where k is a given upper bound on the number of crossing edges in the solution.*

We begin by providing the running time for the first stage. This is analysed in terms of a lower bound p on the crossing number of any labelling λ. This may be defined as follows: First apply Corollary 9, then compute $p = \sum_{t \in S} |\delta(t)|/2$. It is known that p is a lower bound on the multiway cut number of (G, S) [28], hence also on the number of crossing edges of λ, making $k - p$ a valid gap parameter. (Note that p does not measure the *cost* of a crossing edge or path; such results are shown in Section 5.)

▶ **Lemma 10** (⋆). *Let p be the lower bound as above. In $O(4^{k-p}km)$ time and 4^{k-p} guesses, we can reduce to the case where every edge of $\delta(t)$ is a crossing edge in the optimal solution for every $t \in S$.*

A similar result (without a lower bound) finds the remaining crossing edges of a solution.

▶ **Lemma 11** (⋆). *Given an input from stage 1, with p edges already marked as crossing, in $O(4^{2k-p}m)$ time and 4^{2k-p} guesses we can reduce to the case where every edge of G is crossing in the optimal solution.*

After stage 2, the remaining graph contains at most k edges, hence at most $O(k)$ vertices, and it only remains to find the min-cost labelling of the non-terminal vertices. In the absence of any stronger structural properties of the metric μ, this last phase can be completed in $|D|^{O(k)}O(m)$ time. Theorem 2 follows.

4.2 A kernel for any metric

We next show that Zero Extension has a kernel of $O(k^{s+1})$ vertices for any metric μ, where k is a bound on the number of crossing edges of a solution and s is the number of labels of μ. Moreover, the kernel can be computed without access to μ. This gives us a kind of *metric sparsifier* for (G, S), up to parameters k and s, as follows. The result is an adaptation of the kernel for s-Multiway Cut of Kratsch and Wahlström [17]. For an instance I, let $\mathbf{cost}(I, k)$ be the minimum cost of a labelling with at most k crossing edges (otherwise ∞).

▶ **Theorem 12** (⋆). *Let $s \geqslant 3$ be a constant. For every graph $G = (V, E)$ with a set S of terminals, $|S| \leqslant s$, and integer k, there is a randomized polynomial-time computable set $Z \subseteq E$ with $|Z| = O(k^{s+1})$ such that for any instance $I = (G, \tau, \mu, q)$ of* Zero Extension *with S being the set of terminals in I and μ having at most s labels, if $\mathbf{cost}(I, k) < \infty$ then there is a τ-extension λ with crossing number at most k and cost $\mathbf{cost}(I, k)$ such that every crossing edge of λ is contained in Z.*

By contracting any edges not in $\bigcup_{t \in S} \delta(t) \cup Z$ we then get the kernelized instance (G', S).

5 Tree metrics: Gap algorithms and VCSP relaxations

In this section we present more powerful algorithms parameterized by the gap parameter for problems where the metric embeds into a tree metric. We begin by a purely combinatorial algorithm for Zero Extension on leaf metrics, then we move on to the more general Zero Extension and Metric Labelling problems for general induced tree metrics. The algorithms for the latter problems rely on the *domain consistency* property of the relaxation, which allows us to solve the problem by simply branching on the value of a single variable at a time. This property is shown by way of a detour into an analysis of properties of VCSP instances whose cost functions are *weakly tree submodular*, which is a tractable problem class containing tree metrics. The algorithms for these problems are then straight-forward.

At this point, we need to address a subtlety regarding the input cost function μ. So far, the cost function only had to obey basic properties that are easily verifiable or could be seen as a 'promise'. However, some of our arguments below will explictly need the tree T that induces the metric. Luckily this issue has been solved already: given a induced tree metric μ over D in matrix form, one can in time $O(|D|^2)$ compute a tree that induces μ [5]. If μ is a leaf metric, the output will obviously have D as the leaves of T. In conclusion, we will tacitly assume that we have access to the tree T in the following.

5.1 Leaf metrics: A duality approach

The μ-Edge Disjoint Packing problem asks to find an edge-disjoint packing $\mathcal{P}$ of paths whose endpoints both lie in a terminal set $S \subseteq V(G)$ that maximizes $\mathrm{pack}(\mu, G, S) := \sum_{P \in \mathcal{P}} \mu(s_P, t_P)$ (where s_P, t_P denote start- and endpoint of P). Hirai and Pap [10] show that if μ is a tree metric then $\mathrm{pack}(\mu, G, S) = \min_\lambda \max_{F \subseteq E} \sum_{uv \in E \setminus F} \mu(\lambda(u), \lambda(v))$, where λ is a zero-extension of the terminal-set S and the sets $F \subseteq E$ are edges whose deletion leaves every non-terminal vertex with an even degree. It follows that the maximum value of a half-integral τ-path packing is just the minimum cost of a τ-extension λ, since a half-integral path-packing is just a path-packing in the graph where every edge of G has been duplicated.

Let in the following $I = (G, \tau, \mu, q)$ be an instance of Zero Leaf Extension, where μ is a leaf metric over a tree T with leaves D. Let $\hat{\mu} = \mathrm{d}_T$ be the underlying tree metric. We define the *relaxed instance* $\hat{I} = (G, \tau, \hat{\mu}, q)$. Let $\mathbf{opt}(I)$, $\mathbf{opt}(\hat{I})$ denote the set of optimal solutions for the integral and the relaxed instance, respectively. Using this notation, we can summarize the duality: Given a relaxed instance $\hat{I}$, there exists a half-integral τ-path-packing $\mathcal{P}$ of cost precisely $\mathbf{cost}(\hat{I})$. In the following we will assume, by the usual identification argument, that τ is a bijection and a τ-path packing is equivalent to an S-path packing.

▶ **Lemma 13** (⋆). *Let $\mathcal{P}$ be an half-integral τ-path packing with $\frac{1}{2} \sum_{P \in \mathcal{P}} \mu(\tau(s_P), \tau(t_P)) = \mathbf{cost}(\hat{I})$. Let $\lambda \in \mathbf{opt}(\hat{I})$ be a relaxed optimum and let $P \in \mathcal{P}$ with endpoints s, t. Then $\mathbf{cost}_{\hat{\mu}}(\lambda, P) = \mu(\tau(s), \tau(t))$.*

A direct consequence is that if we trace an s-t-path $P \in \mathcal{P}$, then the labels assigned by any relaxed optimum λ to P induce a monotone sequence from s to t in T. That is, not only will we only encounter those labels that lie on $T[s,t]$, we also will encounter them 'in order'.

Consider an edge $xy \in E(T)$. Then, as a consequence of Lemma 13 the set of edges $C_{xy}(\lambda) = \{uv \in E(G) \mid \lambda(u) \in T_x, \lambda(v) \in T_y\}$ between the vertex sets with labels in T_x and T_y, respectively, must be saturated by paths of the packing $\mathcal{P}$. For cuts right above leafs of T, this implies the following.

▶ **Lemma 14** ($\star$). *Let S be the vertices labelled by τ in G and assume that τ is a bijection. Let C be any minimum $(x, S-x)$-cut for some terminal $x \in S$. Then every optimal, half-integral S-path-packing in G will saturate C.*

▶ **Theorem 4.** *Let $I = (G, \tau, \mu, q)$ be an instance of* Zero Extension *where μ is a leaf metric on a set of labels D in a tree T, and let $\hat{I} = (G, \tau, \hat{\mu}, q)$ be the relaxed instance. Let $p = \mathbf{cost}(\hat{I})$. Then we can solve I in time $O(|D|^{q-p}|T|m + |T|\phi(n,m))$, where ϕ is the time needed to run a max-flow algorithm.*

Proof sketch. We first construct for every edge $ij \in T$ a flow network H_{ij} from G as follows: let D_i be those leaves that lie in the same component as i in $T - ij$ and D_j all others. Then H_{ij} is obtained from G by identifying all terminals $\tau^{-1}(D_i)$ into a source s and all terminals $\tau^{-1}(D_j)$ into a sink t. For each H_{ij} we compute a maximum flow f_{ij} in time $\phi(n,m)$. We can show that for every $\lambda \in \mathbf{opt}(\hat{I})$ it holds that $\sum_{xy \in T} |f_{ij}| = \sum_{xy \in T} |C_{xy}(\lambda)| = \mathbf{cost}_\mu(\hat{I})$. Note that we can also, in linear time, find the furthest cuts $\mathrm{C}_{\max}(x)$ for terminals $x \in S$ using the residual network of (H_{ij}, f_{ij}) with $i = \tau(x)$ and j the parent of i in T.

Next, we test whether G contains a vertex u that is not part of a furthest min-cut $\mathrm{C}_{\max}(x)$ for any $x \in S$; such a vertex cannot take an integral value in any relaxed optimum. We then branch on the $|D|$ possible integral values for u: for $x \in S$ with $\tau(x) \in D$ being the chosen integral value, we update the networks (H_{ij}, f_{ij}) by adding an edge xu of infinite capacity, then augment the flow. The number of augmentations is $\leqslant k - p$, as each augmentation witnesses the increase of p and thus the decrease of the parameter. We charge each augmentation to a level of the search tree and thus spend only $O(m)$ time per flow f_{ij}, for a total of $O(|T|m)$.

Otherwise, we find that every vertex of the current graph G is contained in at least one furthest min-cut. It can be shown (see full version) that the intersection of three or more such cuts is empty. Consequently, the graph decomposes into sets whose label is either fixed or is one of two possible values. A simple cut argument shows that we can fix one of the two labels greedily for the latter, and we construct an integral solution that matches the relaxed optimum. ◀

5.2 VCSP toolkit

Given a set of cost functions Γ over a domain D, an instance I of VCSP(Γ) is defined by a set of variables V and a sum of valued constraints $f_i(\bar{v}_i)$, where for each i, $f_i \in \Gamma$ and $\bar{v}_i$ is a tuple of variables over V. We write $f_i(\bar{v}) \in I$ to signify that $f_i(\bar{v})$ is a valued constraint in I. It is known that the tractability of a VCSP is characterized by certain algebraic properties of the set of cost functions. In full generality, such conditions are known as *fractional polymorphisms* for the finite-valued case and more general *weighted polymorphisms* in the general-valued case. Dichotomies are known in these terms both for the finite-valued [30] and general case of VCSP [16], i.e., characterizations of each VCSP as being either in P or NP-hard. We will only need a less general term.

A *binary multimorphism* $\langle \circ, \bullet \rangle$ of a language Γ over a domain D is a pair of binary operators that satisfy $f(\bar{x}) + f(\bar{y}) \geqslant f(\bar{x} \circ \bar{y}) + f(\bar{x} \bullet \bar{y})$, for all $f \in \Gamma, \bar{x}, \bar{y} \in D^{\mathbf{ar}(f)}$, where

$\mathbf{ar}(f)$ is the arity of f and where we extend the binary operators to vectors by applying them coordinate-wise. An operator $\circ$ is *idempotent* if $x \circ x = x$ for every $x \in D$, and *commutative* if $x \circ y = y \circ x$. A (finite, finite-valued) language Γ with a binary multimorphism where both operators are idempotent and commutative is solvable in polynomial time via an LP-relaxation [30]. The most basic example is the Boolean domain $D = \{0, 1\}$, in which case the multimorphism $\langle \wedge, \vee \rangle$ corresponds to the well-known class of *submodular functions*, which is a tractable class that generalizes cut functions in graphs.

The following is folklore, but will be important to our investigations. Again, the corresponding statements apply for arbitrary fractional polymorphisms, but we only give the version we need in the present paper.

▶ **Definition 15** (Preserved under equality)**.** Let f be a function that admits a multimorphism $\langle \circ, \bullet \rangle$. We say that two tuples $\bar{x}, \bar{y} \in D^{\mathbf{ar}(f)}$ are *preserved under equality* if $f(\bar{x}) + f(\bar{y}) = f(\bar{x} \circ \bar{y}) + f(\bar{x} \bullet \bar{y})$. For a relation $R \subseteq D^{\mathbf{ar}(r)}$, we say that f is *preserved under equality in* R if every pair of tuples $\bar{x}, \bar{y} \in R$ is preserved under equality and $\bar{x} \circ \bar{y}, \bar{x} \bullet \bar{y} \in R$.

▶ **Lemma 16** ($\star$)**.** *Let Γ be a language of cost functions that admit a multimorphism $\langle \circ, \bullet \rangle$ and let $\lambda_1, \lambda_2 \in \mathbf{opt}(I)$ for some instance I of* VSCP(Γ)*. Then for every valued constraint $f(\bar{v}) \in I$ it holds that $f(\lambda_1(\bar{v})) + f(\lambda_2(\bar{v})) = f((\lambda_1 \circ \lambda_2)(\bar{v})) + f((\lambda_1 \bullet \lambda_2)(\bar{v}))$, where $f(\lambda(\bar{v})) = f(\lambda(v_1), \ldots, \lambda(v_r))$ for $\bar{v} = v_1, \ldots, v_r$ is the value of $f(\bar{v})$ under λ. In other words, every valued constraint $f(\bar{v}) \in I$ is preserved under equality in $\mathbf{opt}(I)$.*

To illustrate, let us return again to the case of graph cut functions and submodularity over the Boolean domain. Let $G = (V, E)$ be an undirected graph, and define the cut function $f_G \colon 2^V \to \mathbb{Z}$ as $f_G(S) = |\delta(S)|$. Then f_G is the sum over binary valued constraints $f(u, v) = [u \neq v]$ over all edges $uv \in E$, in Iverson bracket notation. Since a single valued constraint $f(u, v)$ is submodular, the same holds for the cut function as a whole. Then Lemma 16 specialises into the statement that for two sets $A, B \subset V$ such that $\delta(A), \delta(B)$ are minimum s-t-cuts in G for some $s, t \in V$, there is no edge between $A \setminus B$ and $B \setminus A$. This kind of observation is a common tool in, e.g., graph theory and approximation algorithms.

The above lemma will be very useful when reasoning about the structure of $\mathbf{opt}(I)$ subject to more complex multimorphisms, as we will define next.

5.3 Submodularity on trees

Let $\preceq_T$ denote the ancestor relationship in a rooted tree T. For a path $P[x, y] \subseteq T$, let z_1, z_2 be the middle vertices of $P[x, y]$ (allowing $z_1 = z_2$ in case $P[x, y]$ has odd length) such that $z_1 \preceq_T z_2$. Define the commutative operators $\curvearrowleft, \curvearrowright$ as returning exactly those two mid vertices, e.g. $x \curvearrowleft y = y \curvearrowleft x = z_1$ and $x \curvearrowright y = y \curvearrowright x = z_2$. Languages admitting the multimorphism $\langle \curvearrowleft, \curvearrowright \rangle$ are called *strongly tree-submodular*.

Define the commutative operator $\uparrow$ to return the common ancestor of two nodes x, y in a rooted tree T. Define $x \nearrow y$ to be the vertex z on $P[x, y]$ which satisfies $\mathrm{d}_T(x, z) = \mathrm{d}_T(y, x \uparrow y)$. In other words, to find $z = x \nearrow y$, we measure the distance from y to the common ancestor of x and y and walk the same distance from x along $P[x, y]$. Languages that admit $\langle \uparrow, \nearrow \rangle$ as a multimorphism are called *weakly tree-submodular*. In particular, all strongly tree-submodular languages are weakly tree-submodular [15]. Tree-metric are, not very surprisingly, strongly tree-submodular:

▶ **Lemma 17** ($\star$)**.** *Every tree-metric is strongly (and thus also weakly) tree-submodular for every rooted version of the tree.*

We will need the following characterization of which value-pairs are preserved under equality by strong tree submodularity for tree distance functions. The proof follows from a long case analysis which we omit here.

▶ **Lemma 18 (⋆).** *Two tuples $(a,b),(x,y) \in V(T) \times V(T)$ are preserved under equality by d_T with multimorphism $\langle \curvearrowleft, \curvearrowright \rangle$ iff all four nodes lie on a single path P in T* and *either $a,b \leqslant_P x,y$ or $a,x \leqslant_P b,y$.*

▶ **Corollary 19 (⋆).** *Let d_T be preserved under equality in R for some $R \subseteq V_T \times V_T$, with at least one pair $(a,b) \in R$ with $a \neq b$. Then there is a path P in T which can be oriented as a directed path such that for every pair $(a,b) \in R$ the nodes a and b lie on P with $a \preceq_P b$.*

5.4 The domain consistency property

Consider a problem VCSP(Γ) over a domain D_I and a discrete relaxation VCSP(Γ') of VCSP(Γ) over a domain $D \supseteq D_I$. We say that the relaxation has the *domain consistency property* if the following holds: for any instance I of VCSP(Γ'), if for every variable v there is an optimal solution to I where v takes a value in D_I, then there is an optimal solution where all variables take values in D_I, i.e. an optimal solution to the original problem at the same cost. We show that the discrete relaxations of Zero Extension and Metric Labelling on induced tree metrics have the domain consistency property, allowing for FPT algorithms under the gap parameter via simple branching algorithms.

The result builds on a careful investigation of the binary constraints that $\mathbf{opt}(I)$ can induce on a pair of vertices $u,v \in V$, starting from Corollary 19. For the rest of the section, let us fix a relaxed instance $I = (G = (V,E), \tau, \mu, q)$ of Zero Extension where μ is a tree metric defined by a tree T, and the original (non-relaxed) metric is the restriction of μ to a set of nodes D_I. Note that I can be expressed as a VCSP instance using assignments and the cost function μ. Let $\mathbf{opt}$ be the set of optimal labellings. For a vertex $v \in V$, let $D(v)$ denote the set $\{\lambda(v) \mid \lambda \in \mathbf{opt}\}$, and let $D_I(v) = D_I \cap D(v)$. Furthermore, for a pair of vertices $u,v \in V$, let $R(u,v) = \{(\lambda(u),\lambda(v)) \mid \lambda \in \mathbf{opt}\}$ be the projection of $\mathbf{opt}$ onto (u,v), and $R_I(u,v) = R(u,v) \cap (D_I \times D_I)$ the integral part of this projection. We begin by observing that the "path property" of Corollary 19 applies to all vertices and edges in $\mathbf{opt}$.

▶ **Lemma 20 (⋆).** *For every vertex v that lies in a connected component of G containing at least one terminal, $D(v)$ is a path in T. Furthermore, for every edge $uv \in E$, $R(u,v)$ embeds into the transitive closure of a directed path in T.*

Next, we show the main result of this section: if u and v is a pair of variables, then whether or not there is an edge uv in E, the constraint $R(u,v)$ induced on u and v by $\mathbf{opt}$ is only non-trivial on values in $D(u) \cap D(v)$.

▶ **Lemma 21 (⋆).** *Let u and v be a pair of variables and $a \in D(u)$, $b \in D(v)$ a pair of values. If $(a,b) \notin R(u,v)$, then $a,b \in D(u) \cap D(v)$ and $a \neq b$.*

This gives us the following algorithmic consequence.

▶ **Lemma 22.** *There is a labelling $\lambda \in \mathbf{opt}$ such that for every variable v with $D_I(v)$ non-empty, we have $\lambda(v) \in D_I$.*

Proof sketch. Order $V(T)$ such that the values D_I come first, and assign to every variable v the value of $D(v)$ that is earliest in this ordering. By Lemma 21, this gives an assignment that is consistent with $R(u,v)$ for every pair of variables $u,v \in V$, and since $\mathbf{opt}$ has a majority polymorphism, this implies that the resulting assignment is in $\mathbf{opt}$. ◀

Let us for reusability spell out the explicit assumptions and requirements made until now.

▶ **Theorem 23** (⋆). *Let $I = (G = (V, E), \tau, \mu, q)$ be an instance of* Zero Extension *with no isolated vertices and where every connected component of G contains at least two terminals, and where μ is an induced tree metric for some tree T and integral nodes $D_I \subseteq T$. Additionally, assume a collection of cost functions $\mathcal{F} = (f_i(\bar{v}_i))_{i=1}^m$ has been given, where for every f_i the scope is contained in V and where f_i is weakly tree submodular for every rooted version of T. Let I' be the VCSP instance created from the sum of the cost functions of I and $\mathcal{F}$. Then I' has the domain consistency property, i.e., there is an integral relaxed optimum if and only if every vertex v is integral in at least one relaxed optimum of I'.*

5.5 Gap algorithms for general induced tree metrics

By Theorem 23, we get FPT algorithms parameterized by the gap parameter $q - p$.

▶ **Theorem 3.** *Let $I = (G, \tau, \mu, q)$ be an instance of* Zero Extension *where μ is an induced tree metric on a set of labels D in a tree T, and let $\hat{I} = (G, \tau, \hat{\mu}, q)$ be the relaxed instance. Let $p = \mathbf{cost}(\hat{I})$. Then we can solve I in time $O(|D|^{q-p}|T||D|nm)$.*

Proof. This algorithm is similar to the algorithm for a leaf metric, except that we are not as easily able to test whether every variable has an integral value in **opt**. By the results of Section 5.1, the value of **opt** is witnessed by the collection of min-cuts for edges in T; we will use this as a value oracle for I. We initially compute a max-flow across every edge of T, then for every assignment made we can compute the new value of **opt** using $O(|T|)$ calls to augmenting path algorithms. This allows us to test for optimality of an assignment in $O(|T|m)$ time. The branching step then in general iterates over at most n variables, testing at most $|D|$ assigned values for each, and testing for optimality each time. Hence the local work in a single node of the branching tree is $O(|T||D|nm)$. This either produces a variable for branching on or (by Theorem 23) produces an integral assignment, and in each branching step the value of p increases but q does not. The time for the initial max-flow computation is eaten by the factor $|T|nm$. The result follows. ◀

For Metric Labelling, we first need to restrict the unary cost functions. Intuitively, the property is analogous to linear interpolation or convexity, applied along paths of the tree. The precise definition is as follows.

▶ **Lemma 24** (⋆). *Let $f\colon V(T) \to \mathbb{R}$ be a unary function on a tree T. Then f is weakly tree submodular on T for every choice of root $r \in V_T$ if and only if it observes the following* interpolation property*: for any nodes $u, v \in V(T)$, at distance $\mathrm{d}_T(u, v) = d$, and every $i \in [d-1]$, let w_i be the node on $T[u, v]$ satisfying $d_T(u, w_i) = i$. Then for any such choice of u, v and i, it holds that $f(w_i) \leqslant ((d-i)/d)f(u) + (i/d)f(v)$.*

Let $f_0\colon U \to \mathbb{Z}^+$ be a non-negative function defined on a subset U of the nodes of a tree T. We say that f_0 *admits an interpolation on* T if there is an extension $f\colon V(T) \to \mathbb{Z}^+$ with the interpolation property such that $f(v) = f_0(v)$ for every $v \in U$. Note that this only restricts the values $f_0(u)$ for nodes $u \in U$ that lie on a tree path between two other nodes $u_1, u_2 \in U$. In particular, if U is the set of leaves of T, then every function f_0 admits an interpolation by simply padding with zero values (although stronger interpolations are in general both possible and desirable).

We get the following.

▶ **Theorem 5.** *Let $I = (G, \sigma, \mu, q)$ be an instance of* Metric Labelling *where μ is an induced tree metric for a tree T and a set of nodes $D \subseteq V(T)$, and where every unary cost $\sigma(v, \cdot)$ admits an interpolation on T. Let $\hat{I} = (G, \hat{\sigma}, \hat{\mu}, q)$ be the relaxed instance, and let $p = \mathbf{cost}(\hat{I})$. Then the instance I can be solved in time $O^*(|D|^{q-p})$. In particular, this applies for any σ if D is the set of leaves of T.*

Proof. Assume that G is connected, or else repeat the below for every connected component of G. Select two arbitrary vertices $u, v \in V$ and exhaustively guess their labels; in the case that you guess them to have the same label, identify the vertices in I (adding up their costs in σ) and select a new pair to guess on. Note that this takes at most $O(|D|^2 n)$ time, terminating whenever you have guessed more than one label in a branch or when you have guessed that all vertices are to be identical. This guessing phase can only increase the value of p. We may now treat u and v as terminals, and the instance I as the sum of a Zero Extension instance on those two terminals and a collection of additional unary cost functions $\sigma(v', \cdot)$, as in Theorem 23. Note that the resulting VCSP is tractable, i.e., the value of an optimal solution can be computed in polynomial time. The running time from this point on consists of iterating through all variables verifying whether each one has an integral value in some optimal assignment, and branching exhaustively on its value if not. ◀

In particular, as noted, for a leaf metric μ the algorithm applies without any assumptions on σ (and without T being explicitly provided).

6 Conclusion

We have given a range of algorithmic results for the Zero Extension and Metric Labelling problems from a perspective of parameterized complexity. Most generally, we showed that Zero Extension is FPT parameterized by the number of *crossing edges* of an optimal solution, i.e. the number of edges whose endpoints receive distinct labels, for a very general class of cost functions μ that need not even be metrics. This is a relatively straight-forward application of the technique of recursive understanding [3].

For the case that μ is a metric we gave two stronger results for the same parameter. First, we showed a linear-time FPT algorithm, with a better parameter dependency, using an important separators-based algorithm. Second, and highly surprisingly, we show that every graph G with a terminal set S admits a polynomial-time computable, polynomial-sized *metric sparsifier* G', with $O(k^{s+1})$ edges, such that (G', S) mimics the behaviour of (G, S) over *any* metric on at most s labels (up to solutions with crossing number k). This is a direct and seemingly far-reaching generalization of the polynomial kernel for s-Multiway Cut [17], which corresponds to the special case of the uniform metric.

Finally, we further developed the toolkit of discrete relaxations to design FPT algorithms under a *gap parameter* for Zero Extension and Metric Labelling where the metric is an induced tree metric. This in particular involves a more general FPT algorithm approach, supported by an applicability condition of *domain consistency*, relaxing the previously used persistence condition.

Let us highlight some questions. First, is there a lower bound on the size of a metric sparsifier for s labels for Zero Extension? This is particularly relevant since the existence of a polynomial kernel for s-Multiway Cut whose degree does not scale with s is an important open problem, and since the metric sparsifier is a more general result.

Second, can the FPT algorithms for induced tree metrics parameterized by the relaxation gap be generalised to restrictions of other tractable metrics, such as graph metrics for median

graphs or the most general tractable class of orientable modular graphs [9]? Complementing this, what are the strongest possible gap parameters that allow FPT algorithms for metrics that are either arbitrary, or do not explicitly provide their relaxation?

References

1. J. Besag. On the statistical analysis of dirty pictures. *J. Royal Stat. Soc. Ser. B*, pages 259–302, 1986.
2. J. Chen, Y. Liu, and S. Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009.
3. R. Chitnis, M. Cygan, M. Hajiaghayi, M. Pilipczuk, and M. Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM J. Sci. Comput.*, 45(4):1171–1229, 2016.
4. J. Chuzhoy and J. Naor. The hardness of metric labeling. *SIAM J. Sci. Comput.*, 36(5):1376–1386, 2007.
5. J.C. Culberson and P. Rudnicki. A fast algorithm for constructing trees from distance matrices. *Inf. Process. Lett.*, 30(4):215–220, 1989.
6. J. Fakcharoenphol, C. Harrelson, S. Rao, and K. Talwar. An improved approximation algorithm for the 0-extension problem. In *Proc. of 14th SODA*, pages 257–265. SIAM, 2003.
7. J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004.
8. H. Hirai. The maximum multiflow problems with bounded fractionality. *Math. Oper. Res.*, 39(1):60–104, 2014.
9. H. Hirai. Discrete convexity and polynomial solvability in minimum 0-extension problems. *Math. Program.*, 155(1-2):1–55, 2016.
10. H. Hirai and G. Pap. Tree metrics and edge-disjoint S-paths. *Math. Program.*, 147(1-2):81–123, 2014.
11. Y. Iwata, M. Wahlström, and Y. Yoshida. Half-integrality, LP-branching, and FPT algorithms. *SIAM J. Comput.*, 45(4):1377–1411, 2016.
12. Y. Iwata, Y. Yamaguchi, and Y. Yoshida. Linear-time FPT algorithms via half-integral non-returning A-path packing. *CoRR*, abs/1704.02700, 2017.
13. H. Karloff, S. Khot, A. Mehta, and Y. Rabani. On earthmover distance, metric labeling, and 0-extension. *SIAM J. Sci. Comput.*, 39(2):371–387, 2009.
14. A.V. Karzanov. Minimum 0-extensions of graph metrics. *Eur. J. Comb.*, 19(1):71–101, 1998.
15. V. Kolmogorov. Submodularity on a tree: Unifying L$^\natural$-natural-convex and bisubmodular functions. In *Proc. of 36th MFCS*, pages 400–411. Springer, 2011.
16. V. Kolmogorov, A. A. Krokhin, and M. Rolínek. The complexity of general-valued CSPs. *SIAM J. Comput.*, 46(3):1087–1110, 2017.
17. S. Kratsch and M. Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *Proc. of 53rd FOCS*, pages 450–459. IEEE Computer Society, 2012.
18. L. Lovász. Flats in matroids and geometric graphs. In *Proc. of 6th Br. Comb. Conf.*, Combinatorial Surveys, pages 45–86, 1977.
19. R. Manokaran, J. Naor, P. Raghavendra, and R. Schwartz. SDP gaps and UGC hardness for multiway cut, 0-extension, and metric labeling. In *Proc. of 40th STOC*, pages 11–20, 2008.
20. D. Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006.
21. D. Marx. A parameterized view on matroid optimization problems. *Theor. Comput. Sci.*, 410(44):4471–4479, 2009.

22 D. Marx and I. Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM J. Comput.*, 43(2):355–388, 2014.

23 M. Naor, L.J.Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *Proc. of 36th FOCS*, pages 182–191. IEEE, 1995.

24 M. Óskarsdóttir, C. Bravo, W. Verbeke, C. Sarraute, B. Baesens, and J. Vanthienen. A comparative study of social network classifiers for predicting churn in the telecommunication industry. In *Proc. of 8th ASONAM*, pages 1151–1158. IEEE, 2016.

25 B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proc. of 43rd ACL*, pages 115–124. Association for Computational Linguistics, 2005.

26 J. Picard and H.D. Ratliff. A cut approach to the rectilinear distance facility location problem. *Oper. Res.*, 26(3):422–433, 1978.

27 Felix Reidl and Magnus Wahlström. Parameterized algorithms for zero extension and metric labelling problems. *CoRR*, abs/1802.06026, 2018. `arXiv:1802.06026`.

28 A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

29 J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. *J. ACM*, 49(5):616–639, 2002.

30 J. Thapper and S. Živný. The complexity of finite-valued CSPs. *J. ACM*, 63(4):37, 2016.

31 N. Vikas. A complete and equal computational complexity classification of compaction and retraction to all graphs with at most four vertices and some general results. *J. Comput. Syst. Sci.*, 71(4):406–439, 2005.

32 M. Wahlström. LP-branching algorithms based on biased graphs. In *Proc. of 28th SODA*, pages 1559–1570. SIAM, 2017.

An Operational Characterization of Mutual Information in Algorithmic Information Theory

Andrei Romashchenko[1]
LIRMM, Univ Montpellier, CNRS, Montpellier, France; on leave from IITP RAS
andrei.romashchenko@lirmm.fr

Marius Zimand
Department of Computer and Information Sciences, Towson University, Baltimore, MD
http://orion.towson.edu/~mzimand

Abstract

We show that the mutual information, in the sense of Kolmogorov complexity, of any pair of strings x and y is equal, up to logarithmic precision, to the length of the longest shared secret key that two parties, one having x and the complexity profile of the pair and the other one having y and the complexity profile of the pair, can establish via a probabilistic protocol with interaction on a public channel. For $\ell > 2$, the longest shared secret that can be established from a tuple of strings $(x_1, ..., x_\ell)$ by ℓ parties, each one having one component of the tuple and the complexity profile of the tuple, is equal, up to logarithmic precision, to the complexity of the tuple minus the minimum communication necessary for distributing the tuple to all parties. We establish the communication complexity of secret key agreement protocols that produce a secret key of maximal length, for protocols with public randomness. We also show that if the communication complexity drops below the established threshold then only very short secret keys can be obtained.

2012 ACM Subject Classification Mathematics of computing → Information theory, Theory of computation → Communication complexity, Theory of computation → Probabilistic computation

Keywords and phrases Kolmogorov complexity, mutual information, communication complexity, secret key agreement

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.95

Related Version A full version of the paper is available at https://arxiv.org/abs/1710.05984.

Acknowledgements We are grateful to Bruno Bauwens for his insightful comments. We thank Tarik Kaced for attracting our attention to [5].

1 Introduction

Mutual information is a concept of central importance in both information theory (IT) and algorithmic information theory (AIT), also known as Kolmogorov complexity. We show an interpretation of mutual information in AIT, which links it to a basic concept from cryptography. Even though a similar interpretation was known in the IT framework, an operational characterization of mutual information in AIT has been elusive till now.

To present our result, let us consider two strings x and y. It is common to draw a Venn-like diagram such as the one in Figure 1 to visualize the information relations between

[1] Supported in part by ANR-15-CE40-0016-01 RaCAF grant.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 95; pp. 95:1–95:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

them. As explained in the figure legend there are six important regions. The regions (1) to

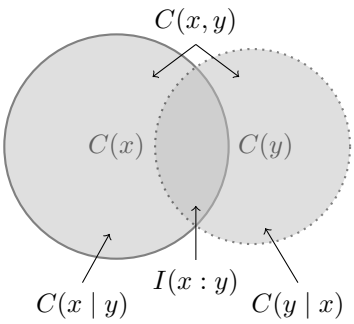


Figure 1 Two strings x and y, and their information. There are six regions that we distinguish: (1) The left solid circle represents the information in x, as given by its Kolmogorov complexity, denoted $C(x)$; (2) The right dotted circle represents the information in y, denoted $C(y)$; (3) The entire grey region (the two circles taken together) represents the information in x and y, denoted $C(x,y)$; (4) The light-grey region in the first circle represents the information in x conditioned by y, denoted $C(x \mid y)$; (5) The light-grey region in the second circle represents the information of y conditioned by x, denoted $C(y \mid x)$; and (6) the dark-grey region in the middle represents the mutual information of x and y, denoted $I(x:y)$.

(5) have a clear operational meaning. For instance, $C(x)$ is the length of a shortest program that prints x, $C(x \mid y)$ is the length of a shortest program that prints x when y is given to it, and so on. On the other hand, the mutual information $I(x:y)$ from region (6) is defined by a formula: $I(x:y) = C(x) + C(y) - C(x,y)$. Intuitively, it is the information shared by x and y. But is there an operational interpretation of the mutual information? As mentioned above, we give a positive answer: The mutual information of x and y is essentially equal to the length of a longest shared secret key that two parties, one having x and the other one having y, and both parties also possessing the complexity profile of the two strings, can establish via a probabilistic protocol.

The following simple example illustrates the above concepts. Suppose that Alice and Bob want to agree on a common secret key. If they could meet face-to-face, they could just generate such a key by, say, flipping a coin. Unfortunately, they cannot meet in person and what makes the situation really troublesome is that they can only communicate through a public channel. There is however a gleam of hope because Alice knows a random line x in the affine plane over the finite field with 2^n elements, and Bob knows a random point y on this line. The line x is specified by the slope a and the intercept b and the point y by its two coordinates c and d. Therefore each of x and y has $2n$ bits of information, but, because of the geometrical correlation, together they have $3n$ bits of information. Thus, in principle, Alice and Bob share n bits. Can they use them to obtain a common secret key? The answer is yes: Alice sends a to Bob, Bob, knowing that his point is on the line, finds x, and now they can use b as the secret key, because the adversary has only seen a, and a and b are independent.

It may appear that the geometrical relation between x and y is crucial for the above solution. In fact it is just a red herring and Alice and Bob can agree on a common secret key in a very general setting. To describe it, we consider the scenario in which Alice has a random string x and Bob has a random string y. If $x = y$, then Alice and Bob can use their common string as a secret key in an encryption scheme (such as the one-time pad) and achieve perfect information-theoretical security. What happens if x and y are not equal,

but only correlated? Somewhat surprisingly, for many interpretations of "correlated," they can still agree on a shared secret key via interaction on a public channel (for instances of this assertion, see [15, 3, 17, 1]). In this paper, we look at this phenomenon using the very general framework of algorithmic information theory to measure the correlation of strings.

1.1 Our contributions

Characterization of mutual information. In a secret key agreement protocol, Alice and Bob, on input x and respectively y, exchange messages and compute a common string that is random conditioned by the transcript of the protocol. Such a string is said to be a *shared secret key*. Unless specified otherwise, we use protocols having the following features:

(1) We assume that Alice and Bob also know how their x and y are correlated. In our setting this means that Alice and Bob know the complexity profile of x and y, which, by definition, is the tuple $(C(x), C(y), C(x, y))$.

(2) The protocols are effective and randomized, meaning that Alice and Bob use probabilistic algorithms to compute their messages.

▶ **Theorem 1** (Main Result, informal statement)**.**

1. *There is a secret key agreement protocol that, for every n-bit strings x and y, allows Alice and Bob to compute with high probability a shared secret key of length equal to the mutual information of x and y (up to an $O(\log n)$ additive term).*
2. *No protocol can produce a longer shared secret key (up to an $O(\log n)$ additive term).*

Secret key agreement for three or more parties. Mutual information is only defined for two strings, but secret key agreement can be explored for the case of more strings. Let us consider again an example. Suppose that each of Alice, Bob, and Charles have a point in the affine plane over the finite field with 2^n elements, and that the three points, which we call A, B, C, are collinear. Thus each party has $2n$ bits of information, but together they have $5n$ bits of information, because given two points, the third one can be described with n bits. The parties want to establish a common secret key, but they can only communicate by broadcasting messages over a public channel. They can proceed as follows. Alice will broadcast a string p_A, Bob a string p_B, and Charles a string p_C, such that each party using his/her point and the received information will reconstruct the three collinear points A, B, C. A protocol that achieves this is called an *omniscience protocol* because it spreads to everyone the information possessed at the beginning individually by each party. In the next step, each party will compress the $5n$ bits, comprising the three points, to a string that is random given p_A, p_B, p_C. The compressed string is the common secret key. We will see that up to logarithmic precision it has length $5n - (|p_A| + |p_B| + |p_C|)$. Assuming we know how to do the omniscience protocol and the compression step, this protocol produces a common secret key of length $5n - \mathrm{CO}(A, B, C)$, where $\mathrm{CO}(A, B, C)$ is the minimum communication for the omniscience task for the points A, B, C. In our example, it is clear that each one of p_A, p_B, p_C must be at least n bits long, and that any two of these strings must contain together at least $3n$ bits. Using some recent results from the reference [28], it can be shown that any numbers satisfying these constraints can be used for the omniscience task. It follows that the smallest communication for omniscience is achieved when $|p_A| = |p_B| = |p_C| = 1.5n$, and thus the key has $5n - 4.5n = 0.5n$ bits (Warning: we have ignored in the entire discussion some $O(\log n)$ terms). We show that this holds in general. If ℓ parties have, respectively, one component of a tuple $(x_1, \ldots, x_\ell)$ of n-bit strings, then up to $O(\log n)$ precision, they can produce a common secret key of length $C(x_1, \ldots, x_\ell) - \mathrm{CO}(x_1, \ldots, x_\ell)$, where $\mathrm{CO}(x_1, \ldots, x_\ell)$ is the

minimum communication for the omniscience task. The protocol that produces such a key is probabilistic, and, as was the case for two strings, assumes that each party i has at the beginning of the protocol besides its input string x_i also the complexity profile of the entire tuple $(x_1, \ldots, x_\ell)$. We also show a matching (up to $O(\log n)$) upper bound: no probabilistic protocol can produce a longer secret key.

▶ Remark. The value $\mathrm{CO}(A, B, C)$ is understood as the communication complexity of the *omniscience problem*. However, it can be computed as a function of the Kolmogorov complexities of the involved strings, see Definition 4 below. This fact (the communication complexity of the optimal omniscience protocol depends only on the complexity profile of the inputs) is not trivial and requires a proof.

Communication complexity for secret key agreement. In the protocol in Theorem 1, Alice and Bob exchange $\min(C(x \mid y), C(y \mid x)) + O(\log n)$ bits and obtain with high probability a shared secret key of length $I(x : y) - O(\log n)$. In this protocol we can assume that Alice and Bob use either private random bits, or public random bits. We show that for the model with public random bits, the communication complexity of the protocol is optimal, in the sense that in any protocol with public random bits there are input strings x and y, on which Alice and Bob have to exchange at least $\min(C(x \mid y), C(y \mid x))$ bits. In fact our lower bound is stronger: we show that, for any constants $\delta_1, \delta_2 > 0$, if Alice and Bob use a protocol with communication complexity $(1 - \delta_1) \min(C(x \mid y), C(y \mid x))$ for every input pair x, y, then there are inputs for which the shared secret key that they obtain has length at most $\delta_2 I(x : y)$. That is, if the communication complexity sinks below the threshold $\min(C(x \mid y), C(y \mid x))$, then the size of the common secret key drops to virtually zero. To determine the optimal communication complexity for the model with private random bits remains an open problem.

1.2 Related previous work.

IT vs. AIT. Before reviewing existing related results in the IT and the AIT frameworks, it is useful to understand the distinction between the two theories. In computer science the attribute *random* is mainly used in two (fairly different) contexts: *random processes* and *random objects*. In short, IT, which we also call Shannon's framework, focuses on the former, whereas AIT, which we also call Kolmogorov's framework, focuses on the latter. On the one hand, we may think of an uncertain physical process with unpredictable outcomes, and employ the framework of the classic probability theory (distributions, random variables, etc.). The notion of a *random variable* formalizes the idea of a process like coin tossing. In this context we can measure the uncertainty of a random variable as a whole (by its Shannon's entropy, its min-entropy, etc.), but we can not ask whether one specific outcome is random or not. On the other hand, people use *tables of random numbers*, which are available as specific sequences of digits, written on a disc or printed on a paper. The usefulness of such a table depends on its individual properties: frequencies of digits, presence or absence of hidden regularities, compressibility, etc. The usual way to measure the uncertainty of an individual string of digits is Kolmogorov complexity. In both contexts the formal measures of randomness may or may not involve computational complexity (see, e.g., different versions of pseudoentropy for distributions and the resource bounded variants of Kolmogorov complexity for individual strings). These two formalizations of randomness are connected but not interchangeable.

Both notions of randomness appear in cryptography. For example, in the one-time pad scheme, two parties share a "random" key that remains "secret" for the attacker. It is

common to use Shannon's framework, and therefore the notions of randomness and secrecy are defined in terms of random processes. In the ideal situation both parties should have access to a common source of randomness, e.g., to the results of tossing an unbiased coin (hidden from the adversary). By tossing this coin n times we get a random variable with maximal possible entropy, and thus, in Shannon's framework, the quality of randomness is perfect. But if by chance we obtain a sequence of n zeros, then this specific one-time pad looks pretty useless in any practical application. However, Shannon's information theory provides no vocabulary to complain about this apparently non-random individual key. Antunes *et al.* [2] suggested to use Kolmogorov complexity to measure the "secrecy" of individual instances of a one-time pad or a secret sharing schemes. We have in mind a similar motivation, and in this work a "secret key" is an individual string that is random in the sense of Kolmogorov complexity.

Related work. We start with a brief account of works on secret key agreement in the IT setting. The secret key agreement is a relatively well-studied problem in information theory, motivated, as the name suggests, by applications in information-theoretically secure cryptography. Wyner [27] and Csiszár and Körner [7] have analyzed the possibility of obtaining a shared secret key in the case when one party sends to the other party a single message on a channel from which the eavesdropper can obtain partial information. Maurer [16, 17] considered the case of protocols with several rounds of communication and showed that interaction can be more powerful than one-way transmission. Ahlswede and Csiszár [1] and Maurer [17] have established the tight relation between interactive secret key agreement and mutual information for *memoryless sources*. In the memoryless model, the input data is given by two random variables (X_1, X_2) obtained by n independent draws from a joint distribution, where Alice observes X_1 and Bob observes X_2. Informally stated, the references [17, 1] show that the longest shared secret key that Alice and Bob can establish via an interactive protocol with an arbitrary number of rounds is equal to the mutual information of X_1 and X_2. Csiszár and Narayan [8] go beyond the scenario with two parties, and consider the case of an ℓ-memoryless source $(X_1, \ldots, X_\ell)$ and ℓ parties, each one observing one component of the tuple. They show that the longest shared secret key the ℓ parties can establish via an interactive protocol with an arbitrary number of rounds is equal to the entropy $H(X_1, \ldots, X_\ell)$ of the ℓ-memoryless source from which one subtracts the minimum communication for omniscience. Their result holds also for stationary ergodic sources, which generalize memoryless sources. As one can see, our results are very similar. They have been inspired by the papers [1, 17, 8] and represent the AIT analogue of the results presented above. Our results imply their IT analogues, and can be viewed as more general because they do not require the memoryless or ergodicity properties of sources (in fact they do not require any generative model at all). Regarding the communication complexity of secret key agreement protocols, we only note here that Tiyagi [26] has shown that for memoryless sources it is equal to the difference between interactive common information in Wyner's sense and mutual information. In the full version of this paper we explain how Tyagi's result compares to our results on communication complexity in the AIT framework.

Let us now say a few words about related results from the AIT world. To the best of our knowledge, in AIT there has been no previous works on secret key agreement. However, the general idea of "materialization" of mutual information was studied extensively. Motivated by the intuition that mutual information represents the amount of shared information in two strings, researchers have explored the extent to which mutual information can be materialized more or less effectively. The relevant concept is that of *common information.* Informally, a

string z is a common information string extracted from strings x and y, if z can be "computed" from x, and also from y, where "computed" is taken in a more liberal sense that allows the utilization of a few help bits. In the most common setting of parameters, we require that $C(z \mid x) = O(\log n)$ and $C(z \mid y) = O(\log n)$, where n is the length of x and y and the constant hidden in the $O(\cdot)$ notation depends only on the universal machine. Informally, the common information of x and y is the length of a longest common information string that can be extracted from x and y. It can be shown that up to logarithmic precision common information is upper bounded by mutual information. In an influential paper, Gács and Körner [9] have constructed strings x and y for which the common information is much smaller than the mutual information. Moreover, the property of a pair (x, y) of having common information equal to mutual information does not depend solely on the complexity profile of x and y: There exist pairs (x_1, y_1) and (x_2, y_2) having the same complexity profile, and for (x_1, y_1) the common information and mutual information are equal, whereas for (x_2, y_2) they are not. Muchnik [19] and Romashchenko [22] have strengthened the Gács-Körner theorem in significant ways, by allowing a larger amount of help bits, parameterizing the mutual information of the constructed pair (x, y), and other ways. Chernov et al. [6] presents alternative constructions of strings for which the common information is smaller than mutual information for several regimes of parameters. A nice, self-contained and accessible exposition of this research line can be found in the book of Shen, Vereshchagin and Uspensky [23, Chapter 11].

Thus, previous works have shown negative results regarding the "materialization" of mutual information in AIT. As far as we know, ours is the first positive result. In summary, we now know that computation without communication, even enhanced with help bits, fails to extract the mutual information of two strings, while interactive computation succeeds.

1.3 The basics of algorithmic information theory

Given a Turing machine M, a string p is said to be a *program* (or a *description*) of a string x, if M on input p prints x. We denote the length of a binary string x by $|x|$. The *Kolmogorov complexity* of x relative to the Turing machine M is

$$C_M(x) = \min\{|p| \mid \ p \text{ is a program for } x \text{ relative to } M\}.$$

If U is universal Turing machine, then for every other Turing machine M there exists a string m such that $U(m, p) = M(p)$ for all p, and therefore for every string x, $C_U(x) \leq C_M(x) + |m|$. Thus, if we ignore the additive constant $|m|$, the Kolmogorov complexity of x relative to U is minimal. We fix a universal Turing machine U, drop the subscript U in $C_U(\cdot)$, and denote the complexity of x by $C(x)$. Similarly to the complexity of x, we define the complexity of x conditioned by y as $C(x \mid y) = \min\{|p| \mid \ U \text{ on input } p \text{ and } y \text{ prints } x\}$. We list below a few basic facts about Kolmogorov complexity and introduce some notation:

- For every string x, $C(x) \leq |x| + O(1)$, because a string x is trivially described by itself. (Formally, there is a Turing machine M that, for every x, on input x prints x.)
- Using some standard computable pairing function $\langle \cdot, \cdot \rangle$ that maps pairs of strings into single strings, we define the complexity of a pair of strings by $C(x, y) = C(\langle x, y \rangle)$. Then we can extend this notation to tuples of larger arity.
- We use the convenient shorthand notation $a \leq^+ b$ to mean that $a \leq b + O(\log n)$, where n is a parameter that is clear from the context and the constant hidden in the $O(\cdot)$ notation only depends on the universal machine U. Similarly, $a \geq^+ b$ means $a \geq b - O(\log n)$, and $a =^+ b$ means ($a \leq^+ b$ and $a \geq^+ b$).

- The chain rule (a.k.a. the Kolmogorov–Levin theorem) claims that for all sufficiently long strings x and y, $\big|C(x,y) - (C(x) + C(y \mid x))\big| \le 3(\log C(x) + \log C(y))$.
- The mutual information of two strings x and y is denoted $I(x : y)$, and is defined as $I(x : y) = C(x) + C(y) - C(x, y)$.
- The complexity profile of a tuple of strings $(x_1, \ldots, x_\ell)$ is given by the tuple consisting of the complexities of all non-empty subsets of the strings in the tuple, i.e., it is the tuple $(C(x_V) \mid V \subseteq [\ell], V \neq \emptyset)$. Here x_V denotes the subtuple obtained by taking the components with indices in V (for example if $V = \{1, 2, 7\}$ then $x_V = (x_1, x_2, x_7)$).

1.4 Shared secret keys and protocols for secret key agreement

Let k be a positive integer. A k-rounds two-party protocol for secret key agreement uses two computable functions A and B and runs as follows. The first party has as input a string x_A and uses private randomness r_A, the second party has as input a string x_B and uses private randomness r_B. We assume that the length of r_A (r_B) is determined by x_A (respectively, x_B). The protocol consists of the following calculations:

$$\begin{array}{lll} x_1 &= A(x_A, r_A), & y_1 = B(x_B, r_B, x_1) \\ x_2 &= A(x_A, r_A, y_1), & y_2 = B(x_B, r_B, x_1, x_2) \\ \vdots & & \\ x_k &= A(x_A, r_A, y_1, \ldots, y_{k-1}), & y_k = B(x_B, r_B, x_1, \ldots, x_k). \end{array}$$

The algorithms A and B can handle inputs of different lengths. We also allow them to be partial (i.e., it is possible that the protocol does not converge for some pairs of inputs). Let us fix parameters ϵ and $\delta(n)$. (We assume ϵ is a positive constant and $\delta(n)$ is a constant or a slow growing function, e.g., $O(\log n)$). A protocol *succeeds* with error probability ϵ and randomness deficiency $\delta(n)$ on a pair (x_A, x_B) of n-bit strings if with probability $(1 - \epsilon)$ over r_A, r_B,

$$A(x_A, r_A, t) = B(x_B, r_B, t) \stackrel{def.}{=} z, \tag{1}$$

and

$$C(z \mid t) \ge |z| - \delta(n), \tag{2}$$

where $t = (x_1, y_1, \ldots, x_k, y_k)$ is the transcript of the protocol.

The string z satisfying equation (1) and inequality (2) is called a *shared secret key* output by the protocol on input (x_A, x_B). Note that the shared secret key z is a random variable since it depends not only on the inputs x_A and x_B, but also on the randomness r_A and r_B.

In words, Alice and Bob start with input strings x_A and respectively x_B, use private randomness r_A, and respectively r_B and execute a protocol in which at round i, first Alice sends to Bob the string x_i, and next Bob sends to Alice the string y_i, and at the end Alice and Bob separately compute with high probability a common string z (equation (1)) such that z is random even conditioned by the transcript of the protocol (inequality (2)). Thus, z is secret to an adversary that has observed the protocol and consequently knows the transcript.

The number of rounds in a protocol (parameter k) may depend on the length of the inputs.

2 Main results

We present here our main results. We first show that there exists a secret key agreement protocol which produces a shared secret key of length equal (up to logarithmic precision) to the mutual information of the inputs, provided the two parties know the complexity profile. Next we show that no protocol can produce a longer shared secret key. The formal statements are as follows.

▶ **Theorem 2** (Lower bound). *There exists a secret key agreement protocol with the following property: For every n-bit strings x and y, for every constant $\epsilon > 0$, if Alice's input x_A consists of x, the complexity profile of (x, y) and ϵ, and Bob's input x_B consists of y, the complexity profile of (x, y) and ϵ, then, with probability $1 - \epsilon$, the shared secret key is a string z such that, $C(z \mid t) \geq |z| - O(\log(1/\epsilon))$ and $|z| \geq I(x : y) - O(\log(n/\epsilon))$, where t is the transcript of the protocol. Moreover, the communication consists of a single message sent by Alice to Bob of length $C(x \mid y) + O(\log(n/\epsilon))$, Alice uses $O(\log(n/\epsilon))$ random bits, and Bob does not use any random bits.*

▶ **Theorem 3** (Upper bound). *Let us consider a protocol for secret key agreement, let x_A and x_B be input strings of length n on which the protocol succeeds with error probability ϵ and randomness deficiency $\delta(n)$, and let z be the random string that is the shared secret key output by the protocol, i.e., a string satisfying relations (1) and (2). Then with probability at least $1 - O(\epsilon)$, if n is sufficiently large, $|z| \leq I(x_A : x_B) + \delta(n) + O(\log(n/\epsilon))$, where the constants in the $O(\cdot)$ notation depend on the universal machine, but not on x_A and x_B.*

Theorem 3 establishes the upper bound claimed in the Introduction. Indeed, for any pair of n-bit strings (x, y), suppose that Alice's input x_A consists of x and the complexity profile of (x, y) and Bob's input x_B consists of y and the complexity profile of (x, y). Note that $I(x_A : x_B) =^+ I(x : y)$, because the length of the complexity profile is bounded by $O(\log n)$. Hence, Theorem 3 implies that secret key agreement protocols in which the two parties, besides x and respectively y, are additionally given the complexity profile of their inputs can not produce a secret key that is longer than $I(x : y) + O(\log n)$ (provided the randomness deficiency of the key satisfies $\delta(n) = O(\log n)$).

▶ Remark. In our secret key agreement protocols, the inputs x_A and x_B have two components: $x_A = (x, h_A)$ and $x_B = (y, h_B)$, where the strings x and y are the main components, while h_A and h_B are short helping strings (for example, containing information about how x and y are correlated). The protocols designed in this paper succeed for all input pairs x_A and x_B in which $h_A = h_B =$ (the complexity profile of x and y). In case one or both of h_A and h_B are not equal to the complexity profile, the protocols still halt on every input, but the outputs may be meaningless. However, the proof of Theorem 2 can be adapted to the situation where Alice and Bob are not given the exact value of the complexity profile of (x, y) but only an approximation of this profile. If Alice and Bob are given upper and lower bounds for each component of the complexity profile of (x, y) with precision $\leq \sigma$, for some integer σ, then with probability $1 - O(\epsilon)$ Alice and Bob agree on a common secret z that is incompressible (i.e., $C(z \mid t) \geq |z| - O(\log(1/\epsilon))$ where t is the transcript of the protocol), and the length of z is greater than $I(x : y) - \sigma - O(\log(n/\epsilon))$.

3 Secret key agreement for three or more parties

In this section we analyze secret key agreement for 3 parties, which we call Alice, Bob, and Charles. Alice has a string x_A, Bob has a string x_B, and Charles has a string x_C. They

also have private random bits r_A, respectively r_B and r_C. They run a k-round protocol. In each of the k rounds, each party broadcasts a message to the other two parties, where the message is a string computed from the party's input string and private random bits, and the messages from the previous rounds. After the completion of the k rounds, each party computes a string. The requirement is that with probability at least $1-\epsilon$, they compute the same string, and that this string is random conditioned by the transcript of the protocol.

Formally, a k-round 3-party protocol for secret key agreement uses three computable functions A, B, C, and runs as follows. The first party has as input an n-bit string x_A and uses private randomness r_A, the second party has as input an n-bit string x_B and uses private randomness r_B, and the third party has as input an n-bit string x_C and uses private randomness r_C. The protocol consists of the following calculations:

$$\left.\begin{array}{lll} t_1 = A(x_A, r_A), & t_2 = B(x_B, r_B), & t_3 = C(x_C, r_C), \\ t_4 = A(x_A, r_A, t[1:3]), & t_5 = B(x_B, r_B, t[1:3]), & t_6 = C(x_C, r_C, t[1:3]), \\ t_7 = A(x_A, r_A, t[1:6]), & t_8 = B(x_B, r_B, t[1:6]), & t_9 = C(x_C, r_C, t[1:6]), \\ \vdots & & \end{array}\right\} k \text{ rounds}$$

Each row corresponds to one round and shows the messages that are broadcast in that round, and we use the notation $t[i:j]$ to denote the tuple of messages $(t_i, \ldots, t_j)$. We also denote $t = t[1:3k]$, the entire transcript of the protocol. The protocol succeeds with probability error ϵ and randomness deficiency $\delta(n)$ on the 3-tuple input (x_A, x_B, x_C) if with probability $(1-\epsilon)$ over r_A, r_B, r_C,

$$A(x_A, r_A, t) = B(x_B, r_B, t) = C(x_C, r_C, t) \stackrel{def.}{=} z, \tag{3}$$

and $C(z \mid t) \geq |z| - \delta(n)$.

▶ **Definition 4.**

(1) For each triple of strings (x_1, x_2, x_3), we denote by $S(x_1, x_2, x_3)$ the set of all triples of integers (n_1, n_2, n_3) that satisfy the following inequalities:

$$\begin{array}{ccc} n_1 \geq C(x_1 \mid x_2, x_3), & n_2 \geq C(x_2 \mid x_1, x_3), & n_3 \geq C(x_3 \mid x_1, x_2), \\ n_1 + n_2 \geq C(x_1, x_2 \mid x_3), & n_1 + n_3 \geq C(x_1, x_3 \mid x_2), & n_2 + n_3 \geq C(x_2, x_3 \mid x_1). \end{array}$$

The constraints defining $S(x_1, x_2, x_3)$ will be referred as the *Slepian-Wolf* constraints.

(2) We define $\mathrm{CO}(x_1, x_2, x_3)$ to be the minimal value of $n_1 + n_2 + n_3$ subject to n_1, n_2, n_3 satisfying the Slepian-Wolf constraints. (CO stands for *communication for omniscience.*)

We show that there exists a protocol that on every input tuple (x_A, x_B, x_C) produces with high probability a secret key of length $C(x_A, x_B, x_C) - \mathrm{CO}(x_A, x_B, x_C) - O(\log n)$ (provided the parties have the complexity profile of the input tuple), and that no protocol can produce a secret key of length larger than $C(x_A, x_B, x_C) - \mathrm{CO}(x_A, x_B, x_C) + O(\log n)$.

▶ **Theorem 5** (Upper bound). *Let us consider a 3-party protocol for secret key agreement with error probability ϵ, where the number of random bits is bounded polynomially in the input length. Let (x_A, x_B, x_C) be a 3-tuple of n-bit strings on which the protocol succeeds. Let z be the random variable which represents the secret key computed from the input (x_A, x_B, x_C) and let t be the transcript of the protocol that produces z. Then, for sufficiently large n, with probability $1 - O(\epsilon)$ we have $C(z \mid t) \leq C(x_A, x_B, x_C) - \mathrm{CO}(x_A, x_B, x_C) + O(\log(n/\epsilon))$.*

▶ **Theorem 6** (Lower bound). *There exists a 3-party protocol for secret key agreement with the following characteristics. For every n, for every tuple (x_1, x_2, x_3) of n-bit strings, for*

every $\epsilon > 0$, if Alice's input x_A consists of x_1, the complexity profile of the tuple and ϵ, Bob's input x_B consists x_2, the complexity profile of the tuple and ϵ, and Charles's input x_C consists of x_3, the complexity profile of the tuple and ϵ, then at the end the three parties compute with probability $1 - O(\epsilon)$ a common string z such that

$$C(z \mid t) \geq |z| - O(\log(1/\epsilon)) \text{ and } |z| \geq C(x_1, x_2, x_3) - \mathrm{CO}(x_1, x_2, x_3) - O(\log(n/\epsilon)),$$

where t is the transcript of the protocol.

▶ Remark. Theorem 5 and Theorem 6 remain valid for any constant number $\ell \geq 3$ of parties, with a suitable generalization of the omniscience $\mathrm{CO}(x_1, \ldots, x_\ell)$, see the full version of the paper.

4 Communication complexity of secret key agreement

It is of interest to find the communication complexity for the task of finding a shared secret key having the optimal length of $I(x : y)$. We solve this problem in the model of randomized protocols with *public random bits*, visible to Alice, Bob, and the adversary. This model is obtained by modifying slightly the definition from Section 1.4 (in which the random bits are private): we require that $r_A = r_B = r$ and we change equation (2) to $C(z \mid t, r) \geq |z| - \delta(n)$.

The protocol presented in the proof of Theorem 2 solves the task with communication $\min(C(x \mid y), C(y \mid x)) + O(\log n)$. This protocol can be easily modified to work in the model with public randomness. We argue that within the model with public randomness the communication complexity of this protocol is optimal, up to the $O(\log n)$ term. In what follows we assume as usual that Alice is given a string x and Bob is given a string y, and both parties know the complexity profile of (x, y).

▶ **Theorem 7.** *Let $\epsilon, \delta_1, \delta_2$ be arbitrary positive real constants. There is no secret key agreement protocol with public random bits such that for all inputs x and y,*

1. *the communication complexity of the protocol (the total number of all bits sent by Alice and Bob) is less than $(1 - \delta_1) \min\{C(x \mid y), C(y \mid x)\}$,*
2. *Alice and Bob agree with probability $> \epsilon$ on a common key z such that $C(z \mid t, r) > \delta_2 I(x : y)$, where r is the public randomness and $t = t(x, y, r)$ is the transcript of the protocol.*

5 Our techniques.

It is common for statements in IT (in the Shannon's entropy framework) to have an analogue version in AIT (in the Kolmogorov complexity framework). However, there is no canonical way to translate a result from one setting to the other, and proofs of homologous results in these two frameworks can be drastically different. A textbook example of this phenomenon is the chain rule: it is valid for Shannon's entropy and for Kolmogorov complexity, and the formal expressions of this rule in both frameworks look very similar. However, in Shannon's case this fact is an easy corollary of the definition, while in Kolmogorov's version it requires a nontrivial argument (which is known as the Kolmogorov–Levin theorem). There are more advanced examples of parallel properties (from IT and AIT respectively), where the discrepancy between their proofs is even more striking.

This phenomenon manifests itself in this work as well. Our main results are motivated by similar ones in IT, and there is a close resemblance of statements. As discussed above, this is not surprising. In what follows we explain the relation between our proofs and the proofs of similar statements in Shannon's framework.

The positive results (the existence of communication protocols) use constructions that at the high level are akin to those from their IT counterparts [17, 1, 8]. We employ a similar intuitive idea – manipulations with "fingerprints" of inputs of appropriate lengths[2]. However, the technical machinery is different. In the AIT framework, for communication-efficient protocols, we need quite explicit constructions, while homologous results in IT are usually proven by choosing random encodings. Our constructions are based on a combination of extractors and universal hashing. Our general protocols are not time-efficient and this is to be expected given the high generality of the type of data correlation in the AIT setting. However, for some particular types of correlation (e.g., for a pair of inputs with a bounded Hamming distance), our protocols can be modified to run in polynomial-time. In this case we use the reconciliation technique from [25, 10, 11].

In the negative results (upper bounds for the size of the common secret key, Theorems 2 and 5) the ideas from IT do not help. The reason is that in the AIT framework, the mutual information of various strings is not exactly zero, but only close to zero within some slack terms. The slack terms are small, but during the rounds of a protocol, the errors can accumulate and grow beyond control (for more detail see the discussion of the limits of the "weak" upper bound int he full version of the paper). To overcome this obstacle we come up with a new type of inequalities for Kolmogorov complexity. These inequalities are substantially different from the classic information inequalities used in the analogous results in IT. This technique is based on ideas similar to the *conditional information inequalities* in [13, 14]. We believe that this technique can be helpful in other cases, including applications in IT (see the discussion in the full version of the paper).

In the proof of a lower bound for communication complexity (Theorem 7), we use methods specific for AIT, with no apparent parallel in IT. We adapt the technique of bounds for the size of common information that goes back to An. Muchnik and use deep results regarding stochastic strings [24, 20, 21], which have not been previously employed in information theory and communication complexity.

6 Final comments

On time-efficient secret key agreement protocols. The secret key agreement protocol in the proof of Theorem 2 is computable but highly non-efficient. The only slow stage is when Bob reconstructs x given his input string y and the fingerprint of x obtained from Alice. At this stage Bob has to simulate all programs of size $C(x \mid y)$ until he obtains a string matching the fingerprint. All other stages of the protocol can be implemented in polynomial time (to this end we need to use an effective version of an extractor in the definition of fingerprints; this increases the overhead in communication complexity from $O(\log n)$ to $\mathrm{poly}(\log n)$, but this is still negligible compared to the size of the fingerprint, which is the dominating term in the communication complexity of the protocol; for details see [28]).

We cannot make Bob's computation effective in general, but we can do it for some specific pairs of inputs (x, y). Actually we can make the entire communication protocol fast, if there is a way to communicate x from Alice to Bob so that (i) communication complexity of this stage (and therefore the information revealed to the adversary) remains about $C(x \mid y)$, and (ii) all computations are performed by Alice and Bob in time $\mathrm{poly}(n)$.

[2] On the high level this protocol consists of three stages: (i) Alice sends to Bob a suitable "fingerprint" of her input $p_1(x)$; (ii) Bob uses y and $p_1(x)$ to recover x; (iii) then both Alice and Bob independently compute another fingerprint $p_2(x)$, which is used as a common secret key. The construction of the fingerprints guarantees that the adversary (who eavesdrops p_1) obtains virtually no information about p_2.

Example 1. (Discussed in Introduction, p. 2.) Let Alice get a random line x in the affine plane over the finite field with 2^n elements, and Bob get a random point y on this line. For *most* inputs of this type we have $C(x \mid y) = n \pm O(\log n)$, and there exists a simple way to transfer x from Alice to Bob with communication complexity n (Alice just sends to Bob the slope of her affine line, and Bob draws a line with this slope incident to his point). Thus, we see once again that for this simple example there exists an effective (polynomial-time) communication protocol to agree on common secret key of size $\approx n$ bits.

Example 2. Let Alice and Bob get n-bits strings x and y respectively, and the Hamming distance between these strings is at most δn for some constant $\delta < 1/2$. For most inputs of this type we have $C(x \mid y) = h(\delta)n \pm O(\log n)$ and $I(x : y) = (1 - h(\delta))n \pm O(\log n)$, where $h(\delta) = \delta \log \frac{1}{\delta} + (1 - \delta) \log \frac{1}{1-\delta}$. Can we transfer x from Alice to Bob with communication complexity $h(\delta)n + o(n)$? It turns out that such a protocol exists; moreover, there exists a communication protocol with asymptotically optimal communication complexity and polynomial time computations, see [25, 10, 11]. Plugging this protocol in our proof of Theorem 3 we conclude that on most pairs of inputs (x, y) of this type Alice and Bob can agree on a common secret key of size $(1 - h(\delta))n - o(n)$, with poly-time computations for both parties.

On using our approach for "one-shot" sources. Most known results for secret key agreement in Shannon's framework are proven under the assumption that the input data available to Alice and Bob is generated by i.i.d. or at least stationary ergodic sources. These results can be derived from Theorem 2 and Theorem 3, using the well-known relation between Shannon entropy and Kolmogorov complexity for the above type of sources [18, 12]. But actually Theorem 2 and Theorem 3 apply in more general settings. We can prove similar bounds for random inputs obtained *in one shot*, without the property of ergodicity.

This is useful because in many natural instances of the secret key agreement problem the input data are far from being ergodic, and therefore the classic technique does not apply. For instance, *Example 1* and *Example 2* discussed above illustrate this situation if we reformulate them in the probabilistic setting (i.e., we introduce the uniform distribution on the set of all valid pairs of inputs). For the probabilistic versions of these examples the matching upper and lower bounds on the size of the common secret key can be easily deduced from Theorem 2 and Theorem 3.

On the error probability. The standard results on secret key agreement deal with the paradigm that the protocol works properly for most randomly chosen inputs (which is typical for the information theory), while in our approach we prove a somewhat stronger statement: for each valid pair of input data the protocol works properly with high probability (which is typical for the theory of communication complexity).

7 Open problems and acknowledgements

▶ **Open Question 1.** *In Theorem 7 we establish a lower bound on how many bits Alice and Bob must communicate to agree on a common secret key. Our proof is valid only for communication protocols with public randomness. Is the same bound true for protocols with private sources of random bits?*

▶ **Open Question 2.** *Our communication protocols are randomized. On the other hand they use unusually few random bits, only $O(\log n)$. It is natural to ask whether we can get rid of external randomness. We conjecture that for $(O(\log n), O(\log n))$-stochastic tuples of inputs the protocol can be made purely deterministic (though it would require very high computational complexity), but this cannot be done in the general case. The proof of this fact likely requires a better understanding of the nature of non-stochastic objects (such as Chaitin's Omega number, [4], see also [24] and [23]).*

References

1 Rudolf Ahlswede and Imre Csiszár. Common randomness in information theory and cryptography - I: secret sharing. *IEEE Trans. Information Theory*, 39(4):1121–1132, 1993. doi:10.1109/18.243431.

2 Luis Antunes, Sophie Laplante, Alexandre Pinto, and Liliana Salvador. Cryptographic security of individual instances. *ICITS*, pages 195–210, 2010.

3 Charles H. Bennett, Gilles Brassard, and Jean-Marc Robert. Privacy amplification by public discussion. *SIAM Journal on Computing*, 17(2):210–229, 1988.

4 Gregory J Chaitin. A theory of program size formally identical to information theory. *Journal of the ACM (JACM)*, 22(3):329–340, 1975.

5 Chung Chan, Ali Al-Bashabsheh, Javad B Ebrahimi, Tarik Kaced, and Tie Liu. Multivariate mutual information inspired by secret-key agreement. *Proceedings of the IEEE*, 3(10):1883–1913, 2015.

6 Alexey V. Chernov, Andrei A. Muchnik, Andrei E. Romashchenko, Alexander Shen, and Nikolai K. Vereshchagin. Upper semi-lattice of binary strings with the relation "x is simple conditional to y". *Theor. Comput. Sci.*, 271(1-2):69–95, 2002. doi:10.1016/S0304-3975(01)00032-9.

7 Imre Csiszár and János Körner. Broadcast channels with confidential messages. *IEEE Trans. Information Theory*, 24(3):339–348, 1978. doi:10.1109/TIT.1978.1055892.

8 Imre Csiszár and Prakash Narayan. Secrecy capacities for multiple terminals. *IEEE Trans. Information Theory*, 50(12):3047–3061, 2004. doi:10.1109/TIT.2004.838380.

9 Peter Gács and János Körner. Common information is far less than mutual information. *Probl. Control Inf. Theory*, 2(2):149–162, 1973.

10 Venkatesan Guruswami and Adam Smith. Codes for computationally simple channels: Explicit constructions with optimal rate. *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 723–732, 2010. doi:10.1109/FOCS.2010.74.

11 Venkatesan Guruswami and Adam Smith. Optimal rate code constructions for computationally simple channels. *Journal of the ACM (JACM)*, 63(4):35, 2016.

12 Yasuichi Horibe. A note on Kolmogorov complexity and entropy. *Applied mathematics letters*, 16(7):1129–1130, 2003.

13 Tarik Kaced and Andrei Romashchenko. Conditional information inequalities for entropic and almost entropic points. *IEEE Transactions on Information Theory*, 59(11):7149–7167, 2013.

14 Tarik Kaced, Andrei Romashchenko, and Nikolay Vereshchagin. Conditional information inequalities and combinatorial applications. *arXiv preprint arXiv:1501.04867*, 2015.

15 Sik Kow Leung-Yan-Cheong. Multi-user and wiretap channels including feedback, July 1976. Tech. Rep. No. 6603-2, Stanford Univ.

16 Ueli M. Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. *Journal of Cryptology*, 5(1):53–66, 1992.

17 Ueli M. Maurer. Secret key agreement by public discussion from common information. *IEEE Trans. Information Theory*, 39(3):733–742, 1993. doi:10.1109/18.256484.

18 Li Ming and Paul M.B. Vitányi. *Kolmogorov complexity and its applications*. Elsevier, 2014.

19 Andrei A. Muchnik. On common information. *Theor. Comput. Sci.*, 207:319–328, 1998.

20 Andrei A. Muchnik and Andrei E. Romashchenko. Stability of properties of Kolmogorov complexity under relativization. *Problems of information transmission*, 46(1):38–61, 2010.

21 Ilya Razenshteyn. Common information revisited. *arXiv preprint arXiv:1104.3207*, 2011.

22 Andrei Romashchenko. Pairs of words with nonmaterializable mutual information. *Problems of Information Transmission*, 36(1):3–20, 2000.

23 Alexander Shen, Vladimir Uspensky, and Nikolay Vereshchagin. *Kolmogorov complexity and algorithmic randomness*. American Mathematical Society, 2017.

24 Alexander Kh. Shen. The concept of (α, β)-stochasticity in the Kolmogorov sense, and its properties. *Soviet Math. Dokl.*, 28(1):295–299, 1983.

25 Adam D. Smith. Scrambling adversarial errors using few random bits, optimal information reconciliation, and better private codes. *Symposium on Discrete Algorithms: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 7(09):395–404, 2007.

26 Himanshu Tyagi. Common information and secret key capacity. *IEEE Transactions on Information Theory*, 59(9):5627–5640, 2013.

27 Aaron D. Wyner. The wire-tap channel. *Bell Syst. Tech J.*, 54(8):1355–1387, 1975.

28 Marius Zimand. Kolmogorov complexity version of Slepian-Wolf coding. In *STOC 2017*, pages 22–32. ACM, June 2017.

Privacy Preserving Clustering with Constraints

Clemens Rösner
Department of Theoretical Computer Science, University of Bonn, Germany
roesner@cs.uni-bonn.de

Melanie Schmidt
Department of Theoretical Computer Science, University of Bonn, Germany
melanieschmidt@uni-bonn.de

Abstract

The k-center problem is a classical combinatorial optimization problem which asks to find k centers such that the maximum distance of any input point in a set P to its assigned center is minimized. The problem allows for elegant 2-approximations. However, the situation becomes significantly more difficult when constraints are added to the problem. We raise the question whether general methods can be derived to turn an approximation algorithm for a clustering problem with some constraints into an approximation algorithm that respects one constraint more. Our constraint of choice is privacy: Here, we are asked to only open a center when at least ℓ clients will be assigned to it. We show how to combine privacy with several other constraints.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Clustering, k-center, Constraints, Privacy, Lower Bounds, Fairness

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.96

Related Version A full version of the paper is available at [33], `https://arxiv.org/abs/1802.02497`.

1 Introduction

Clustering is a fundamental unsupervised learning task: Given a set of objects, partition them into clusters, such that objects in the same cluster are well matched, while different clusters have something that clearly differentiates them. The three classical clustering objectives studied in combinatorial optimization are *k-center*, *k-median* and *facility location*. Given a point set P, k-center and k-median ask for a set of k centers and an assignment of the points in P to the selected centers that minimize an objective. For k-center, the objective is the maximum distance of any point to its assigned center. For k-median, it is the sum of the distances of all points to their assigned center (this is called connection cost). Facility location does not restrict the number of centers. Instead, every center (here called facility) has an opening cost. The goal is to find a set of centers such that the connection cost plus the opening cost of all chosen facilities is minimized. In the unconstrained versions each point will be assigned to its closest center. With the addition of constraints a different assignment is often necessary in order to satisfy the constraints.

A lot of research has been devoted to developing approximation algorithms for these three. The earliest success story is that of k-center: Gonzalez [21] as well as Hochbaum and Shmoys [23] gave a 2-approximation algorithm for the problem, while Hsu and Nemhauser [24] showed that finding a better approximation is NP-hard.

Since then, much effort has been made to approximate the other two objectives. Typically, facility location will be first, and transferring new techniques to k-median poses additional challenges. Significant techniques developed during the cause of many decades are LP rounding

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 96; pp. 96:1–96:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

techniques [11, 34], greedy and primal dual methods [25, 26], local search algorithms [6, 29], and, more recently, the use of pseudo-approximation [32]. The currently best approximation ratio for facility location is 1.488 [31], while the best lower bound is 1.463 [22]. For k-median, the currently best approximation algorithm achieves a ratio of $2.675+\epsilon$ [9], while the best lower bound is $1 + \frac{2}{e} \approx 1.736$ [25].

While the basic approximability of the objectives is well studied, a lot less is known once constraints are added to the picture. Constraints come naturally with many applications of clustering, and since machine learning and unsupervised learning methods become more and more popular, there is an increasing interest in this research topic. It is one of the troubles with approximation algorithms that they are often less easy to adapt to a different scenario than some easy heuristic for the problem, which was easier to understand and implement in the first place. Indeed, it turns out that adding constraints to clustering often requires fundamentally different techniques for the design of approximation algorithms and is a very new challenge altogether.

A good example for this is the *capacity* constraint: Each center c is now equipped with a capacity $u(c)$, and can only serve $u(c)$ points. This natural constraint is notoriously difficult to cope with; indeed, the standard LP formulations for the problems have an unbounded integrality gap. Capacitated k-center was first approximated with uniform upper bounds [8, 28]. Local search provides a way out for facility location, leading to 3- and 5-approximations for uniform [1] and non-uniform capacities [7], and preprocessing together with involved rounding proved sufficient for k-center to obtain a 9-approximation [15, 5]. However, the choice of techniques that turned out to work for capacitated clustering problems is still very limited, and indeed *no* constant factor approximation is known to date for k-median.

And all the while, new constraints for clustering problems are proposed and studied. In *private* clustering [3], we demand a lower bound on the number of points assigned to a center. As stated in [2, 3] this ensures a certain anonymity and is motivated through the need to obtain data privacy. The more general form where each cluster has an individual lower bound is called clustering *with lower bounds* [4]. *Fair* clustering [14] assumes that points have a protected feature (like gender), modeled by a color, and that we want clusters to be fair in the sense that the ratios between points of different colors is the same for every cluster. Clustering *with outliers* [12] assumes that our data contains measurement errors and searches for a solution where a prespecified number of points may be excluded from the cost computation. Other constraints include fault tolerance [27], matroid or knapsack constraints [13], must-link and cannot-link constraints [35], diversity [30] and chromatic clustering constraints [18, 19].

The abundance of constraints and the difficulty to adjust methods for all of them individually asks for ways to *add* a constraint to an approximation algorithm in an oblivious way. Instead of adjusting and reproving known algorithms, we would much rather like to take an algorithm as a black box and ensure that the solution satisfies one more constraint in addition. This is a challenging request. We start the investigation of such add-on algorithms by studying the privacy constraint. Indeed, we develop a method to add the privacy constraint to approximation algorithms for constraint k-center problems. That means that we use an approximation algorithm as a subroutine and ensure that the final solution will additionally respect a given lower bound. The method has to be adjusted depending on the constraint, but it is oblivious to the underlying approximation algorithm used for that constraint.

This works for the basic k-center problem (giving an algorithm for the private k-center problem), but we also show how to use the method when the underlying approximation algorithm is for k-center with outliers, fair k-center and capacitated k-center. We also demonstrate that our method suffices to approximate *strongly private* k-center, where we

assume a protected feature like in fair clustering, but instead of fairness, now demand that a minimum number of points of each color is assigned to each open center to ensure anonymity for each class individually.

Our Technique. The general structure of the algorithm is based on standard thresholding [23], i.e., the algorithm tests all possible thresholds and chooses the smallest for which it finds a feasible solution. For each threshold, it starts with the underlying algorithm and computes a non private solution. Then it builds a suitable network to shift points to satisfy the lower bounds. The approximation ratio of the method depends on the underlying algorithm and on the structure of this network.

The shifting does not necessarily work right away. If it does not produce a feasible solution, then using the max flow min cut theorem, we obtain a set of points for which we can show that the clustering uses too many clusters (and can thus not satisfy the lower bounds). The algorithm then recomputes the solution in this part. Depending on the objective function, we have to overcome different hurdles to ensure that the recomputation works in the sense that it a) makes sufficient progress towards finding a feasible solution and b) does not increase the approximation factor. The process is then iterated until we find a feasible solution.

Results. We obtain the following results for multiple combinations of privacy with other constraints. Note that our definition of k-center (see §2) distinguishes between the set of points P and the set of possible center locations L. This general case is also called the *k-supplier problem*, while classical k-center often assumes that $P = L$. Our reductions can handle the general case (with a slight increase in approximation ratio); whether the resulting algorithm is then for k-center or k-supplier thus depends on the evoked underlying algorithm.

- We obtain a 4-approximation for private k-center with outliers (5 for the supplier version). This matches the best known bounds [3] ([4] for the supplier version (this also holds for non-uniform lower bounds)).
- We compute an 11-approximation for private capacitated k-center (i.e., centers have a lower bound *and* an upper bound), and a 8-approximation for private uniform capacitated k-center (where the upper bounds are uniform, as well). The best known bounds for these two problems are 9 and 6 [17]. For the supplier version we obtain a 13-approximation which matches the best known bound [17] (for uniform upper bounds a 9-approximation-algorithm is known [17]).
- We achieve constant factor approximations for private capacitated/uncapacitated and fair k-center/k-supplier clustering. The approximation factor depends on the *balance* of the input point set and the type of upper bounds, it ranges between 10 in the uncapacitated case where for each color c the number of points with color c is an integer multiple of the number of points with the rarest color and 325 in the general supplier version with non-uniform upper bounds. To the best of our knowledge, all these combinations have not been studied before.
- Along the way, we propose constant factor algorithms for general cases of fair clustering. While [14] introduces a pretty general model of fairness, it only derives approximation algorithms for inputs with two colors and a balance of $1/t$ for an integer t. We achieve ratios of 14 and 15 for the general fair k-center and supplier problem, respectively.
- Finally, we propose the *strongly private k-center problem*. As in the fair clustering problem, the input here has a protected feature like gender, modeled by colors. Now instead of a fair clustering, we aim for anonymity for each color, meaning that we have a lower bound for each color. Each open center needs to be assigned this minimum number

Table 1 An overview on the approximation results that we combine with privacy.

	Vanilla	Capacities		Outlier	Fair Subset Partition	
		uniform	non-uniform		$\frac{r}{b} \in \mathbb{N}$	general
k-center	2 [23]	6 [28]	9 [5]	2 [10]	2 [14]	12 (full version Thm.22 [33])
k-supplier	3 [23]		11 [5]	3 [12]		

of points for each color. To the best of our knowledge, this problem has not been studied before; we obtain a 4-approximation as well as a 5-approximation for the supplier version.

Since our method does not require knowledge of the underlying approximation algorithm, the approximation guarantees improve if better approximation algorithms for the underlying problems are found. There is also hope that our method could be used for new, not yet studied constraints, with not too much adjustment.

Related Work. The unconstrained k-center problem can be 2-approximated [21, 23], and it is NP-hard to approximate it better [24]. The k-supplier problem can be 3-approximated [23], and this is also tight.

Capacitated k-center was first approximated with uniform upper bounds [8, 28]. Two decades after the first algorithms for the uniform case, [15] provided the first constant factor approximation for non-uniform capacities. The algorithm was improved and applied to the k-supplier problem in [5]. In contrast to capacities, *lower* bounds are less studied. The private k-center problem is introduced and 2-approximated in [3], and non-uniform lower bounds are studied in [4]. The k-center/k-supplier problem with outliers is 3-approximated in [12] alongside approximations to other robust variants of the k-center problem. The approximation factor for the k-center problem with outliers was improved to 2 in [10].

The fair k-center problem was introduced in [14]. The paper describes how to approximate the problem by using an approximation for a subproblem that we call fair subset partition problem. Algorithms for this subproblem are derived for two special cases where the number of colors is two, and the points are either perfectly balanced or the number of points of one color is an integer multiple of the number of points of the other color.

These are the constraints for which we make use of known results. We state the best known bounds and their references in Table 1. Approximation algorithms are also e.g. known for fault tolerant k-center [27] and k-center with matroid or knapsack constraints [13].

Relatively little is known about the combination of constraints. Cygan and Kociumaka [16] give a 25-approximation for the capacitated k-center problem with outliers. Aggarwal et. al [3] give a 4-approximation for the private k-center problem with outliers. Ahmadian and Swamy [4] consider the combination of k-supplier with outliers with (non-uniform) lower bounds and derive a 5-approximation. The paper also studies the k-supplier problem with outliers (without lower bounds), and the min-sum-of-radii problem with lower bounds and outliers. Their algorithms are based on the Lagrangian multiplier preserving primal dual method due to Jain and Vazirani [26].

Ding et. al [17] study the combination of capacities and lower bounds as well as capacities, lower bounds and outliers by generalizing the LP algorithms from [5] and [16] to handle lower bounds. They give results for several variations, including a 6-approximation for private capacitated k-center and a 9-approximation for private capacitated k-supplier.

Friggstad, Rezapour, Salavatipour [20] consider the combination of uniform capacities and non-uniform lower bounds for *facility location* and obtain bicriteria approximations.

Outline. In §2, we introduce necessary notation. §3 then presents our method, applied to the private k-center problem with outliers. We choose the outlier version since it is non-trivial but still intuitive and does thus give a good impression on the application of our method. In §4, we then briefly explain how to adjust the method to approximate private and fair k-center, private and capacitated k-center, k-center with all three constraints as well as the strongly private k-center problem. §5 provides a conclusion.

2 Preliminaries

Let (X, d) be a finite metric space, i.e., X is a finite set and $d : X \times X \to \mathbb{R}_{\geq 0}$ is a metric. We use $d(x, T) = \min_{y \in T} d(x, y)$ for the smallest distance between $x \in X$ and a set $T \subseteq X$. For two sets $S, T \subseteq X$, we use $d(S, T) = \min_{x \in S, y \in T} d(x, y)$ for the smallest distance between any pair $x \in S, y \in T$.

Let $P \subseteq X$ be a subset of X called *points* and let $L \subseteq X$ be a subset of X called *locations*. An instance of a private *assignment constrained* k-center problem consists of P, L, an integer $k \in \mathbb{N}$, a lower bound $\ell \in \mathbb{N}$ and possibly more parameters. Given the input, the problem is to compute a set of centers $C \subseteq L$ with $|C| \leq k$ and an *assignment* $\phi : P \to C$ of the points to the selected centers that satisfies $\ell \leq |\phi^{-1}(c)|$ for every selected center $c \in C$, and some specific *assignment restriction*. The solution C, ϕ shall be chosen such that

$$\max_{x \in P} d(x, \phi(x))$$

is minimized. Different assignment restrictions lead to different constrained private k-center problems. The *capacity* assignment restriction comes with an upper bound function $u : L \to \mathbb{N}$ for which we require $\ell \leq u(x)$ for all $x \in L$, and then demands $|\phi^{-1}(c)| \leq u(c)$. When we have $u(x) = u$ for all $x \in L$ and some $u \in \mathbb{N}$, then we say that the capacities are *uniform*, otherwise, we say they are *non-uniform*. The *fairness* assignment restriction provides a mapping $\chi : P \to Col$ of points to colors and then requires that each cluster has the same ratio between the numbers of points with different colors (see §4.2 in the full version [33] for specifics). The *strongly private k-center problem* can also be cast as a k-center problem with an assignment restriction. Again, the input now additionally contains a mapping χ of points to colors. Now the assignment is restricted to ensure that it satisfies the lower bound for the points of each color. We even consider the slight generalization where each color has its own lower bound, and call this problem the strongly private k-center problem.

An instance of the *private k-center problem with outliers* consists of P, L, an integer $k \in \mathbb{N}$, a lower bound ℓ, and a parameter o for the maximum number of outliers. The problem is to compute a set of centers $C \subseteq L$ with $|C| \leq k$ and an assignment $\phi : P \to C \cup \{out\}$, with $|\phi^{-1}(out)| \leq o$, which assigns each point to a center in C or to be an outlier. The choice of C, ϕ shall minimize

$$\max_{x \in P \setminus \phi^{-1}(out)} d(x, \phi(x)).$$

3 Private k-center with Outliers

▶ **Theorem 1.** *Assume that there exists an approximation algorithm A for the k-center problem with outliers with approximation factor α.*

Then for instances P, L, k, ℓ, o of the private k-center problem with outliers, we can compute an $(\alpha + 2)$-approximation in polynomial time.

Proof. Below, we describe an algorithm that uses a threshold graph with threshold τ. We show that for any given $\tau \in \mathbb{R}$, the algorithm has polynomial runtime and, if τ is equal to opt, the value of the optimal solution, computes an $(\alpha + 2)$-approximation. Since we know that the value of every solution is equal to the distance between a point and a location, we test all $O(|P||L|)$ possible distances for τ and return the best feasible clustering returned by any of them. The main proof is the proof of Lemma 2 below, which concludes this proof. ◀

Let us start with a general idea on how the algorithm works for a fixed τ. If $\tau < \mathsf{opt}$, then the algorithm will detect this, so assume that $\tau \geq \mathsf{opt}$ for now. We first use the approximation algorithm to obtain a clustering which does not necessarily satisfy the privacy constraint. Then we try to reassign points to establish the lower bounds. A point p may be reassigned to any cluster which contains at least one point which is within distance 2τ of p. (Note that any point in p's optimum cluster is at distance $\leq 2\tau$). If it is possible to reassign points like that and obtain a feasible clustering, we can compute such a reassignment with a maximum flow algorithm. Otherwise we find a set of points $P(V')$ for which we can show that the optimal clustering uses less clusters to cover all of them. We add all outliers to $P(V')$ to account for the fact that $P(V')$ may already contain outliers of the optimum solution. We then use the approximation algorithm on $P(V')$ to compute a new clustering with outliers. The output will contain fewer clusters or the same number of clusters and fewer outliers. We repeat the process until we find a feasible solution.

▶ **Lemma 2.** *Assume that there exists an approximation algorithm A for the k-center problem with outliers with approximation factor α. Let P, L, k, ℓ, o be an instance of the private k-center problem with outliers, let $\tau > 0$ and let opt denote the maximum radius in the optimal feasible clustering for P, L, k, ℓ, o. We can in polynomial time compute a feasible clustering with a maximum radius of at most $(\alpha + 2)\tau$ or determine $\tau < \mathsf{opt}$.*

Proof. We first use A to compute a solution without the lower bound. Let $\mathcal{C} = (C, \phi)$ be an α-approximate solution for the k-center problem with outliers on P, L, k, o. Note that $\mathcal{C}$ may contain clusters with less than ℓ points. Let $k' = |C|$ (note that $k' < k$ is possible), $C = \{c_1, \ldots, c_{k'}\}$, and let $C_1, \ldots, C_{k'}$ be the clusters that $\mathcal{C}$ induces, i.e., $C_j := \phi^{-1}(c_j)$. Finally, let $r = \max_{x \in P} d(x, \phi(x))$ be the largest distance of any point to its center. Observe that an optimal solution to the k-center problem with outliers can only have a lower objective value than the optimal solution to our problem because we only dropped a condition. Therefore, $\tau \geq \mathsf{opt}$ implies that $r \leq \alpha \cdot \mathsf{opt} \leq \alpha \cdot \tau$. If we have $r > \alpha \cdot \tau$, we return $\tau < \mathsf{opt}$.

We use $\mathcal{C}$ and τ to create a threshold graph which we use to either reassign points between the clusters to obtain a feasible solution or to find a set of points P' for which we can show that every feasible clustering with maximum radius τ uses less clusters than our current solution to cover it. In the latter case we compute another α-approximate solution which uses fewer clusters on P' and repeat the process. Note that for $\tau < \mathsf{opt}$ such a clustering does not necessarily exist, but for $\tau \geq \mathsf{opt}$ the optimal clustering provides a solution for P' with fewer clusters. If we do not find such a clustering with maximum radius at most $\alpha \cdot \tau$, we return $\tau < \mathsf{opt}$.

We show that every iteration of the process reduces the number of clusters or the number of outliers, therefore the process stops after at most $k \cdot o$ iterations. It may happen that our final solution contains much less clusters than the optimal solution (but it will be an approximate solution for the optimal solution with k centers).

We will use a network flow computation to move points from clusters with more than ℓ points to clusters with less than ℓ points. Moving a point to another cluster can increase the radius of the cluster. We only want to move points between clusters such that the radius

does not increase by too much. More precisely, we only allow a point p to be moved to another cluster C_i if the distance $d(p, C_i)$ between the point and the clusters is at most 2τ. This is ensured by the structure of the network described in the next paragraph. Unless stated otherwise, when we refer to distances between a point and a cluster in the following, we mean the distance between the point and the cluster in its original state before any points have been reassigned.

Given $\mathcal{C}$ and τ, we create the threshold graph $G_\tau = (V_\tau, E_\tau)$ as follows. V_τ consists of a source s, a sink t, a node v_i for each cluster C_i, a node v_{out} for the set of outliers and a node w_p for each point $p \in P$. For all $i \in [k']$, we connect s to v_i if the cluster C_i contains more than ℓ points and set the capacity of (s, v_i) to $|C_i| - \ell$. If the cluster C_i contains fewer than ℓ points, we connect v_i with t and set the capacity of (v_i, t) to $\ell - |C_i|$. Furthermore, we connect v_i with w_p for all $p \in C_i$ and set the capacity of (v_i, w_p) to 1. We also connect s to v_{out} with capacity o and v_{out} with w_p for all $p \in \phi^{-1}(out)$ with capacity 1. Whenever a point p and a cluster C_i with $p \notin C_i$ satisfy $d(p, C_i) \leq 2\tau$ (i.e., there is a point $q \in C_i$ that satisfies $d(p, q) \leq 2\tau$), we connect w_p with v_i with capacity 1.

Formally the graph $G_\tau = (V_\tau, E_\tau)$ is defined by

$$V_\tau = \{v_{out}\} \cup \{v_i \mid 1 \leq i \leq k'\} \cup \{w_p \mid p \in P\} \cup \{s, t\} \text{ and} \tag{1}$$

$$E_\tau = \{(v_i, w_p) \mid p \in C_i\} \cup \{(w_p, v_i) \mid p \notin C_i \wedge d(p, C_i) \leq 2\tau\} \tag{2}$$

$$\cup \{(v_{out}, w_p) \mid \phi(p) = out\} \tag{3}$$

$$\cup \{(s, v_{out})\} \cup \{(s, v_i) \mid |C_i| - \ell > 0\} \cup \{(v_i, t) \mid |C_i| - \ell < 0\}. \tag{4}$$

We define the capacity function $cap : E_\tau \to \mathbb{R}$ by

$$cap(e) = \begin{cases} \ell - |C_i|, & \text{if } e = (v_i, t) \\ |C_i| - \ell, & \text{if } e = (s, v_i) \\ o, & \text{if } e = (s, v_{out}) \\ 1 & \text{otherwise.} \end{cases} \tag{5}$$

We use $G = (V, E)$ to refer to G_τ as τ is clear from context. We now compute an integral maximum s-t-flow f on G. According to f we can reassign points different clusters.

▶ **Lemma 3.** *Let f be an integral maximal s-t-flow on G. It is possible to reassign p to C_i for all edges (w_p, v_i) with $f((w_p, v_i)) = 1$.*

The resulting solution has a maximum radius of at most $r + 2\tau$. If f saturates all edges of the form (v_i, t), then the solution is feasible.

Proof. Let $p \in C_i$. The choice of capacity 1 on (v_i, w_p) and flow conservation ensure $\sum_{(w_p, v_j) \in E} f((w_p, v_j)) \leq 1$ for p. Therefore no point would have to be reassigned to more than one cluster. Note that for every point $p \in C_i$ that would be reassigned we must have $f((v_i, w_p)) = 1$ and for every edge (v_i, w_p) with $f((v_i, w_p)) = 1$ the point p would be reassigned.

For any $1 \leq j \leq k'$, let $p \in C_i$ be any point which we want to reassign to C_j. Then we must have $(w_p, v_j) \in E$ and therefore there must be a point $q \in C_j$ with $d(p, q) \leq 2\tau$. Thus we have

$$d(p, c_j) \leq d(p, q) + d(q, c_j) \leq 2\tau + r = r + 2\tau.$$

Now assume that f saturates all edges of the form (v_i, t) and let $1 \leq i \leq k'$. If E contains the edge (v_i, t), then it can not contain the edge (s, v_i) and therefore all incoming edges of v_i

are of the form (w_p, v_i). Flow conservation then implies that the number of points reassigned to C_i minus the points reassigned away from C_i is equal to $f((v_i, t))$, which increases the number of points in C_i to ℓ.

If E contains the edge (s, v_i), then it can not contain the edge (v_i, t) and therefore all outgoing edges of v_i are of the form (v_i, w_p). Flow conservation then implies that the number of points reassigned away from C_i minus the points reassigned to C_i is equal to $f((s, v_i))$, which reduces the number of points in C_i to at least ℓ.

If E contains neither (s, v_i) nor (v_i, t), then the number of points in C_i is equal to ℓ and does not change (the points may change, but their number does not).

In all three cases C_i contains at least ℓ points after the reassignment. ◀

If f saturates all edges of the form (v_i, t) in G, then we reassign points according to Lemma 3 and return the new clustering.

Otherwise we look at the residual network G_f of f on G. Let V' be the set of nodes in G_f which can not be reached from s. V' contains all nodes representing clusters which would not satisfy the privacy constraint after the reassignment. We say cluster C_i *belongs* to V' if $v_i \in V'$, and a point $p \in C_i$ is *adjacent* to V' if $w_p \in V'$ and $v_i \notin V'$. Let $C(V')$ denote the set of clusters belonging to V'. Let $k'' = |C(V')|$. We say a point p belongs to V' if the cluster C_i with $p \in C_i$ belongs to V'. Let $P(V')$ and $P_A(V')$ denote the set of points that belong to V' and the set of points adjacent to V'.

▶ **Lemma 4.** *Any clustering on P with maximum radius at most τ that contains at least ℓ points in every cluster uses fewer than k'' clusters to cover all points in $P(V')$.*

Proof. We first observe that V' must have the following properties:

- $v_i \in V'$ and $(w_p, v_i) \in E$ implies $w_p \in V'$.
- $w_p \in V'$, $(w_p, v_i) \in E$ and $f((w_p, v_i)) > 0$ implies $v_i \in V'$.
- $w_p \in V'$ for some $p \in C_i$ and $v_i \notin V'$ implies $f((v_i, w_p)) = 1$.

The first property follows from the fact that f can only saturate (w_p, v_i) if f also saturates (v_j, w_p) for $p \in C_j$. So, either (w_p, v_i) is not saturated, which means that v_i can be reached from any vertex that reaches w_p, or (w_p, v_i) *is* saturated, which means that the only incoming edge of w_p in G_f is (v_i, w_p). In both cases, if $v_i \in V'$, then $w_p \in V'$. The second property follows since $f((w_p, v_i)) > 0$ implies $(v_i, w_p) \in E(G_f)$. The third property is true since we defined $cap((v_i, w_p)) = 1$.

This implies that a reassignment due to Lemma 3 would reassign all points adjacent to V' to clusters in $C(V')$ and moreover all reassignments from points in $P(V') \cup P_A(V')$ would be to clusters in $C(V')$. Let n_i denote the number of points that would be assigned to C_i after the reassignment. Then $|P(V')| + |P_A(V')| = \sum_{C_i \in C(V')} n_i$.

Now we argue that this sum is smaller than $k'' \cdot \ell$ by observing that each $n_i \leq \ell$ and at least one n_i is strictly smaller than ℓ.

Let C_i be a cluster with more than ℓ points after the reassignment. Then (s, v_i) is not saturated by f and v_i can be reached from s in G_f. Therefore after the reassignment no cluster $C_i \in C(V')$ would contain more than ℓ points; in other words, $n_i > \ell$ implies $C_i \notin C(V')$.

Let C_i be a cluster which would still contain fewer than ℓ points after the reassignment. This implies that f does not saturate the edge (v_i, t). Therefore t can be reached from v_i and since f is a maximum s-t flow, v_i can not be reached from s. We must have $v_i \in V'$.

Because we assumed that the reassignment does not satisfy all lower bounds, at least one such cluster has to exist. This implies

$$|P(V')| + |P_A(V')| = \sum_{C_i \in C(V')} n_i < k'' \cdot \ell.$$

Which means that the clusters in $C(V')$ and $P_A(V')$ do not contain enough points to satisfy the lower bound in k'' clusters.

By definition of G and V', for two points p, q with $p \in P(V')$ and $d(p,q) \leq 2\tau$ we must have $q \in P(V') \cup P_A(V')$. Let $\mathcal{C}'$ be a clustering that abides the lower bounds and has a maximal radius of at most τ. Then every cluster C' in $\mathcal{C}'$ that contains at least one point from $P(V')$ can only contain points from $P(V') \cup P_A(V')$. Therefore $\mathcal{C}'$ must contain fewer than k'' clusters which contain at least one point from $P(V')$. ◀

If we have $\tau \geq \mathsf{opt}$, then Lemma 4 implies that the optimal solution covers all points in $P(V')$ with fewer than k'' clusters. An α-approximative solution on the point set $P(V')$ with at most $k'' - 1$ clusters which contains at most o outliers is then α-approximative for $P(V')$.

Unfortunately, we do not know how many outliers an optimal clustering has in $P(V')$. We therefore involve the outliers $\phi^{-1}(out)$ in our new computation as well. Let $o' = |\phi^{-1}(out)|$ denote the current number of outliers. We obtain the following Lemma through a counting argument.

▶ **Lemma 5.** *We call a cluster special if it contains at least one point from $P(V')$ or only contains points from $\phi^{-1}(out)$. Let $\mathcal{C}'$ be a clustering on P with a maximum radius of at most τ on all special clusters that respects the lower bounds, has at most o outliers and consists of at most k clusters out of which at most k'' are special. If $\mathcal{C}'$ has exactly k'' special clusters, then $\mathcal{C}'$ has at most $o' - 1$ outliers in $P(V') \cup \phi^{-1}(out)$.*

Proof. Assume the clustering contains exactly special k'' clusters. Each of these clusters has to contain at least ℓ points from $P(V') \cup P_A(V') \cup \phi^{-1}(out)$. We know

$$|P(V') \cup P_A(V') \cup \phi^{-1}(out)| \leq |P(V') \cup P_A(V')| + o' < k''\ell + o'.$$

So there remain at most $o' - 1$ unclustered points in $P(V') \cup \phi^{-1}(out)$. ◀

Now we need to show that such a clustering exists if $\tau \geq opt$ is the case.

▶ **Lemma 6.** *If $\tau \geq opt$, then there exists a clustering $\mathcal{C}'$ on P with a maximum radius at most τ on all special clusters that respects the lower bounds, has at most o outliers and consists of at most k clusters out of which at most k'' are special.*

Proof. We look at an optimal clustering $\mathcal{C}_{opt}$. The only way $\mathcal{C}_{opt}$ can violate a condition is if it contains $k''' > k''$ special clusters. Lemma 4 implies that $\mathcal{C}_{opt}$ contains at least $k''' - k''$ clusters that contain only points in $\phi^{-1}(out)$. If all clusters in $\mathcal{C}_{opt}$ are special we know $P = P_A(V') \cup P(V') \cup \phi^{-1}(out)$. We arbitrarily select $k''' - k''$ clusters from $\mathcal{C}_{opt}$ that contain only points in $\phi^{-1}(out)$, declaring all points in them as outliers and closing the corresponding centers. This leaves us with k'' clusters which contain at least $k'' \cdot \ell$ points. Since $P = P_A(V') \cup P(V') \cup \phi^{-1}(out)$ this leaves at most $o' - 1$ outliers. Otherwise, if $\mathcal{C}_{opt}$ contains at least one cluster C which is not special, we add all outliers from $P \setminus (P_A(V') \cup P(V') \cup \phi^{-1}(out)$ to C. Again we arbitrarily select $k''' - k''$ clusters from $\mathcal{C}_{opt}$ that contain only points in $\phi^{-1}(out)$, declaring all points in them as outliers and closing the corresponding centers. By creation there are no unclustered points in $P \setminus (P_A(V') \cup P(V') \cup \phi^{-1}(out)$ and exactly k'' special clusters with radius at most τ. Therefore this clustering contains at most $o' - 1$ outliers and has at most k clusters. ◀

We now use A again to compute new solutions without the lower bound: Let $\mathcal{C}'_1 = (C'_1, \phi'_1)$ be an α-approximate solution for the k-center problem with outliers on $P(V') \cup \phi^{-1}(out)$, L, $k'' - 1$, o and let $\mathcal{C}'_2 = (C'_2, \phi'_2)$ be an α-approximate solution for the k-center problem with outliers on $P(V') \cup \phi^{-1}(out)$, L, k'', $o' - 1$. Let $r'_i = \max_{x \in P(V') \cup \phi^{-1}(out)} d(x, \phi'_i(x))$.

Note that in case $\tau < \mathsf{opt}$, it can happen that no such clustering exists or that we obtain $r'_i > \alpha \cdot \tau$ for both $i = 1$ and $i = 2$. We then return $\tau < \mathsf{opt}$. Otherwise for at least one $i \in \{1, 2\}$ $\mathcal{C}'_i$ must exist together with $r'_i \leq \alpha \cdot \tau$.

If $\mathcal{C}'_2$ exists and we have $r'_2 \leq \alpha \cdot \tau$ we replace $C(V')$ by C'_2 in $\mathcal{C}$ and adjust ϕ accordingly to obtain $\mathcal{C}_1 = (C_1, \phi_1)$ with $C_1 = (C \setminus C(V')) \cup C'_2$ and

$$\phi_1(p) = \begin{cases} \phi'_2(p) & \text{if } p \in P(V') \cup \phi^{-1}(out) \\ \phi(p) & \text{otherwise.} \end{cases} \tag{6}$$

Otherwise, if $\mathcal{C}'_1$ exists, we have $r'_1 \leq \alpha \cdot \tau$ and either $\mathcal{C}'_2$ does not exist or we have $r'_2 > \alpha \cdot \tau$, we analogous replace $C(V')$ by C'_1 to obtain $\mathcal{C}_1$.

▶ **Lemma 7.** *If we did not return $\tau < \mathsf{opt}$, then $\mathcal{C}_1$ is a solution for the k-center problem with outliers on P, L, k, o and we have $r_1 = \max_{x \in P} d(x, \phi_1(x)) \leq \alpha \cdot \tau$.*

Proof. $\mathcal{C}$ is a solution for the k-center problem with outlier on P, L, k, o with $r < \alpha \cdot \tau$ and since we did not return $\tau < \mathsf{opt}$, we must have $r'_i \leq \alpha\tau$ for the chosen $i \in \{1, 2\}$. ◀

We iterate the previous process with the new clustering $\mathcal{C}_1$ until we either determine $\tau < \mathsf{opt}$ or the reassignment of points according to Lemma 3 yields a feasible solution. Since each iteration reduces the number of clusters or keeps the same number of clusters and reduces the number of outliers, the process terminates after at most $k \cdot o$ iterations. ◀

▶ **Corollary 8.** *We can compute a 4-approximation for instances of the private k-center problem with outliers and a 5-approximation for instances of the private k-supplier problem in polynomial time.*

Proof. Follows from Theorem 1 together with the 2-approximation for k-center with outliers in [10] and the 3-approximation for k-supplier with outliers in [12]. ◀

4 Combining Privacy with other Constraints

In this section, we take the general idea from §3 and instead of outliers use it to combine privacy with other restrictions on the clusters. Given a specific restriction $\mathcal{R}$ and an approximation algorithm A for the k-center problem with restriction $\mathcal{R}$ with approximation factor α we ask: Can we similar to §3 extend A to compute an $O(\alpha)$-approximation for the private k-center problem with restriction $\mathcal{R}$?

In §3, we made use of two properties of a clustering with outliers. In Lemma 3 we used that reassigning points to another cluster never increases the number of outliers and in Lemma 4 we used that outliers have the somewhat local property that computing a new clustering on the points V' from a subset of the clusters together with the set of outliers can not create more outliers on the remaining points.

We use this section to briefly explain how the proofs with other restriction properties differ from §3 and state our results for the general cases (for the complete proofs as well as the approximation factors for all different cases, see §4+§5 in the full version [33]).

As new restrictions we study capacities, fairness and multiple lower bounds for different types of points. For the private capacitated k-center problem, the proof is easier than for

private k-center with outliers. The threshold graph $G_\tau = (V_\tau, E_\tau)$ does not need to contain nodes for the outliers anymore. It is defined by

$$V_\tau = \{v_i \mid 1 \leq i \leq k'\} \cup \{w_p \mid p \in P\} \cup \{s, t\} \text{ and} \tag{7}$$

$$E_\tau = \{(v_i, w_p) \mid p \in C_i\} \cup \{(w_p, v_i) \mid p \notin C_i \wedge d(p, C_i) \leq 2\tau\} \tag{8}$$

$$\cup \{(s, v_i) \mid |C_i| - \ell > 0\} \cup \{(v_i, t) \mid |C_i| - \ell < 0\}, \tag{9}$$

together with the capacity function $cap : E_\tau \to \mathbb{R}$

$$cap(e) = \begin{cases} \ell - |C_i|, & \text{if } e = (v_i, t) \\ |C_i| - \ell, & \text{if } e = (s, v_i) \\ 1 & \text{otherwise.} \end{cases} \tag{10}$$

The capacities on the clusters do not influence the threshold graph; they are handled by the underlying approximation algorithm. Since we never increase the number of points of a cluster to more than ℓ, our method will not introduce any capacity violation. We obtain the following results.

▶ **Theorem 9.** *Assume that there exists an approximation algorithm A for the capacitated k-center problem with approximation factor α. Then we can compute an $(\alpha + 2)$-approximation for the private capacitated k-center problem in polynomial time.*

In order to combine privacy with fairness, we first develop an approximation algorithm for general cases of the fair k-center problem. [14] show how to solve fair k-center based on approximating a subproblem that we call *fair subset partitioning problem* (it's called fairlet decomposition in [14]), and which consists of partitioning the input into fair subsets such that the maximum diameter of any subset is minimal. They give an exact algorithm for two colors with perfect balance and a 2-approximation for two colors with balance $1/t$ for an integer t. We propose an algorithm for multiple colors and general balance values. It separates the points into the different colors, computes a capacitated clustering on the points for one of the colors and uses a matching algorithm on threshold graphs in order to add the points of the other colors.

▶ **Theorem 10.** *A 12-approximation for the fair subset partition problem can be computed in polynomial time. If $b_c = 1$ for at least one color $c \in Col$, then a 2-approximation for the fair subset partition problem can be computed in polynomial time (even if $|Col| > 2$).*

With that we obtain the first approximation algorithm for the general fair k-center problem. For the private and fair k-center problem we then use a partitioning into fair subsets and let these be the nodes of the threshold graph instead of single points. When establishing the lower bounds, we move fair subsets as a whole. Let $F = \{F_1, \ldots\}$ denote the fair subsets. Then we define $G_\tau = (V_\tau, E_\tau)$ by

$$V_\tau = \{v_{out}\} \cup \{v_i \mid 1 \leq i \leq k'\} \cup \{f_i \mid F_i \in F\} \cup \{s, t\} \text{ and} \tag{11}$$

$$E_\tau = \{(v_i, f_j) \mid F_j \subseteq C_i\} \cup \{(f_j, v_i) \mid F_j \cap C_i = \emptyset \wedge d(C_i, F_j) \leq 2\tau\} \tag{12}$$

$$\cup \{(s, v_i) \mid |C_i| - \ell > 0\} \cup \{(v_i, t) \mid |C_i| - \ell < 0\} \tag{13}$$

with the capacity function $cap : E_\tau \to \mathbb{R}$

$$cap(e) = \begin{cases} \left\lceil \frac{\ell - |C_i|}{b} \right\rceil, & \text{if } e = (v_i, t) \\ \left\lceil \frac{|C_i| - \ell}{b} \right\rceil, & \text{if } e = (s, v_i) \\ 1 & \text{otherwise.} \end{cases} \tag{14}$$

We obtain the following results.

▶ **Theorem 11.** *Assume that there exists an approximation algorithm A for the fair subset partition problem with approximation factor α. Then we can compute a $(3\alpha+4)/(3\alpha+5)$-approximation for the private fair k-center/supplier problem in polynomial time.*

We also adjust our method to approximate the private fair and capacitated k-center problem, and combine it with our results on approximating the fair subset partitioning problem. The approximation ratio now becomes $\alpha(2\beta+1)$, where α is the approximation ratio for the capacitated k-center problem, and β is the approximation ratio for the fair subset partitioning problem (see Lemma 29 in the full version [33]).

For the strongly private k-center problem (see §5 in the full version [33]) we create a separate threshold graph for each color and separately try to satisfy the lower bound for each color. For a color $i \in Col$ we let P^i denote the points in P with color i and let C_j^i denote the points in C_j with color i. We create the threshold graph $G_{\tau,i} = (V_{\tau,i}, E_{\tau,i})$ by

$$V_{\tau,i} = \{v_j \mid 1 \le j \le k'\} \cup \{w_p \mid p \in P^i\} \cup \{s,t\} \text{ and} \tag{15}$$

$$E_{\tau,i} = \{(v_j, w_p) \mid p \in C_j^i\} \cup \{(w_p, v_j) \mid p \in P^i \setminus C_j \wedge d(p, C_j) \le 2\tau\} \tag{16}$$

$$\cup \{(s, v_j) \mid |C_j \cap \chi^{-1}(i)| - \ell_i > 0\} \cup \{(v_j, t) \mid |C_j \cap \chi^{-1}(i)| - \ell_i < 0\}. \tag{17}$$

We define the capacity functions $cap_i : E_{\tau,i} \to \mathbb{R}$ by

$$cap(e) = \begin{cases} \ell_i - |C_j \cap \chi^{-1}(i)|, & \text{if } e = (v_j, t) \\ |C_j \cap \chi^{-1}(i)| - \ell_i, & \text{if } e = (s, v_j) \\ 1 & \text{otherwise.} \end{cases} \tag{18}$$

We obtain the following result.

▶ **Theorem 12.** *Assume that there exists an approximation algorithm A for the k-center problem with approximation factor α. Then we can compute an $(\alpha+2)$-approximation for the strongly private k-center problem in polynomial time.*

The following corollary summarizes the remaining results in the full version [33] for the different constrained private k-center problems, and fair k-center clustering.

▶ **Corollary 13.** *There are polynomial approximation algorithms that compute*

- *an 11/13-approximation for the private capacitated k-center/supplier problem (Cor. 14 + 15),*
- *a 14/15-approximation for the fair k-center/supplier problem (Cor. 23),*
- *a 40/41-approximation for the private and fair k-center/supplier problem (Cor. 25),*
- *a 225/325-approximation for the private fair capacitated k-center/supplier problem (C. 30),*
- *improved approximations for the last three problems in special cases, and*
- *a 4/5-approximation for the strong private k-center/supplier problem (Cor. 36).*

5 Conclusion and open questions

We studied k-center with capacities, fairness and outliers and have coupled these constraints with privacy, and we proposed strongly private k-center. In addition to improving the approximation guarantee of the presented coupling process, open questions include extending it to arbitrary lower bounds, and to different objective functions. In Appendix A of the full

version [33], we present a bicriteria result for facility location that violates the lower bounds. For k-median, a similar procedure is not known at all so far. Finally, the question how to obliviously add other constraints than privacy is completely open, too.

References

1 Ankit Aggarwal, Anand Louis, Manisha Bansal, Naveen Garg, Neelima Gupta, Shubham Gupta, and Surabhi Jain. A 3-approximation algorithm for the facility location problem with uniform capacities. *Mathematical Programming*, 141(1-2):527–547, 2013.

2 Gagan Aggarwal, Tomas Feder, Krishnaram Kenthapadi, Rajeev Motwani, Rina Panigrahy, Dilys Thomas, and An Zhu. Approximation algorithms for k-anonymity. In *Proceedings of the International Conference on Database Theory (ICDT 2005)*, November 2005. URL: `http://ilpubs.stanford.edu:8090/645/`.

3 Gagan Aggarwal, Rina Panigrahy, Tomás Feder, Dilys Thomas, Krishnaram Kenthapadi, Samir Khuller, and An Zhu. Achieving anonymity via clustering. *ACM Transaction on Algorithms*, 6(3):49:1–49:19, 2010.

4 Sara Ahmadian and Chaitanya Swamy. Approximation algorithms for clustering problems with lower bounds and outliers. In *43rd International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 69:1–69:15, 2016.

5 Hyung-Chan An, Aditya Bhaskara, Chandra Chekuri, Shalmoli Gupta, Vivek Madan, and Ola Svensson. Centrality of trees for capacitated k-center. *Mathematical Programming*, 154(1-2):29–53, 2015.

6 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

7 Manisha Bansal, Naveen Garg, and Neelima Gupta. A 5-approximation for capacitated facility location. In *20th Annual European Symposium on Algorithms (ESA)*, pages 133–144, 2012.

8 Judit Bar-Ilan, Guy Kortsarz, and David Peleg. How to allocate network centers. *Journal of Algorithms*, 15(3):385–415, 1993.

9 Jaroslaw Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Transactions on Algorithms*, 13(2):23:1–23:31, 2017.

10 Deeparnab Chakrabarty, Prachi Goyal, and Ravishankar Krishnaswamy. The non-uniform k-center problem. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 55 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 67, 15. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2016.

11 Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002.

12 Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms (SODA)*, pages 642–651, 2001.

13 Danny Z. Chen, Jian Li, Hongyu Liang, and Haitao Wang. Matroid and knapsack center problems. *Algorithmica*, 75(1):27–52, 2016.

14 Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017 (NIPS)*, pages 5036–5044, 2017.

15 Marek Cygan, MohammadTaghi Hajiaghayi, and Samir Khuller. LP rounding for k-centers with non-uniform hard capacities. In *53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 273–282, 2012.

16 Marek Cygan and Tomasz Kociumaka. Constant factor approximation for capacitated k-center with outliers. In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 251–262, 2014.

17 Hu Ding, Lunjia Hu, Lingxiao Huang, and Jian Li. Capacitated center problems with two-sided bounds and outliers. In *Proceedings of the 15th International Symposium on Algorithms and Data Structures (WADS)*, pages 325–336, 2017.

18 Hu Ding and Jinhui Xu. Solving the chromatic cone clustering problem via minimum spanning sphere. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 773–784, 2011.

19 Hu Ding and Jinhui Xu. A unified framework for clustering constrained data without locality property. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1471–1490, 2015.

20 Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Approximating connected facility location with lower and upper bounds via LP rounding. In *15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 1:1–1:14, 2016.

21 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

22 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31(1):228–248, 1999.

23 Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33(3):533–550, 1986.

24 Wen-Lian Hsu and George L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979.

25 Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 731–740, 2002.

26 Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

27 Samir Khuller, Robert Pless, and Yoram J. Sussmann. Fault tolerant k-center problems. *Theoretical Computer Science*, 242(1-2):237–245, 2000.

28 Samir Khuller and Yoram J. Sussmann. The capacitated K-center problem. *SIAM Journal on Discrete Mathematics*, 13(3):403–418, 2000.

29 Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of Algorithms*, 37(1):146–188, 2000.

30 Jian Li, Ke Yi, and Qin Zhang. Clustering with diversity. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 188–200, 2010.

31 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 222:45–58, 2013.

32 Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. *SIAM Journal on Computing*, 45(2):530–547, 2016.

33 Clemens Rösner and Melanie Schmidt. Privacy preserving clustering with constraints. *CoRR*, abs/1802.02497, 2018. `arXiv:1802.02497`.

34 David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing (STOC)*, pages 265–274, 1997.

35 Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pages 577–584, 2001.

NC Algorithms for Weighted Planar Perfect Matching and Related Problems

Piotr Sankowski
Institute of Informatics, University of Warsaw
sank@mimuw.edu.pl

Abstract

Consider a planar graph $G = (V, E)$ with polynomially bounded edge weight function $w : E \to [0, \mathsf{poly}(n)]$. The main results of this paper are NC algorithms for finding minimum weight perfect matching in G. In order to solve this problems we develop a new relatively simple but versatile framework that is combinatorial in spirit. It handles the combinatorial structure of matchings directly and needs to only know weights of appropriately defined matchings from algebraic subroutines.

Moreover, using novel planarity preserving reductions, we show how to find: maximum weight matching in G when G is bipartite; maximum multiple-source multiple-sink flow in G where $c : E \to [1, \mathsf{poly}(n)]$ is a polynomially bounded edge capacity function; minimum weight f-factor in G where $f : V \to [1, \mathsf{poly}(n)]$; min-cost flow in G where $c : E \to [1, \mathsf{poly}(n)]$ is a polynomially bounded edge capacity function and $b : V \to [1, \mathsf{poly}(n)]$ is a polynomially bounded vertex demand function. There have been no known NC algorithms for these problems previously.

2012 ACM Subject Classification Theory of computation → Network flows, Theory of computation → Shared memory algorithms, Theory of computation → Pseudorandomness and derandomization

Keywords and phrases planar graph, NC algorithms, maximum cardinality matching, maximum weight matching, min-cost flow, maximum multiple-source multiple-sink flow, f-factors

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.97

Related Version A full version of the paper is available at [36], `https://arxiv.org/abs/1709.07869`.

Acknowledgements I would like to thank anonymous reviewers for helpful comments on this and previous version of this paper. This work was partially supported by grant 2014/13/B/ST6/01811 of Polish National Science Center and ERC StG grant TOTAL no. 677651.

1 Introduction

In this paper we study deterministic parallel algorithms for the maximum planar matching problem. In particular, we concentrate our attention on the NC class where we are given polynomially many processors which need to solve the problem in polylogarithmic time. The fundamental algorithmic challenge is to compute a maximum cardinality matching, or a maximum weight matching, or a minimum weight perfect matching.

So far in the case of NC algorithms for planar graphs there seemed to be no tools to attack the most general case and we were only able to find perfect matchings in planar bipartite graphs [30, 29]. This might be bit surprising as in non-planar graph, perfect matchings are essentially as powerful as maximum cardinality matchings, see e.g., [34] for a reduction. Such reductions either make two copies of the graph, where each vertex is connected with its copy, or add a set of vertices that are connected to every vertex in the graph. However,

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 97; pp. 97:1–97:16


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

there exists no similar planarity preserving reduction, and so the usefulness of algorithms for planar perfect matchings is limited. Hence, in order to find a maximum cardinality matching we needed to retool to less efficient polynomial time algorithms [14, 18, 19] that are limited to bipartite case only. Or, alternatively, we could find a 2-approximate solution [21]. The lack of such planarity preserving reduction is one of the reasons why computing maximum cardinality matchings in almost linear time was recognized as one of the core problems in planar graph algorithms [4]. This despite the fact that almost linear time algorithms for the perfect matching problem existed before [31, 11].

Formally speaking, our results are as follows. Consider a planar graph $G = (V, E)$ with polynomially bounded edge weight (cost) function $w : E \to [0, \mathsf{poly}(n)]$. A *matching* in G is an independent set of edges, whereas a *perfect matching* is a matching that is incident to every vertex in G. We start by giving a relatively simple NC algorithm for finding a perfect matching in G. Then we extend this algorithm to finding *minimum weight perfect matching*, i.e., a perfect matching M of G with minimum total weight $w(M)$. Our algorithm is combinatorial in spirit as it does not manipulate a fractional matching, but only requires to know weights of appropriately defined minimum weight perfect matchings. These weights can be computed using Kasteleyn's Pfaffian orientation of planar graphs and standard algebraic techniques. The algorithm is based on the fundamental notion of balanced duals that was developed in [7] as well as in [16]. This idea allows to define and construct a dual solution in a unique way. In particular, a very simple NC algorithm for constructing a family of blossoms of this dual follows by a direct application of the algorithm given in [7].

The next problem we consider is the *minimum weight f-factor problem*, where for a given $f : V \to [1, \mathsf{poly}(n)]$ we need to find a set of edges F of minimum total cost $w(F)$ such that $\deg_F(v) = f(v)$ for all $v \in V$. Typically, this problem is reduced to the perfect matching problem via vertex splitting [15, 3]. These reductions, however, do not preserve planarity. Our contribution, is to show a new planarity preserving reduction, that allows to solve the f-factor problem in NC. Due do space limitations of this paper, this reduction and the next ones are given in the full version of this paper available on arXiv [36].

The following implication of this result was rather surprising (at least for the author). We show that the maximum weight bipartite planar matching problem can be efficiently reduced to the minimum weight non-bipartite planar perfect matching problem. Thus implying the first known NC algorithm for finding maximum weight matching in bipartite planar graphs. Moreover, our reduction preserves the size of the graph, thus any further development for weighted perfect matching in planar graphs will imply similar results for maximum weight bipartite matchings. There seem to be no easy way to extend this result to non-bipartite case, and we note that finding NC algorithms for this problem remains open. Still, we report partial progress on this problem by showing the first $o(n)$ time PRAM algorithm for maximum cardinality matching – again available in the full version of this paper.

Finally, we consider directed flow networks where capacity of edges is given by a polynomially bounded capacity function $c : E \to [1, \mathsf{poly}(n)]$ and vertex demand is given by a polynomially bounded demand function $b : V \to [1, \mathsf{poly}(n)]$. Our aim is to compute a min-cost flow that obeys edge capacities and satisfies all vertex demands. We give the first known NC algorithm for this problem resolving the long standing open problem from [31]. This result is further extended to finding maximum multiple-source multiple-sink flow in planar graphs.

Related Work

The key result that allowed development of NC algorithms for the perfect matching problems in planar graph is Kasteleyn's idea [25]. He showed that the problem of counting perfect matchings in planar graphs is reducible to determinant computation. Thus [6] implied an NC algorithm checking whether a planar graph contains a perfect matching. In [41] this result was extended to $K_{3,3}$-free graphs. Counting, however, does not allow to construct a perfect matching. The first algorithm for constructing perfect matchings was given in [31] but only in the bipartite case. Another solution that is based on different principles was developed in [29]. A partial solution for the non-bipartite case was given in [27] where an algorithm for computing half integral solution was shown. Finally, an algorithm for non-bipartite planar graphs has been proposed in [2]. The result of [2] extends to finding minimum weight perfect matchings as well. Here, we give an independent proof of this result.[1] In comparison to [2], our paper gives a simpler algorithm for constructing a perfect matchings in NC than the one in [2]. Moreover, we show several nontrivial extension of this results, i.e., to maximum weight bipartite matching, to minimum weight f-factor, or min-cost flow. We note that previously we knew how to solve only (non-weighted) maximum cardinality matching in bipartite graphs in NC [22]. However, this result contains a gap in the argument, as is based on [9] which embeds planar graph into a grid graph what does not preserve the size of maximum matchings. This problem can be solved by using [26] or [8, 40]. Moreover, in [1] it was shown how to construct perfect and weighed perfect matchings in NC when the number of perfect matchings is polynomially bounded.

When allowing randomization (i.e., when considering RNC complexity) the problem can be solved even in general non-planar graphs [24, 17, 33, 35]. All of these Monte Carlo algorithms can be turned into Las Vegas ones using [23]. Finally, we note that we are unaware of any previous parallel deterministic algorithms for weighted non-bipartite problems like minimum weight perfect matching or f-factor problems. Bipartite versions of these problem have some solutions that require at least polynomial $\Omega(n^{2/3})$ time [14, 18, 19]. Finally, we note that very recently it was shown that the general problem is solvable in quasi-NC, i.e., in polylogarithmic time using quasi-polynomially many processors [12, 38]. Moreover, the bipartite problem can be solved in pseudo-NC, i.e., in polylogarithmic time, using polynomially many processors and polylogarithmically many random bits [20].

2 Preliminaries

$G = (V, E)$ denotes an n-vertex, embedded, undirected graph. This embedding partitions the plane into maximal open connected sets and we refer to the closures of these sets as the *faces* of G. The *dual* G^* of G is a multigraph having a vertex for each face of G. For each edge e in G, there is an edge e^* in G^* between the vertices corresponding to the two faces of G adjacent to e. We identify faces of G with vertices of G^* and since there is a one-to-one correspondence between edges of G and edges of G^*, we identify an edge of G with the corresponding edge in G^*.

[1] Our framework was developed independently from [2] and was posted on arXiv on the same day [36]. Nevertheless, the author's initial write-up was very "crude" and contained some gaps. In particular, Algorithm 1 was described in a confusing way without stating which parts should happen in parallel. Moreover, we aimed to give, an alternative to [7], construction of matching duals that was incorrect. This, however, is a known result, so in this paper we just cite [7] instead. Finally, the presentation of this paper was greatly improved with respect to [36], while keeping the original framework. Moreover, consequences for more general problems have been added.

The number of faces is denoted by f. For a subset of vertices $U \subseteq V$, $\delta(U)$ denotes all edges $uv \in E$ having $|\{u, v\} \cap U| = 1$. We write $\delta(u)$ for $\delta(\{u\})$, $x(F)$ for $\sum_{e \in F} x_e$ and $\deg_F(u) = |F \cap \delta(u)|$.

The linear programming formulation of the minimum weight perfect matching problem and its dual is as follows [10]. An *odd set* has odd size; Ω is the collection of odd subsets of V of size ≥ 3.

LP of minimum weight perfect matching

$$\min \sum_{e \in E} w(e)x_e$$
$$x(\delta(v)) = 1, \forall v \in V$$
$$x(\delta(U)) \geq 1, \forall U \in \Omega$$
$$x_e \geq 0, \forall e \in E$$

LP of the dual prolem

$$\max \sum_{v \in V} \pi_v + \sum_{U \in \Omega} \pi_U$$
$$\pi_u + \pi_v + \sum_{U \in \Omega,\ uv \in \delta(U)} \pi_U \leq w(uv), \quad \forall uv \in E \quad (*)$$
$$\pi_U \geq 0, \qquad \forall U \in \Omega$$

The variables x_e in the primal indicate when an edge is included in the solution. The dual problem has variables π_v for each vertex v and π_U for each odd set U.

Moreover, a graph G is *factor critical* if for all $v \in V$ after removing v the graph has a perfect matching. A *laminar family* is a collection $\mathcal{B}$ of subsets of V such that for every pair $X, Y \in \mathcal{B}$ either $X \cap Y = \emptyset$, or $X \subseteq Y$, or $Y \subseteq X$. We use the existence of the following dual.

▶ **Lemma 1** (implicitly in [10]). *If the dual LP is feasible, then there exists an optimal dual solution $\pi : V \cup \Omega \to \mathbb{R}$ that:*

1. *the set system $\{U \in \Omega : \pi_U > 0\}$ forms a laminar family,*
2. *for each $U \in \Omega$ with $\pi_U > 0$, the graph denoted by G_U obtained by contracting each set $\{S \in \Omega : S \subset U, \pi_S > 0\}$ to a point is factor critical.*

An optimum dual solution π satisfying the above conditions is a *critical dual solution*. A set $U \in \Omega$ such that $\pi_U > 0$ is a *blossom* w.r.t. π. An important idea that is used in almost all algorithms for weighted matching is that after computing the dual we can work with a non-weighted problem. This non-weighted problem is defined in the following way: leave only *tight* edges in the graph, i.e., there is equality in (*); find a perfect matching *respecting* all blossoms $\mathcal{B}$, i.e., such that for all $B \in \mathcal{B}$ we have $|M \cap \delta(B)| = 1$. By duality slackness any matching obtained this way is a minimum weight perfect matching.

Laminar family of sets is equipped with a natural parent-child relation and can be seen as a forest. We assume this tree is rooted at V and call the resulting tree the *laminar tree*. We note that given a laminar family it is straightforward to deduce the parent-child relation, i.e., parent of a set B is the minimal set containing B, whereas children of B are maximal sets contained in B. Hence, whenever working with a laminar family we assume that the laminar tree is available as well as it can be easily computed in NC. A useful property of this view is that tree T has a *vertex separator*, i.e., there exists a vertex v such that the size of every connected component of $T - x$ is at most $\frac{|T|}{2}$.

Basic NC Algorithms

Our algorithm builds upon the following NC algorithms for computing:

- components and a spanning forest of an undirected graph [37, 5],
- paths in a directed graph – this can be done by repetitive matrix squaring,
- a maximal independent set in a graph [28],
- a vertex separator of a tree – by computing the numbers of vertices in each subtree using any of the standard techniques [32, 39].

Consider a graph G with an edge weight function $w : E \to [0, \mathsf{poly}(n)]$. For a vertex v we denote by G_v the graph $G - v$. If G has an even number of vertices then M_v denotes some *minimum weight almost perfect matching* in G_v, i.e., a minimum weight matching that misses exactly one vertex. If G has an odd number of vertices then M_v denotes some minimum weight perfect matching in G_v. M_v is not defined in a unique way, but its weight $w(M_v)$ is. In our algorithms we will only use these weights that can be computed using standard techniques (see full version for the proof).

▶ **Corollary 2.** *For a graph $G = (V, E)$ with edge weights $w : E \to [0, \mathsf{poly}(n)]$ we can in NC:*
- *for a given vertex $v \in V$, compute the weight $w(M_v)$,*
- *for a given edge $e \in E$, check whether e is* allowed*, i.e., belongs to some minimum weight perfect matching.*

Observe that the set of allowed edge is a subset of tight edges. Hence, when we remove all not allowed edges only tight edges are left in the graph.

Let us now state the following implication of the results in [7]. Basically, assuming that all $w(M_v)$ are given, Algorithm 5 from [7] gives an NC procedure for computing the blossoms of the critical dual solution.

▶ **Lemma 3** (based on Lemma 6.19 [7])**.** *Let $G = (V, E)$ be undirected connected graph where edge weights are given by $w : E \to \mathbb{Z}$ and where every edge is allowed. Given all values $w(M_v)$ for $v \in V$, the blossoms of the critical dual solution can be computed in NC.*

Algorithm 5 from [7] actually constructs a critical dual with an additional property which is called balanced. In a *balanced dual* the root of the laminar tree is required to be the central vertex of this tree as well. This makes balanced duals unique. Thus when one constructs them in parallel, all processors construct the same solution, and it can be implemented in NC.

3 The High Level Idea: Walks and Blossoms

This section aims to introduce two core ideas of our algorithm that allow us to reduce the size of the graph for the recursion, as well as a high level idea that reveals around them. We will first give an algorithm for finding a perfect matching in a graph. However, we will view the problem as weighted and seek a minimum weight perfect matching. The weighed view is useful as we will find even length walks in the graph and introduce weights on them. These weights will either cause some edges to become not allowed, or induce a blossom as shown by the following lemma. Hence, till Section 7, where we show how to handle weighted case, $w : E \to [0, \mathsf{poly}(n)]$ denotes weights defined by the algorithm as the graph considered is unweighted.

To make it more precise we say that a closed walk C in graph G is *semi-simple* if it contains an edge that appears on C only once. By e_C we denote arbitrary such edge on C.

▶ **Lemma 4.** *Consider $G = (V, E)$ with edge weight function $w : E \to [0, \mathsf{poly}(n)]$. Let C be an even semi-simple closed walk in G. Let $w(e_C) = 1$ and let $w(e) = 0$ for all $e \in C - e_C$. Then, either some edge of C is not allowed or some edge of C is in $\delta(B)$ for some blossom B.*

Proof. Assume by a contradiction that all edges of C are allowed and there is no blossom intersecting C. Hence, by complementary slackness conditions we have that $\pi_x + \pi_z = w(xz)$ for all edges $xz \in C$. In particular, for the edge $u_C v_C = e_C$ we need to have $\pi_{u_C} + \pi_{v_C} = 1$, whereas for all other edges $uv \in C - e_C$ we have $\pi_u + \pi_v = 0$. Now consider edge $zy \in C$ which is next to xz. By subtracting equalities for edge xz and zy we obtain $\pi_x - \pi_y = w(xz) - w(zy)$.

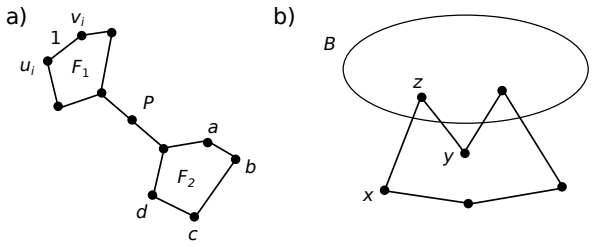


Figure 1 Figure a) shows a non-simple even length closed walk composed out of two faces F_1, F_2 and a path P. We have $\pi_a + \pi_b = 0$, $\pi_b + \pi_c = 0$ and $\pi_c + \pi_d = 0$, what gives $\pi_a - \pi_c = 0$ and $\pi_a + \pi_d = 0$. In Figure b) even length closed cycle does not contain an edge inside a blossom. In such a case $\pi_z + \pi_x + \pi_B = 0$ and $\pi_z + \pi_y + \pi_B = 0$, so $\pi_x - \pi_y = 0$.

If yz and xz are not equal to e_C we have $\pi_x - \pi_y = 0$. In general, we have $\pi_x - \pi_y = 0$ for any two vertices at even distance along path $C - e_C$. And $\pi_x + \pi_y = 0$ for any two vertices at odd distance along path $C - e_C$. Note that the distance from u_C to v_C along $C - e_C$ is odd, so we obtain $\pi_{u_C} + \pi_{v_C} = 0$, what leads to contradiction. See Figure 1 a) for an illustration. ◀

Blossoms are natural objects to recurse on, as by duality slackness there must be exactly one edge of any perfect matching M that belongs to $\delta(B)$ for any blossom B. Thus, in the recursion, we can find an almost perfect matching outside of B, an almost perfect matching inside of B and then combine them by matching one edge in $\delta(B)$ – see Section 6. However, having an edge in $\delta(B)$ does not directly guarantee that the size of the graph reduces in the recursion, as the same blossom can intersect many closed walks. We need the following stronger observation for this.

▶ **Lemma 5.** *Consider $G = (V, E)$ with edge weight function $w : E \to [0, \mathsf{poly}(n)]$. Let C be an even semi-simple closed walk in G and assume that all edges on C are allowed. Let $w(e_C) = 1$ and let $w(e) = 0$ for all $e \in C - e_C$. Then there exist edges $e_1, e_2 \in C$, such that $e_1 \in E(G \setminus B)$ and $e_2 \in E(B)$ for some blossom B.*

Proof. For contradiction assume that no edge of C is in $E(B)$ – the other case is symmetric. Hence, there can only be vertices z in $E(B)$ such that their both incident edges xz and zy on C are in $\delta(B)$. In such a case we have $\pi_x + \pi_z + \pi_B = w(xz)$ and $\pi_z + \pi_y + \pi_B = w(zy)$. Thus $\pi_x - \pi_y = w(xz) - w(zy)$, i.e., the contribution of π_B when subtracting these equalities cancels. See Figure 1 b) for illustration. Thus B does not contribute anything to the telescoping sum along $C - e_C$ and we reach similar contradiction as in the previous lemma. ◀

Hence, when recursing on the inside of the blossom or on the outside of the blossom we reduce the graph size as well. However, to obtain an NC algorithm we need to reduce the size of the graph by a constant factor, so we need to have $\Omega(n)$ edge disjoint even closed walks. The following lemma, that was implicitly proven in [27], becomes handy now.

▶ **Lemma 6.** *In a 2-connected planar graph G with f faces we can find $\Omega(f)$ edge disjoint even semi-simple closed walks in NC.*

Intuitively, the proof of this lemma puts faces into pairs that are incident. Now, either a pair contains an even face and thus this face is semi-simple, or both faces are odd. In the later case we can build an even semi-simple closed walk by walking around both faces.

The above lemma shows that in order to have many even semi-simple closed walks we just need to guarantee that the graph has many faces. Let us say that a planar graph G is *simplified*, if there are no degree 1 vertices and no two vertices of degree 2 are incident. The next lemma shows that such graphs have many faces.

▶ **Lemma 7.** *Let $G = (V, E)$ be a simplified planar graph with a perfect matching, then G has at least $\frac{n}{4} + 2$ faces.*

Proof. By Euler's formula $n - m + f = 2$, where m is the number of edges, and f is the number of faces. Let V_2 be the set of degree 2 vertices in G. Consequently, the vertices in $V \setminus V_2$ have degree at least 3. As G has a perfect matching and no two degree 2 vertices are incident, all vertices in V_2 need to be matched to vertices in $V \setminus V_2$, i.e., $|V_2| \leq |V \setminus V_2|$. By using this inequality we have.

$$2m = \sum_{v \in V} \deg(v) = \sum_{v \in V_2} \deg(v) + \sum_{v \in V \setminus V_2} \deg(v) \geq 2|V_2| + 3|V \setminus V_2| \geq \frac{5}{2}|V_2| + \frac{5}{2}|V \setminus V_2| = \frac{5n}{2}.$$

By plugging this inequality into Euler's formula we obtain $f = 2 + m - n \geq 2 + \frac{5n}{4} - n = 2 + \frac{n}{4}$. ◀

A simplified graph has many faces and thus many edge disjoint even length closed walks. We can assign weights to each of these closed walks separately. This way either many edges become not allowed, or there exists a family of blossoms containing many edges inside. The main technical part of the algorithm is to handle this family of blossoms. The algorithm has the following steps.

1. Simplify the graph as shown in Section 4. First, we take care of degree 1 vertices, by removing not allowed edges and matching edges incident to degree 1 vertices. Next, we contract paths composed of degree 2 vertices. Finally, we find perfect matchings in each connected component separately. Due to removal of not allowed edges each connected component is 2-connected as well.
2. Using Lemma 6 we find many edge disjoint even length closed walks. We assign weights to even closed walks using Lemma 4 and we remove not allowed edges. Next, we find blossoms of the critical dual using Lemma 3. These steps are described in Section 5.
3. Finally, we recurse on a critical dual as explained in Section 6, where we show how to construct a perfect matching that respects all blossoms in the dual. This construction is recursive and calls back step 1 for subgraphs of G that do not have any blossoms.

The recursion depth in this procedure is $O(\log n)$ as either $\Omega(n)$ edges become not allowed, or there are many edges separated by blossoms. In the second case for each blossom B_e there is an edge e_1 inside and an edge e_2 outside of it – see Figure 2 a). We observe that blossom divide the graph into regions and each lowest level recursive call goes to one of these regions, i.e., we recourse on inside and outside of some blossom as long as there are some blossoms intersecting the current subgraph. As visualized on Figure 2 b) there are many edges that are not incident to each region, as one of the edges from each pair (e_1, e_2) needs be outside of this region. Hence, the size of each of these regions decreases by a constant factor with respect to the original graph. This will be proven more formally in Section 5 where we analyze the running time of the algorithm.

4 Simplifying the Graph

The first ingredient of our algorithm is the following entry procedure – Algorithm 1 that simplifies the graph and assures that we are working with 2-connected graphs.

The following lemma proves the correctness of this algorithm.

▶ **Lemma 8.** *Algorithm 1 executes Algorithm 2 on a simplified 2-connected graph.*

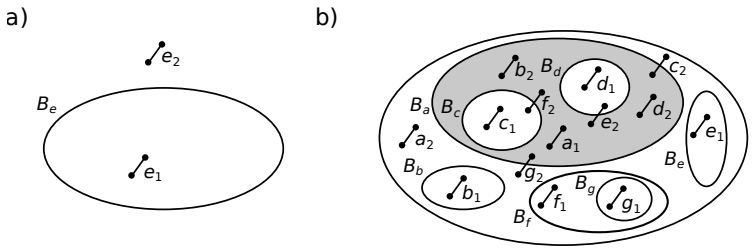


Figure 2 For each blossom B_e there needs to be a pair of edges: e_1 inside and edge e_2 outside of B_e – Figure a). Figure b): when we recurse from Algorithm 4 to Algorithm 1 the blossom tree is empty, and we are recursing on one of the regions into which blossoms divide the plane – marked with gray. Observe that at least one edge from each pair is not incident to this region.

Algorithm 1 Simplifies graph G and seeks perfect matchings in its 2-connected components.

Remove all not-allowed edges from G.
Add edges incident to degree 1 vertices to M and remove them from G.
for all maximal paths p composed out of degree 2 vertices **do** ▷ In parallel
    if p has odd length **then**
        remove p and connect vertices incident to p with an edge e_p.
    else
        remove p and add a single vertex v_p connected to vertices incident to p.
for all connected components $C \in \mathcal{C}$ of G **do** ▷ In parallel
    find perfect matching M_C using Algorithm 2.
Extend matching M on paths of degree 2 vertices.
return $M \cup \bigcup_{C \in \mathcal{C}} M_C$.

Proof. We first observe that after removing not allowed edges, an edge e incident to a vertex of with degree 1 needs to belong to all perfect matchings. Hence, it's both endpoints need to be of degree 1. After matching such edges all vertices have degree 2 or higher. Now note that the manipulation of the path p does not change degrees of other vertices. In the algorithm such path is either replaced by an edge, or single vertex. Hence, afterwards degree 2 vertices cannot be incident. Thus G is simplified and its all connected components are simplified.

Now, for contradiction assume that a connected component C is not 2-connected, i.e., there exists an articulation point v. Consider the connected components obtained from C after removal of v. Only one of them can be matched in the perfect matching to v. Thus only this one has odd number of vertices. The remaining components must have even number of vertices and no perfect matching can match them to v. Hence, their edges incident to v are not allowed. However, all not allowed edges were removed by the algorithm, so we reach a contradiction. ◀

Finally, we note that the above algorithm can be implemented in NC. First of all, in order to compute maximal paths composed out of degree 2 vertices, we just need to compute connected components of the subgraph induced by degree 2 vertices. In the final step of the algorithm we need to expand all paths that were replaced during the execution of Algorithm 1. Observe that the matching M in the simplified graph can be extended to the matching in the original graph in a straight-forward way. If p was odd, then depending on whether e_p is matched we either match even or odd edges of p. If p was even, then v_p has degree 2 and it can be matched in one of two possible ways. In these two cases the matching can be extended to the whole path.

Algorithm 2 Finds a perfect matching M in a connected graph G.

1: Find a set $\mathcal{F}$ of $\Omega(n)$ edge disjoint even semi-simple closed walks using Lemma 6.
2: Set $w(e) = 0$ for all $e \in E$.
3: **for all** $C \in \mathcal{F}$ **do** ▷ In parallel
4: Set $w(e_C) = 1$.
5: Remove all not-allowed edges from G.
6: Compute blossoms of a critical dual $\mathcal{B}$ with respect to w using Lemma 3.
7: Compute a matching M that respects $\mathcal{B}$ using Algorithm 4.
8: Return M.

5 The Main Routine

Algorithm 2 implements our main procedure. First, we find $\Omega(n)$ even semi-simple closed walks and introduce weights on them. Next, in order to reduce the size of the graph we remove not allowed edges. We then find blossoms of the critical dual solution with respect to these weights and call Algorithm 4 to find a perfect matching that respects all blossoms.

6 Finding a Perfect Matching that Respects a Family of Blossoms

Let $G = (V, E)$ be a graph and let M be a matching in G. An *alternating path* p is a path in G such that edges on p alternate between matched and unmatched. Assume that G is factor critical (e.g., inside of a blossom) and that M_s is an almost perfect matching that misses vertex s. We start by showing how using M_s we construct an almost perfect matching M_v for any v. To this end, we need to find a simple alternating path starting in s and ending in v. Denote by w_M a weight function assigning 0 to edges in M_s and 1 to edges not in M_s.

▶ **Lemma 9.** *Let M_v be the minimum weight almost perfect matching in G_v with respect to w_M, then $2w_M(M_v)$ is the length of the shortest alternating path with respect to M_s from s to v.*

Proof. Observe that the symmetric difference $M_s \oplus M_v$ contains an alternating path with respect to M_s that needs to start at s and end at v. The weight of this path is equal to the number of edges from M_s on it. As this path is alternating, the number of edges of M_v is the same. Thus minimizing $w_M(M_v)$ we minimize the length of an alternating path with respect to M_s from s to v. ◀

Now, we want to construct a graph G_L that will represent all shortest alternating paths from s with respect to M. G_L will be a layered graph, where layer l contains vertices at distance l from s — the distance is measured along alternating paths. For each $v \in V$, let G_L contain two copies v^o and v^e of v. We define $l(v^e) = 2w_M(M_v)$ for all $v \in V$, and $l(v^o) = l(u^e) - 1$ for all $uv \in M$. We add edges of G to G_L only if they connect two consecutive layers given by l – see Figure 3. Every shortest alternating path from s is contained in G_L by Lemma 9. Alteratively, if a path in G_L represents a simple path in G then it is a shortest alternating path in G. However, there are paths in G_L that do not correspond to simple paths in G, i.e., they contain both v^o and v^e for some $v \in V$ – see Figure 3 b). Nevertheless, as every vertex in G is reachable via alternating path, we can modify G_L in such a way that only paths corresponding to simple paths in G remain. This is done using Algorithm 3.

The correctness of this algorithm is established by the next lemma.

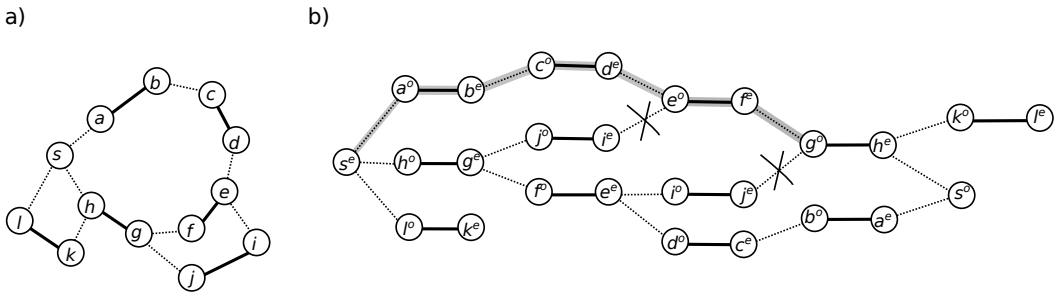


Figure 3 Figure a) shows a factor-critical graph, matched edges are marked with solid lines and s is the free vertex. Figure b) presents layered graph G_L from Algorithm 3. There are two non-simple paths represented by this graph that go from g^e to g^o. These paths are destroyed by the algorithm as g^o is reachable via a simple path p marked with grey.

Algorithm 3 Finds an alternating path with respect to M_s in G from a vertex s to vertex t.

1: For all $v \in V$ compute $w_M(M_v)$.
2: Let G_L be a graph where v has two copies v^o and v^e.
3: For all $v \in V$ set $l(v^e) = 2w_M(M_v)$.
4: For all $vu \in M$ set $l(v^o) = l(u^e) - 1$.
5: Add edges of G to G_L only if they connect vertices in consecutive layers l.
6: **for all** $v \in V$ **do** ▷ In parallel
7: **for all** u^z on some v^x-v^y path in G_L, where $x, y, z \in \{e, o\}$, $x \neq y$ **do** ▷ In parallel
8: **if** there exists s^e-u^z path p avoiding v^x in G_L **then**
9: remove all edges entering u^z but the edge on p.
10: Return any path from s^e to t^e in G_L.

▶ **Lemma 10.** *Let G be a factor critical graph and let M_s be an almost perfect matching missing vertex s. An almost perfect matching M_t missing vertex t can be found in NC using Algorithm 3.*

Proof. We first observe that the removal of edges entering u^z from G_L does not affect reachability from s, as the path p from s^e to u^z is left in the graph. Now, by contradiction, assume that at the end of the algorithm there is a path q in G_L that contains both v^e and v^o for some $v \in V$. Without loss of generality assume v^e precedes v^o. As G_L contains all simple alternating paths, there exists an s^e-v^o path p avoiding v^e. Hence, the edge of q entering the first shared vertex with p was removed by the algorithm – see Figure 3 b).[2] ◀

Algorithm 4 constructs a perfect matching M that respects a family of blossoms. Here, we explicitly consider $\mathcal{B}$ as a tree $T_{\mathcal{B}}$, i.e., the vertices of $T_{\mathcal{B}}$ are sets in $\mathcal{B}$ whereas edges in $T_{\mathcal{B}}$ represent child parent relationship. See Figure 4 for an example and an illustration of the recursion. It calls Algorithm 1 that handles the case without blossoms. The next theorem argues about the correctness of this algorithm.

▶ **Lemma 11.** *Algorithm 4 finds a perfect matching M respecting $\mathcal{B}$. The recursion depth of the internal calls of the algorithm to itself is $O(\log n)$.*

[2] The graph constructed in this algorithm can be seen as an extended version of generalized shortest path tree [13]. Alternatively, such tree could be constructed using Algorithm 5 from [7]. This, however, would result in a slightly more complicated solution.

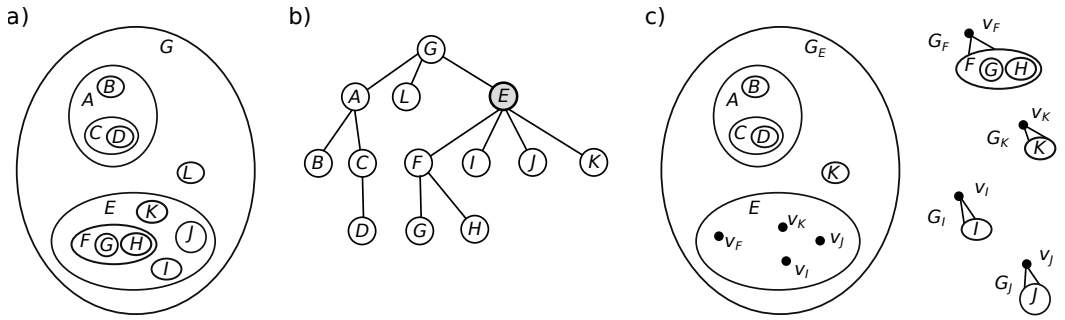


Figure 4 Figure a) shows a laminar family of sets. Figure b) gives the same family using laminar tree. Vertex separator – node E is marked with grey. Figure c) shows the graphs on which we recurse in Algorithm 4 using the marked separator E.

Algorithm 4 Computes a perfect matching M of G respecting family of blossoms $\mathcal{B}$.

if $\mathcal{B} = \emptyset$ **then**
  Return the matching M computed by Algorithm 1 on G.
Let B be a vertex separator of $T_{\mathcal{B}}$.
Begin ▷ In parallel with the next loop
  Let G_B be the graph with all children of B contracted.
  Let T_B be $T_{\mathcal{B}}$ with children of B removed.
  Recurse on G_B, T_B to obtain matching M_B.
End
for all children C of B in $T_{\mathcal{B}}$ **do** ▷ In parallel
  Let G_C be G with all vertices not in C contracted to a vertex denoted by v_C.
  Let T_C be subtree of $T_{\mathcal{B}}$ rooted at C.
  Recurse on G_C, T_C to obtain matching M_C.
for all children C of B in $T_{\mathcal{B}}$ **do** ▷ In parallel
  Remove from G_C vertex v_C and let u_C be the resulting free vertex in G_C.
  Let e_B be the edge of M_B incident to G_C.
  Let v_B be the endpoint of e_B in G_C after expanding G_C.
  Apply to M_C an alternating path from u_C to v_B found using Algorithm 3
Return $M_B \cup \bigcup_{C \text{ child of } B} M_C$.

Proof. We need to argue that M_B can be extended to a perfect matching in the whole graph. Consider a child blossom C of B. By Lemma 3 we know that C is factor-critical, so there exists an almost perfect matching M'_C in C that together with M_B forms a perfect matching. This matching differs from M_C by a single alternating path. Hence, for each child blossom the algorithm is able to combine M_B with M_C by finding a single alternating path.

As we do not recompute the family of blossoms while recursing, we need to argue that the blossoms remain intact. Observe that after contraction of a given blossom all nonintersecting blossoms remain blossoms and the graph remains planar, so we can continue recursing on subtrees of $T_{\mathcal{B}}$.

Finally, the recursion depth of the algorithm to itself is $\Omega(\log n)$, because we recurse on a vertex separator of a tree $T_{\mathcal{B}}$, so the size of the subtrees decreases by a constant factor. ◀

Lemma 11 leads to the correctness of Algorithm 2 and Algorithm 1 as well.

▶ **Theorem 12.** *Algorithm 1 finds a perfect matching in G.*

Algorithm 5 Finds a minimum weight perfect matching in $G = (V, E)$ with respect to the edge weight function $w : E \to [0, \mathsf{poly}(n)]$.

Remove all not allowed edges from G.
Compute blossoms of a critical dual $\mathcal{B}$ with respect to w using Lemma 3.
Find a perfect matching M respecting $\mathcal{B}$ using Algorithm 4.
Return M.

We now can turn our attention to arguing about the running time of our algorithm. In this section we are going to quantify the progress related to perturbing weights on each semi-simple closed walks. The entry point to our procedures is Algorithm 1. This algorithm calls Algorithm 2, which constructs a family of blossoms. Now, Algorithm 4 is used to recurse on this family till subgraphs with no blossoms are obtained. At this moment Algorithm 1 is called back. We will argue that each time Algorithm 1 is called in this recursion, the size of the graph decreases by a constant factor.

▶ **Lemma 13.** *The number of edges in G decreases by a constant factor in each recursive call to Algorithm 1.*

Proof. By Lemma 6 and Lemma 7, after graph simplification we have $\Omega(n)$ even semi-simple closed walks in the graph G. Now, by Lemma 4, for each semi-simple closed walks, either one edge becomes not-allowed, or there exists a blossom $B \in \mathcal{B}$ that intersects this closed walk. Hence, either $\Omega(n)$ edges become not-allowed, or we have $\Omega(n)$ closed walks intersected by some blossom. Lemma 5 implies that for each such intersection there exists an edge e_1 inside B and an edge e_2 outside B – see Figure 2 a). Consider a plane embedding of G and of dual graph G^*. Mark boundaries of each blossom $\delta(B)$ in G^* on the plane, thus dividing the plane into maximal open connected sets. We call these sets regions $\mathcal{R}$.

The outside of each region $R \in \mathcal{R}$ contains at least one edge from e_1 and e_2, i.e., there are $\Omega(n)$ edges not incident to a region – see Figure 2 b). We note that when we call Algorithm 1 from Algorithm 4, we recurse onto some region $R \in \mathcal{R}$ with parts of G not in R contracted to vertices. This graph contains only edges incident to a region R, so it does not contain $\Omega(n)$ edges. By Euler's formula the total numer of edges is $\leq 3n - 6$, so when recursing on each region it decreases by a constant factor. ◀

By Lemmas 11 and 13 the recursion depth in Algorithm 2 is $O(\log^2 n)$, thus:

▶ **Corollary 14.** *A perfect matching in a planar graph can be computed in NC.*

7 Minimum Weight Perfect Matching

So far we have assumed that the graph is unweighed and we have coped with the problem of constructing any perfect matching. However, our approach is versatile enough to handle weighted case in a rather straightforward way using Algorithm 5.

Hence, we obtain the following.

▶ **Lemma 15.** *A minimum weight perfect matching in a planar graph $G = (V, E)$ with edge weight function $w : E \to [0, \mathsf{poly}(n)]$ can be computed in NC.*

Proof. Observe that all allowed edges in G need to be tight. By complementary slackness conditions, a perfect matching that is composed out of allowed edges and that respects all blossoms is a minimum weight perfect matching. ◀

We note that maximum weight perfect matching can be computed by using redefined weight function $w'(e) = -w(e) + \max_{g \in E} w(g)$.

8 Planarity Preserving Reductions

Let $G = (V, E)$ be a multi-graph, i.e., we allow parallel edges as well as self-loops. For a function $f : V \to [1, \mathsf{poly}(n)]$, an *$f$-factor* is a set of edges $F \subseteq E$ such that $\deg_F(v) = f(v)$ for every $v \in V$. Without loss of generality we assume that any edge uv has multiplicity at most $\min\{f(u), f(v)\}$. A *minimum f-factor* is an f-factor F with minimum weight $w(F)$. The f-factor problem can be reduced via planarity preserving reduction to minimum weight perfect matching problem.

▶ **Lemma 16.** *Let $G = (V, E)$ be a planar multigraph with edge weight function $w : E \to [0, \mathsf{poly}(n)]$. For a function $f : V \to [1, \mathsf{poly}(n)]$, minimum f-factor can be computed in NC.*

Now consider, a graph $G = (V, E)$ be a bipartite planar graph with edge weight function $w : E \to [0, \mathsf{poly}(n)]$, and a problem of computing a maximum matching in graph G, i.e., a matching of maximum total weight. This problem in turn can be reduced to f-factor problem.

▶ **Lemma 17.** *A maximum matching in a planar bipartite graph $G = (V, E)$ with edge weight function $w : E \to [0, \mathsf{poly}(n)]$ can be computed in NC.*

In the min-cost planar flow problem, we are given a directed planar network $N = (V, E)$. The edges have integral capacities given by $c : E \to [1, \mathsf{poly}(n)]$ and integral costs $a : E \to [0, \mathsf{poly}(n)]$. Moreover, each vertex has integral demand $b : V \to [-\mathsf{poly}(n), \mathsf{poly}(n)]$. Again there exists planarity preserving reduction to f-factor problem.

▶ **Theorem 18.** *Let $N = (V, E)$ be a planar flow network with integral capacities $c : E \to [1, \mathsf{poly}(n)]$, integral costs $a : E \to [0, \mathsf{poly}(n)]$ and integral demands $b : V \to [-\mathsf{poly}(n), \mathsf{poly}(n)]$. The minimum cost flow in N can be computed in NC.*

We can modify the above reduction to handle the case when source and sink demands are not fixed but need to be maximized. We are given a directed planar network $N = (V, E)$ with integral edge capacities given by $c : E \to [1, \mathsf{poly}(n)]$. Moreover, each vertex has integral demand $b : V \to [-\mathsf{poly}(n), \mathsf{poly}(n)]$. Vertices $v \in V$ such that $b(v) \geq 0$ are called *sources* and we require for them $0 \leq f(v) \leq b(v)$. Vertices $v \in V$ such that $b(v) \leq 0$ are called *sinks* and we require $b(v) \leq f(v) \leq 0$. For remaining vertices v, i.e., when $b(V) = 0$ we need to have $f(v) = 0$. The *maximum multiple-source multiple-sink flow* in N is the flow that maximizes value $f(S)$, where S is the set of all sources.

▶ **Theorem 19.** *Let $N = (V, E)$ be a planar flow network with integral capacities $c : E \to [1, \mathsf{poly}(n)]$, integral costs $a : E \to [0, \mathsf{poly}(n)]$ and integral demands $b : V \to [-\mathsf{poly}(n), \mathsf{poly}(n)]$. The minimum cost flow in N can be computed in NC.*

References

1. Manindra Agrawal, Thanh Minh Hoang, and Thomas Thierauf. The polynomially bounded perfect matching problem is in NC2. In *Proceedings of the 24th Annual Conference on Theoretical Aspects of Computer Science*, STACS'07, pages 489–499, Berlin, Heidelberg, 2007. Springer-Verlag.
2. Nima Anari and Vijay V. Vazirani. Planar graph perfect matching is in NC, 2017. `arXiv:arXiv:1709.07822`.
3. R.P. Anstee. A polynomial algorithm for b-matching: An alternative approach. *IPL*, 24:153–157, 1987.

4 G. Borradaile, P. N. Klein, S. Mozes, Y. Nussbaum, and C. Wulff-Nilsen. Multiple-Source Multiple-Sink Maximum Flow in Directed Planar Graphs in Near-Linear Time. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, FOCS '11, pages 170–179, Oct 2011.

5 K.W. Chong and T.W. Lam. Finding Connected Components in $O(\log n \log\log n)$ Time on the EREW PRAM. *Journal of Algorithms*, 18(3):378–402, 1995.

6 L. Csanky. Fast Parallel Matrix Inversion Algorithms. *SIAM Journal on Computing*, 5(4):618–623, 1976.

7 Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic Applications of Baur-Strassen's Theorem: Shortest Cycles, Diameter, and Matchings. *J. ACM*, 62(4):28:1–28:30, 2015.

8 Samir Datta, Arjun Gopalan, Raghav Kulkarni, and Raghunath Tewari. Improved Bounds for Bipartite Matching on Surfaces. In *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14 of *LIPIcs*, pages 254–265, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

9 Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory of Computing Systems*, 47(3):737–757, Oct 2010.

10 Jack Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research National Bureau of Standards-B.,*, 69B:125–130, 1965.

11 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006. Special Issue on FOCS 2001.

12 Stephen Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-nc. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 754–763, New York, NY, USA, 2016. ACM.

13 H. N. Gabow and P. Sankowski. Algebraic algorithms for b-matching, shortest undirected paths, and f-factors. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '13, pages 137–146, Oct 2013.

14 Harold Gabow and Robert Tarjan. Almost-optimum speed-ups of algorithms for bipartite matching and related problems. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 514–527, New York, NY, USA, 1988. ACM.

15 Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 448–456, New York, NY, USA, 1983. ACM.

16 Harold N. Gabow. A combinatoric interpretation of dual variables for weighted matching and f-factors. *Theor. Comput. Sci.*, 454:136–163, 2012.

17 Zvi Galil and Victor Y. Pan. Improved processor bounds for combinatorial problems in RNC. *Combinatorica*, 8(2):189–200, 1988.

18 A. V. Goldberg, S. A. Plotkin, and P. M. Vaidya. Sublinear-time parallel algorithms for matching and related problems. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '88, pages 174–185, Oct 1988.

19 A. V. Goldberg, D. B. Shmoys, S. A. Plotkin, and E. Tardos. Interior-point methods in parallel computation. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, FOCS '89, pages 350–355, Washington, DC, USA, 1989. IEEE Computer Society.

20 Shafi Goldwasser and Ofer Grossman. Bipartite Perfect Matching in Pseudo-Deterministic NC. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*,

volume 80 of *LIPIcs*, pages 87:1–87:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

21 Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. On the distributed complexity of computing maximal matchings. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pages 219–225, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

22 T. M. Hoang. On the matching problem for special graph classes. In *Proceedings of the 25th Annual IEEE Conference on Computational Complexity*, CCC '10, pages 139–150, June 2010.

23 Howard J. Karloff. A Las Vegas RNC algorithm for maximum matching. *Combinatorica*, 6(4):387–391, 1986.

24 Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

25 P. W. Kasteleyn. Dimer statistics and phase transitions. *Journal of Mathematical Physics*, 4(2):287–293, 1963.

26 Arpita Korwar. Finding an NC algorithm for perfect matching in planar graphs. Master's thesis, IIT Kanpur, 2009.

27 Raghav Kulkarni and Meena Mahajan. Seeking a vertex of the planar matching polytope in nc. In Susanne Albers and Tomasz Radzik, editors, *In Proceedings of the 12th Annual European Symposium on Algorithms*, pages 472–483, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

28 M Luby. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 1–10, New York, NY, USA, 1985. ACM.

29 Meena Mahajan and Kasturi R. Varadarajan. A new NC-algorithm for finding a perfect matching in bipartite planar and small genus graphs (extended abstract). In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 351–357, New York, NY, USA, 2000. ACM.

30 Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, 1986.

31 Gary L. Miller and Joseph (Seffi) Naor. Flow in planar graphs with multiple sources and sinks. *SIAM Journal on Computing*, 24(5):1002–1017, 1995.

32 Gary L. Miller and John H. Reif. Parallel tree contraction part 1: Fundamentals. In Silvio Micali, editor, *Randomness and Computation*, pages 47–72. JAI Press, Greenwich, Connecticut, 1989. Vol. 5.

33 K. Mulmuley, U.V. Vazirani, and V.V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the nineteenth annual ACM conference on Theory of computing*, STOC '87, pages 345–354. ACM Press, 1987.

34 Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 118–126, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

35 Piotr Sankowski. Processor Efficient Parallel Matching. *Theory of Computing Systems*, 42(1):73–90, Jan 2008.

36 Piotr Sankowski. Planar perfect matching is in NC, 2017. `arXiv:arXiv:1709.07869`.

37 Y. Shiloach and Uzi Vishkin. An $O(\log n)$ parallel connectivity algorithm. *Journal of Algorithms*, 3:57–67, 1982.

38 Ola Svensson and Jakub Tarnawski. The Matching Problem in General Graphs Is in Quasi-NC. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science*, FOCS '17, pages 696–707. IEEE Computer Society, 2017.

39 R. E. Tarjan and U. Vishkin. Finding biconnected componemts and computing tree functions in logarithmic parallel time. In *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '84, pages 12–20, Oct 1984.

40 Raghunath Tewari and N. V. Vinodchandran. Green's theorem and isolation in planar graphs. *Inf. Comput.*, 215:1–7, 2012.

41 Vijay V. Vazirani. Nc algorithms for computing the number of perfect matchings in $K_{3,3}$-free graphs and related problems. *Information and Computation*, 80(2):152–164, 1989.

Computing Tutte Paths

Andreas Schmid
Max Planck Institute for Informatics, Saarbrücken, Germany
aschmid@mpi-inf.mpg.de

Jens M. Schmidt[1]
Technische Universität Ilmenau, Ilmenau, Germany
jens.schmidt@tu-ilmenau.de

Abstract

Tutte paths are one of the most successful tools for attacking problems on long cycles in planar graphs. Unfortunately, results based on them are non-constructive, as their proofs inherently use an induction on overlapping subgraphs and these overlaps prevent any attempt to bound the running time by a polynomial.

For special cases however, computational results of Tutte paths are known: For 4-connected planar graphs, Tutte paths are in fact Hamiltonian paths and Chiba and Nishizeki [5] showed how to compute such paths in linear time. For 3-connected planar graphs, Tutte paths have a significantly more complicated structure, and it has only recently been shown that they can be computed in polynomial time [24]. However, Tutte paths are defined for general 2-connected planar graphs and this is what most applications need. In this unrestricted setting, no computational results for Tutte paths are known.

We give the first efficient algorithm that computes a Tutte path (in this unrestricted setting). One of the strongest existence results about such Tutte paths is due to Sanders [23], which allows one to prescribe the end vertices and an intermediate edge of the desired path. Encompassing and strengthening all previous computational results on Tutte paths, we show how to compute such a special Tutte path efficiently. Our method refines both, the existence results of Thomassen [29] and Sanders [23], and avoids that the subgraphs arising in the inductive proof intersect in more than one edge by using a novel iterative decomposition along 2-separators. Finally, we show that our algorithm runs in time $O(n^2)$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Tutte Path, Tutte Cycle, 2-Connected Planar Graph, Hamiltonian Cycle

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.98

1 Introduction

The question whether a graph $G = (V, E)$ is *Hamiltonian*, i.e. contains a cycle of length $n := |V|$, is among the most fundamental graph problems. For planar graphs and graphs embeddable on higher surfaces, *Tutte paths* have proven to be one of the most successful tools for attacking Hamiltonicity problems and problems on long cycles. For this reason, there is a wealth of existential results in which Tutte paths serve as main ingredient; in chronological order, these are [31, 29, 26, 4, 22, 23, 27, 33, 16, 28, 11, 13, 18, 21, 20, 17, 24, 7, 2].

As a historical starting point to these results, Whitney [32] proved that every 4-connected maximal planar graph is Hamiltonian. Tutte extended this to arbitrary 4-connected planar graphs by showing that every 2-connected planar graph G contains a Tutte path [30, 31]

[1] This research is supported by the grant SCHM 3186/1-1 (270450205) from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation).

(for a definition of Tutte paths, see Section 2). Thomassen [29] in turn proved the following generalization, which also implies that every 4-connected planar graph is Hamiltonian-connected, i.e. contains a path of length $n-1$ between any two vertices. For a plane graph G, let C_G be its outer face.

▶ **Theorem 1** (Thomassen [29]). *Let G be a 2-connected plane graph, $x \in V(C_G)$, $\alpha \in E(C_G)$ and $y \in V(G) - x$. Then G contains a Tutte path from x to y through α.*

Sanders [23] then generalized Thomassen's result further by allowing to choose the start vertex x of the Tutte path arbitrarily.

▶ **Theorem 2** (Sanders [23]). *Let G be a 2-connected plane graph, $x \in V(G)$, $\alpha \in E(C_G)$ and $y \in V(G) - x$. Then G contains a Tutte path from x to y through α.*

On top of the above series of fundamental results, Tutte paths have been used in two research branches: While the first deals with the existence of Tutte paths on graphs embeddable on higher surfaces [26, 3, 27, 33, 28, 17], the second [15, 9, 3, 10, 16, 11, 19] investigates generalizations or specializations of Hamiltonicity such as *k-walks*, *long cycles* and *Hamiltonian connectedness*.

Unfortunately, in all the results mentioned so far, very little is known about the complexity of finding a Tutte path. This is crucial, as the task of finding Tutte paths is almost always the only reason that hinders the computational tractability of the problem. The main obstruction so far is that Tutte paths are found by decomposing the input graph into overlapping subgraphs, on which induction is applied. Although this is enough to prove existence results, these overlapping subgraphs do not allow to bound the running time polynomially (as argued in [12, 24]). The only known computational results on Tutte paths [12, 1, 5, 21, 24] deal therefore with very restricted settings, such as the case that Tutte paths are just Hamiltonian paths: While it is known how to compute Tutte paths for planar 4-connected graphs [5] efficiently (in which case Tutte paths are just Hamiltonian paths), for planar 3-connected graphs a first polynomial-time algorithm was only recently shown [24].

However, no efficient algorithm is published so far that computes Tutte paths in general 2-connected planar graphs (i.e. the ones of Theorem 1 or 2). In fact, the claimed algorithmic results in [26, 27] require polynomial running times for computing such Tutte paths, without giving proofs that such algorithms exist. Given the subtlety of the arguments inherent to Tutte paths, we feel that giving such a proof is necessary. Indeed, history shows that even for the much easier setting that Tutte paths are Hamiltonian paths, an existence result for Tutte paths has been incorrectly claimed to imply a polynomial-time algorithm [29, 4] (again, due to overlapping subgraphs). For finding Tutte paths in certain restrictions of 2-connected and 3-connected planar graphs, the related results in [22, 17] claim polynomial running times as well.

Our Results

Our motivation is two-fold. First, we want to make Tutte paths accessible to algorithms. We will show that Tutte paths can be computed in time $O(n^2)$. This has impact on almost all the applications using Tutte paths listed above.

For several of them, e.g. [26, 22, 27, 17], we immediately obtain polynomial-time algorithms where no efficient algorithms were published before. In addition, Tutte paths were also used in [7, 8] to show that every essentially 4-connected polyhedral graph contains a cycle of length proportional to n. As the existence proofs in this paper are constructive, our result directly implies a efficient (in fact, an $O(n^2)$-time) algorithm for the computation of these

cycles. Furthermore, [2] showed that every 3-connected planar graph having at most three 3-separators is Hamiltonian. If a 3-connected planar graph contains at most one 3-separator, our algorithm shows that a Hamiltonian cycle can be computed in $O(n^2)$ time, by using a suitable choice of the intermediate edge α.

Second, we aim for computing the strongest possible known variant of Tutte paths, encompassing the many incremental improvements on Tutte paths made over the years. We will therefore develop an algorithm for Sander's existence result [23], which was proven to be best possible in many aspects. Sanders result has also an immediate extension to connected planar graphs [20], which can be computed simply and efficiently from our result by using block-cut trees.

We will first give a decomposition that refines the original ones used for Theorems 1 and 2, and allows to decompose G into graphs that pairwise intersect in at most one edge. We then show that this small overlap does not prevent us from achieving a polynomial running time. All graphs will be simple. We proceed by showing how this decomposition can be computed efficiently in order to find the Tutte paths of Theorem 2. Our main result is hence the following, giving the first polynomial-time algorithm for computing Tutte paths.

▶ **Theorem 3.** *Let G be a 2-connected plane graph, $x \in V(G)$, $\alpha \in E(C_G)$ and $y \in V(G) - x$. Then a Tutte path of G from x to y through α can be computed in time $O(n^2)$.*

Section 3 presents the decomposition with small overlap that proves the existence of Tutte paths. On the way to our main result, we give full algorithmic counterparts of the approaches of Thomassen and Sanders; for example, we describe small overlap variants of Theorem 1 and of the *Three Edge Lemma* [26, 22], which was used in the purely existential result of Sanders [23] as a black box.

Our Techniques

We broadly follow the idea of [5] and construct a Tutte path that is based on certain 2-separators of the graphs constructed during our decomposition. This depends on many structural properties of the given graph. In [5], the necessary properties follow from the restriction to the class of internally 4-connected planar graphs, the restriction on the endpoints of the desired Tutte path, and the fact that the Tutte paths computed recursively are actually Hamiltonian. In contrast, here we give new insights into the much wilder structure of Tutte paths of 2-connected planar graphs, allow $x, y \notin C_G$, and hence extend this technique. We show that based on the prescribed vertices and edge, there are always unique non-interlacing 2-separators that are contained in every possible Tutte path of the given graph. We then use this set of 2-separators to iteratively construct a preliminary Tutte path and use this iterative procedure to avoid overlaps of more than one edge in the decomposition of the input graph.

2 Preliminaries

We assume familiarity with standard graph theoretic notations as in [6]. Let $deg(v)$ be the degree of a vertex v. We denote the subtraction of a graph H from a graph G by $G - H$, and the subtraction of a vertex or edge x from G by $G - x$.

A *k-separator* of a graph $G = (V, E)$ is a subset $S \subseteq V$ of size k such that $G - S$ is disconnected. A graph G is *k-connected* if $|V| > k$ and G contains no $(k-1)$-separator. For a path P and two vertices $x, y \in P$, let xPy be the smallest subpath of P that contains x and y. For a path P from x to y, let $inner(P) := V(P) - \{x, y\}$ be the set of its inner vertices. Paths that intersect pairwise at most at their endvertices are called *independent*.

A connected graph without a 1-separator is called a *block*. A *block of a graph* G is an inclusion-wise maximal subgraph that is a block. Every block of a graph is thus either 2-connected or has at most two vertices. It is well-known that the blocks of a graph partition its edge-set. A graph G is called a *chain of blocks* if it consists of blocks $B_1, B_2, \ldots, B_k$ such that $V(B_i) \cap V(B_{i+1})$, $1 \leq i < k$, are pairwise distinct 1-separators of G and G contains no other 1-separator. In other words, a chain of blocks is a graph, whose block-cut tree [14] is a path.

A *plane* graph is a planar embedding of a graph. Let C be a cycle of a plane graph G. For two vertices x, y of C, let xCy be the clockwise path from x to y in C. For a vertex x and an edge e of C, let xCe be the clockwise path in C from x to the endvertex of e such that $e \notin xCe$ (define eCx analogously). Let the subgraph of G *inside* C consist of $E(C)$ and all edges that intersect the open set inside C into which C divides the plane. For a plane graph G, let C_G be its outer face.

A central concept for Tutte paths is the notion of H-bridges (see [31] for some of their properties): For a subgraph H of a 2-connected plane graph G, an *H-bridge* of G is either an edge that has both endvertices in H but is not itself in H or a component K of $G - H$ together with all edges (and the endvertices of these edges) that join vertices of K with vertices of H. An H-bridge is called *trivial* if it is just one edge. A vertex of an H-bridge L is an *attachment* of L if it is in H, and an *internal* vertex of L otherwise. An *outer* H-bridge of G is an H-bridge that contains an edge of C_G.

A *Tutte path* (*Tutte cycle*) of a plane graph G is a path (a *cycle*) P of G such that every outer P-bridge of G has at most two attachments and every P-bridge at most three attachments. In most of the cases we consider, G will be 2-connected, so that every P-bridge has at least two attachments. For vertices x, y and an edge $\alpha \in C_G$, let an *x-α-y-path* be a Tutte path from x to y that contains α. An *x-y-path* is an *x-α-y-path* for an arbitrarily chosen edge $\alpha \in C_G$.

3 Decomposition with Small Overlap

After excluding several easy cases of the decomposition, we prove Thomassen's Theorem 1 constructively and then show how to use this for a proof of the Three Edge Lemma. The Three Edge Lemma, in turn, allows us to give a constructive proof of Sander's Theorem 2 in which only small overlaps occur in the induction. Due to space constraints, we have to omit this proof, it however derived from [23] in a similar way as Theorem 1 from [29].

We will use induction on the number of vertices. In all proofs about Tutte paths of this section, the induction base is a triangle, in which the desired Tutte path can be found trivially; thus, we will assume in these proofs by the induction hypothesis that graphs with fewer vertices contain Tutte paths. All graphs in the induction will be simple.

The following sections cover different cases of the induction steps of the three statements to prove, starting with some easy cases for which a decomposition into edge disjoint subgraphs was already given [29]. For the remainder of the article, let G be a simple plane 2-connected graph with outer face C_G and let $x \in V(G)$, $\alpha \in E(C_G)$ and $y \in V(G) - x$. If $\alpha = xy$, the desired path is simply xy; thus, assume $\alpha \neq xy$. Since G is 2-connected, C_G is a cycle.

3.1 The Easy Cases

We say that G is *decomposable into G_L and G_R* if it contains subgraphs G_L and G_R such that $G_L \cup G_R = G$, $V(G_L) \cap V(G_R) = \{c, d\}$, $x \in V(G_L)$, $\alpha \in E(G_R)$, $V(G_L) \neq \{x, c, d\}$ and $V(G_R) \neq \{c, d\}$ (or the analogous setting with y taking the role of x) (see Figure 1). In

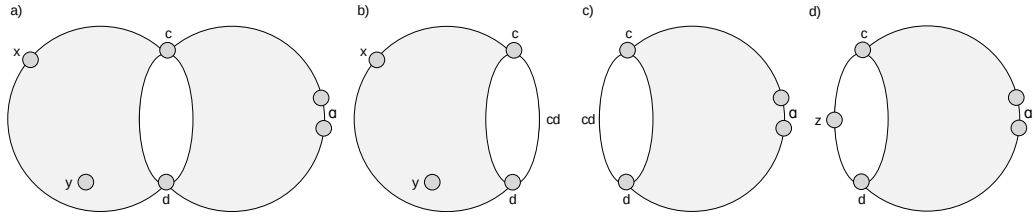


Figure 1 a) shows a graph G that is decomposable into G_L and G_R. The figures b) to d) show the graphs G'_L, G'_R and G^*_R (in this order) that are constructed to process G in [29] .

particular, $G_L \neq \{c, d\}$, even if $x \in \{c, d\}$. Hence $\{c, d\}$ is a 2-separator of G. There might exist multiple pairs (G_L, G_R) into which G is decomposable; we will always choose a pair that minimizes $|V(G_R)|$. Note that G_R intersects C_G (for example, in α), but G_L does not have to intersect C_G. In [29], it was shown that every decomposable graph G contains a Tutte path, without using recursion on overlapping subgraphs.

▶ **Lemma 4** ([29]). *If G is decomposable into G_L and G_R, then G contains an x-α-y-path.*

Even if G is not decomposable into G_L and G_R, G may contain other 2-separators $\{c, d\}$ that allow for a similar reduction as in Lemma 4 (for example, when modifying its prerequisites to satisfy $\{x, \alpha, y\} \subseteq G_R - \{c, d\}$).

▶ **Lemma 5** ([29]). *Let $\{c, d\}$ be a 2-separator of G and let J be a $\{c, d\}$-bridge of G having an internal vertex in C_G such that x, y and α are not in J. Then G contains an x-α-y-path.*

3.2 Proof of Theorem 1

We now prove that G contains a Tutte path from $x \in V(C_G)$ to $y \in V(G) - x$ through $\alpha \in E(C_G)$. For simplicity, if y is not in $V(C_G)$ but has degree two and both of its neighbors are in $V(C_G)$, then we change the embedding of G (and therefore C_G) such that y belongs to the outer face. If Lemma 4 or 5 can be applied, we obtain such a Tutte path directly, so assume their prerequisites are not met. Let l_α be the endvertex of α that appears first when we traverse C_G in clockwise order starting from x, and let r_α be the other endvertex of α. If $y \in xC_G l_\alpha$, we interchange x and y (this does not change l_α); hence, we have $y \notin xC_G l_\alpha$. If $y = r_\alpha$, we mirror the embedding such that y becomes l_α and proceed as in the previous case; hence, $y \notin xC_G r_\alpha$.

We define two paths P and Q in G, whose union will, step by step, be modified into a Tutte path of G. Let $Q := xC_G l_\alpha$ and let $H := G - V(Q)$; in particular, $y \notin Q$ and, if x is an endvertex of α, $Q = \{x\}$. Since G is not decomposable, we have $deg(r_\alpha) \geq 3$, as otherwise the neighborhood of r_α would be the 2-separator of such a decomposition. Since $deg(r_\alpha) \geq 3$, r_α is incident to an edge $e \notin C_G$ that shares a face with α. Let B_1 be the block of H that contains e. It is straight-forward to prove the following about B_1 (see Thomassen [29]), which shows that every vertex of C_G is either in Q or in B_1.

▶ **Lemma 6** ([29]). *B_1 contains $C_G - V(Q)$ and is the only block of H containing r_α.*

Consider a component A of H that does not contain B_1. Then the neighborhood of A in G is in Q and must contain a 2-separator of G due to planarity. Hence, either $y \in A$ and we can apply Lemma 4 or $y \notin A$ and we can apply Lemma 5. Since both contradicts our assumptions, H is connected and contains B_1 and y. Let K be the minimal plane chain of blocks $B_1, \ldots, B_l$ of H that contains B_1 and y (hence, $y \in B_l$). Let v_i be the intersection of B_i and B_{i+1} for $1 \leq i \leq l - 1$; in addition, we set $v_0 := r_\alpha$ and $v_l := y$.

Consider any $(K \cup C_G)$-bridge J. Since Lemma 5 cannot be applied, J has an attachment $v_J \in K$. Further, J cannot have two attachments in K, as this would contradict the maximality of the blocks in K. Let $C(J)$ be the shortest path in C_G that contains all vertices in $J \cap C_G$ and does not contain r_α as inner vertex (here, r_α serves as a *reference vertex* of C_G that ensures that the paths $C(J)$ are chosen consistently on C_G). Let l_J be the endvertex of $C(J)$ whose counterclockwise incident edge in C_G is not in $C(J)$ and let r_J be the other endvertex of $C(J)$.

3.2.1 Decomposing along Maximal 2-Separators

At this point we will deviate from the original proof of Theorem 1 in [29], which continues with an induction on every block of K that leads to overlapping subgraphs in a later step of the proof. Instead, we will show that a v_0-v_l-path P of K can be found iteratively such that the graphs in the induction have only small overlap.

For every block $B_i \neq B_1$ of K, we choose an arbitrary edge $\alpha_i = l_{\alpha_i} r_{\alpha_i}$ in C_{B_i}. In B_1 we choose α_1 such that α_1 is incident to the endvertex of $C_{B_1} \cap C_G$ that is not r_α. As done for G, we may assume for every B_i that l_{α_i} is the endvertex of α_i that is contained in $v_{i-1}C_{B_i}\alpha_i$ and that $v_i \notin v_{i-1}C_{B_i}r_{\alpha_i}$ and (by mirroring the planar embedding and interchanging v_i and v_{i-1} if necessary). However, unlike G, every B_i may satisfy the prerequisites of Lemmas 4 and 5. By the induction hypothesis of Theorem 1, B_i contains a v_{i-1}-α_i-v_i-path P_i. In [29], the outer P_i-bridges of B_i are not only being processed during this induction step, but also in a later induction step when modifying Q. We avoid such overlapping subgraphs by using a new iterative structural decomposition of B_i along certain 2-separators on C_{B_i}. This decomposition allows us to construct P_i iteratively such that the outer P_i-bridges of B_i are not part of the induction applied on B_i. Eventually, $P := \bigcup_{1 \leq i \leq l} P_i$ will be the desired v_0-v_l-path of K.

The outline is as follows. After explaining the basic split operation that is used by our decomposition, we give new insights into the structure of the Tutte paths P_i of the blocks B_i. These are used in Section 3.2.2 to define the iterative decomposition of every block B_i into a modified block $\eta(B_i)$, which will in turn allow to compute every P_i step-by-step. This gives the first part P of the desired Tutte path x-α-y of G. Subsequently, we will show how the remaining path Q can be modified to obtain the second part.

For a 2-separator $\{c, d\} \subseteq C_B$ of a block B, let B^+_{cd} be the $\{c, d\}$-bridge of B that contains cC_Bd and let B^-_{cd} be the union of all other $\{c, d\}$-bridges of B (note that B^+_{cd} contains the edge cd if and only if B^+_{cd} is trivial). For a 2-separator $\{c, d\} \subseteq C_B$, let *splitting off* B^+_{cd} (from B) be the operation that deletes all internal vertices of B^+_{cd} from B and adds the edge cd if cd does not already exist in B. Our decomposition proceeds by iteratively splitting off bridges B^+_{cd} from the blocks B_i of K for suitable 2-separators $\{c, d\} \subseteq C_{B_i}$ (we omit the subscript i in such bridges B^+_{cd}, as it is determined by c and d). The following lemma restricts these 2-separators to be contained in specific parts of the outer face.

▶ **Lemma 7.** *Let P' be a Tutte path of a block B such that P' contains an edge α' and two vertices $a, b \in C_B$. Then every outer P'-bridge J of B has both attachments in aC_Bb or both in bC_Ba. If additionally J is non-trivial and $P' \neq \alpha'$, the attachments of J form a 2-separator of B.*

Proof. Let e be an edge in $J \cap C_B$ and assume without loss of generalization that $e \in aC_Bb$. Let c and d be the last and first vertices of the paths aC_Be and eC_Bb, respectively, that are contained in P' (these exist, as a and b are in P'). Then J has attachments c and d and no further attachment, as P' is a Tutte path. This gives the first claim. For the second claim, let z be an internal vertex of J. Since $P' \neq \alpha'$, P' contains a third vertex $c \notin \{a, b\}$. As c is not contained in J, $\{c, d\}$ separates z and c and is thus a 2-separator of B. ◀

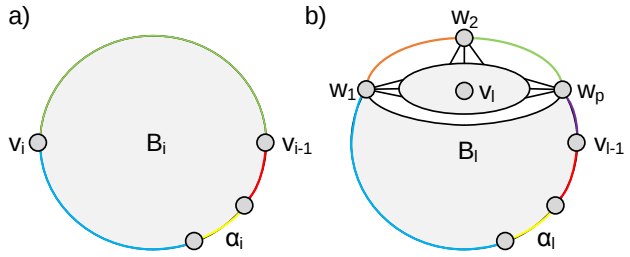


Figure 2 a) The boundary points and -parts of a block $B_i \neq B_l$. b) An instance in which the block B_l contains a 2-separator $\{w_1, w_p\}$ that splits off v_l.

For every block $B_i \neq B_l$ of K, let the *boundary points* of B_i be the vertices $v_{i-1}, l_{\alpha_i}, r_{\alpha_i}, v_i$ and let the *boundary parts* of B_i be the inclusion-wise maximal paths of C_{B_i} that do not contain any boundary point as inner vertex (see Figure 2a; note that boundary parts may be single vertices). Hence, every boundary point will be contained in any possible v_{i-1}-α_i-v_i-path P_i, and there are exactly four boundary parts, one of which is α_i. Now, if $P_i \neq \alpha_i$, applying Lemma 7 for all boundary points $a, b \in \{v_{i-1}, l_{\alpha_i}, r_{\alpha_i}, v_i\}$ and $\alpha' := \alpha_i$ implies that the two attachments of every outer non-trivial P_i-bridge of B_i form a 2-separator that is contained in one boundary part of B_i. For this reason, our decomposition will split off only 2-separators that are contained in boundary parts.

In principle, we will do the same for the block B_l. If $v_l \in C_{B_l}$, we define the boundary points of B_l just as before for $i < l$. However, B_l is special in the sense that v_l may not be in C_{B_l}. Then we have to ensure that we do not loose v_l when splitting off a 2-separator, as v_l is supposed to be contained in P_l (see Figure 2b). To this end, consider for $v_l \notin C_{B_l}$ the 2-separator $\{w_1, w_p\} \subseteq C_{B_l}$ of B_l such that $B^+_{w_1,w_p}$ contains v_l, the path $w_1 C_{B_l} w_p$ is contained in one of the paths in $\{v_{l-1} C_{B_l} \alpha_l, \alpha_l, \alpha_l C_{B_l} v_{l-1}\}$ and $w_1 C_{B_l} w_p$ is of minimal length if such a 2-separator exists. The restriction to these three parts of the boundary is again motivated by Lemma 7: If $P_l \neq \alpha_l$ and there is an outer non-trivial P_l-bridge of B_l, its two attachments are in P_l and thus we only have to split off 2-separators that are in one of these three paths to avoid these P_l-bridges in the induction. If the 2-separator $\{w_1, w_p\}$ exists, let $w_1, \dots, w_p$ be the $p \geq 2$ attachments of the $w_1 C_{B_l} w_p$-bridge of B_l that contains v_l, in the order of appearance in $w_1 C_{B_l} w_p$; otherwise, let for notational convenience $w_1 := \cdots := w_p := l_{\alpha_l}$. In the case $v_l \notin C_{B_l}$, let the *boundary points* of B_l be $v_{l-1}, l_{\alpha_l}, r_{\alpha_l}, w_1, \dots, w_p$ and let the *boundary parts* of B_l be the inclusion-wise maximal paths of C_{B_l} that do not contain any boundary point as inner vertex.

▶ **Lemma 8.** *If the 2-separator $\{w_1, w_p\}$ exists, it is unique and every v_{l-1}-α_l-v_l-path P_l of B_l contains the vertices $w_1, \dots, w_p$.*

Proof. Let $J \subset B^+_{w_1,w_p}$ be the $w_1 C_{B_l} w_p$-bridge of B_l that contains v_l and has attachments $w_1, \dots, w_p$. For the first claim, assume to the contrary that there is a 2-separator $\{w'_1, w'_{p'}\} \neq \{w_1, w_p\}$ of B_l having the same properties as $\{w_1, w_p\}$. By the connectivity of J and the property that restricts $\{w'_1, w'_{p'}\}$ to the three parts of the boundary of B_l, $\{w'_1, w'_{p'}\}$ may only split off a subgraph containing v_l if $w_1 C_{B_l} w_p \subset w'_1 C_{B_l} w'_{p'}$. This however contradicts the minimality of the length of $w'_1 C_{B_l} w'_{p'}$.

For the second claim, let P_l be any v_{l-1}-α_l-v_l-path of B_l. Assume to the contrary that $w_j \notin P_l$ for some $j \in \{1, \dots, p\}$. Then w_j is an internal vertex of an outer P_l-bridge J' of B_l. By Lemma 7, both attachments of J' are in C_{B_l}. However, since J contains a path from $w_j \notin P_l$ to $v_l \in P_l$ in which only w_j is in C_{B_l}, at least one attachment of J' is not in C_{B_l}, which gives a contradiction. ◀

Lemma 8 ensures that the boundary points of any B_i are contained in every Tutte path P_i of B_i. Every block $B_i \neq B_l$ has exactly four boundary parts and B_l has at least three boundary parts (three if $v_l \notin C_{B_l}$ and $\{w_1, w_p\}$ does not exist), some of which may have length zero. For every $1 \leq i \leq l$, the boundary parts of B_i partition C_{B_i}, and one of them consists of α_i. This implies in particular that B_i has at least two boundary parts of length at least one unless $B_i = \alpha_i$. We need some notation to break symmetries on boundary parts. For a boundary part Z of a block B, let $\{c, d\}^* \subseteq Z$ denote two elements c and d (vertices or edges) such that cC_Bd is contained in Z (this notation orders c and d consistently to the clockwise orientation of C_B); if cC_Bd is contained in some boundary part of B that is not specified, we just write $\{c, d\}^* \subseteq C_B$.

We now define which 2-separators are split off in our decomposition. Let a 2-separator $\{c, d\}^* \subseteq C_B$ of B be *maximal in a boundary part Z of B* if $\{c, d\} \subseteq Z$ and Z does not contain a 2-separator $\{c', d'\}$ of B such that $cC_Bd \subset c'C_Bd'$. Let a 2-separator $\{c, d\}^* \subseteq C_B$ of B be *maximal* if $\{c, d\}^*$ is maximal with respect to at least one boundary part of B. Hence, every maximal 2-separator is contained in a boundary part, and 2-separators that are contained in a boundary part are maximal if they are not properly "enclosed" by other 2-separators on the same boundary part.

Let two maximal 2-separators $\{c, d\}^*$ and $\{c', d'\}^*$ of B *interlace* if $\{c, d\} \cap \{c', d'\} = \emptyset$ and their vertices appear in the order c, c', d, d' or c', c, d', d on C_B (in particular, both 2-separators are contained in the same boundary part of B). In general, maximal 2-separators of a block B_i of K may interlace; for example, consider the two maximal 2-separators when B_i is a cycle on four vertices in which v_{i-1} and v_i are adjacent. However, the following lemma shows that such interlacing is only possible for very specific configurations.

▶ **Lemma 9.** *Let $\{c, d\}^*$ and $\{c', d'\}^*$ be interlacing 2-separators of B_i in a boundary part Z such that $c' \in cC_{B_i}d$ and at least one of them is maximal. Then $d'C_{B_i}c = v_{i-1}v_i = \alpha_i$.*

Proof. Since $\{c, d\}$ is a 2-separator, $B_i - \{c, d\}$ has at least two components. We argue that there are exactly two. Otherwise, $B_i - \{c, d\}$ has a component that contains the inner vertices of a path P' from c to d in $B_i - (C_{B_i} - \{c, d\})$. Then $B_i - \{c', d'\}$ has a component containing $(P' \cup C_{B_i}) - \{c', d'\}$ and no second component, as this would contain the inner vertices of a path from c' to d' in $B_i - ((P' \cup C_{B_i}) - \{c', d'\})$, which does not exist due to planarity. Since this contradicts that $\{c', d'\}$ is a 2-separator, we conclude that $B_i - \{c, d\}$, and by symmetry $B_i - \{c', d'\}$, have exactly two components.

By the same argument, $inner(cC_{B_i}d)$ and $inner(dC_{B_i}c)$ are contained in different components of $B_i - \{c, d\}$ and the same holds for $inner(c'C_{B_i}d')$ and $inner(d'C_{B_i}c')$ in $B_i - \{c', d'\}$. Hence, the component of $B_i - \{c, d'\}$ that contains $inner(cC_{B_i}d') \neq \emptyset$ does not intersect $inner(d'C_{B_i}c)$. If $inner(d'C_{B_i}c) \neq \emptyset$, this implies that $\{c, d'\} \subseteq Z$ is a 2-separator of B_i, which contradicts the maximality of $\{c, d\}$ or of $\{c', d'\}$. Hence, $inner(d'C_{B_i}c) = \emptyset$, which implies that $d'C_{B_i}c$ is an edge. As Z is not an edge, $d'C_{B_i}c = \alpha_i$. Since c and d' are the only boundary points of B_i, either $\{c, d'\} = \{v_{i-1}, v_i\}$ or $B_i = B_l$, $v_l \notin C_{B_l}$, $\{c, d'\} = \{v_{i-1}, w_2\}$, $v_{i-1} = w_1$ and $w_2 = w_p$. However, the latter case is impossible, as then $\{c, d'\}$ would be a 2-separator that separates $inner(cC_{B_i}d') \neq \emptyset$ and v_l, which contradicts the maximality of $\{c, d\}$ or of $\{c', d'\}$. This gives the claim. ◀

If two maximal 2-separators interlace, Lemma 9 thus ensures that these two are the only maximal 2-separators that may contain v_{i-1} and v_i, respectively. This gives the following direct corollary.

▶ **Corollary 10.** *Every block of K has at most two maximal 2-separators that interlace.*

Note that any boundary part may nevertheless contain arbitrarily many (pairwise non-interlacing) maximal 2-separators. The next lemma strengthens Lemma 7.

▶ **Lemma 11.** *Let P_i be a v_{i-1}-α_i-v_i-path of B_i. Let J be a non-trivial outer P_i-bridge of B_i and let e be an edge in $J \cap C_{B_i}$. Then the attachments of J are contained in the boundary part of B_i that contains e.*

Proof. Let c and d be the attachments of J such that $e \in cC_{B_i}d$ and let Z be the boundary part of B_i that contains e. If $P_i = \alpha_i$, $v_{i-1} = l_{\alpha_i}$ and $v_i = r_{\alpha_i}$ are the only boundary points of B_i. Then c and d are the endvertices of $Z = v_iC_{B_i}v_{i-1} \ni e$, which gives the claim.

Otherwise, let $P_i \neq \alpha_i$. By applying Lemma 7 with $a = l_{\alpha_i}$ and $b = r_{\alpha_i}$, $\{c, d\}$ is a 2-separator of B_i that is contained in C_{B_i}. By definition of $w_1, \dots, w_p$, there are at least three independent paths between every two of these vertices in B_i; thus, $\{c, d\}$ does not separate two vertices of $\{w_1, \dots, w_p\}$. Since all other possible boundary points $(v_{i-1}, l_{\alpha_i}, r_{\alpha_i}, v_i)$ are contained in P_i, applying Lemma 7 on these implies that $\{c, d\}$ does not separate two vertices of these remaining boundary points. Hence, if $\{c, d\} \not\subseteq Z$, we have $B_i = B_l$ and $v_l \notin C_{B_l}$ such that $\{c, d\}$ separates $\{w_1, \dots, w_p\}$ from the remaining boundary points. Since the P_i-bridge J does not contain $\alpha_l \in P_i$, $cC_{B_l}d \subseteq J$ contains $\{w_1, \dots, w_p\}$, but $inner(cC_{B_l}d)$ does not contain any other boundary point. As $v_l \in P_i$, at least one of $\{w_1, w_p\}$ must be in P_i, say w_p by symmetry. Then $d = w_p$, as $w_p \in P_i$ cannot be an internal vertex of J. Now, in both cases $p = 2$ (which implies $c \neq w_1$, as $\{c, d\} \not\subseteq Z = w_1C_{B_l}w_2$) and $p \geq 3$, J contains the edge of P_i that is incident to v_l. As this contradicts that J is a P_i-bridge, we conclude $\{c, d\} \subseteq Z$. ◀

Now we relate non-trivial outer P_i-bridges of B_i to maximal 2-separators of B_i. In the next section, we will use this lemma as a fundamental tool for a decomposition into subgraphs having only small overlaps, which will eventually construct P.

▶ **Lemma 12.** *Let P_i be a v_{i-1}-α_i-v_i-path of B_i such that $P_i \neq \alpha_i$. Then the maximal 2-separators of B_i are contained in P_i and do not interlace pairwise. If J is a non-trivial outer P_i-bridge of B_i, there is a maximal 2-separator $\{c, d\}^*$ of B_i such that $J \subseteq B^+_{cd}$.*

3.2.2 Construction of P

We do not know P_i in advance. However, Lemma 12 ensures under the condition $P_i \neq \alpha_i$ that we can split off every non-trivial outer bridge J of P_i by a maximal 2-separator, no matter how P_i looks like. This allows us to construct P_i iteratively by decomposing B_i along its maximal 2-separators. Since maximal 2-separators only depend on the graph B_i (in contrast to the paths P_i, which depend for example on the $K \cup C_G$-bridges), we can access them without knowing P_i itself. We now give the details of such a decomposition.

▶ **Definition 13.** For every $1 \leq i \leq l$, let $\eta(B_i)$ be α_i if $\alpha_i = v_{i-1}v_i$ and otherwise the graph obtained from B_i as follows: For every maximal 2-separator $\{c, d\}^*$ of B_i, split off B^+_{cd}. Moreover, let $\eta(K) := \eta(B_1) \cup \dots \cup \eta(B_l)$.

If $\alpha_i \neq v_{i-1}v_i$, α_i cannot be a v_{i-1}-α_i-v_i-path of B_i; hence, the maximal 2-separators of K that were split in this definition do not interlace due to Lemma 12. This implies that the order of the performed splits is irrelevant. In any case, we have $V(C_{\eta(B_i)}) \subseteq V(C_{B_i})$ and the only 2-separators of $\eta(B_i)$ must be contained in some boundary part of B_i, as there would have been another split otherwise. See Figure 3 for an illustration of $\eta(B_l)$. The following lemma highlights two important properties of every $\eta(B_i)$.

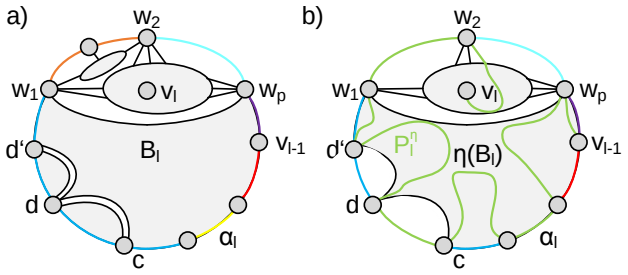


Figure 3 a) A block B_l with boundary points $v_{l-1}, l_{\alpha_l}, r_{\alpha_l}, w_1, \ldots, w_3$ that has two maximal 2-separators on the same boundary part. b) The graph $\eta(B_l)$.

▶ **Lemma 14.** *Every $\eta(B_i)$ is a block. Let P_i^η be a v_{i-1}-α_i-v_i-path of some $\eta(B_i)$ such that $P_i^\eta \neq \alpha_i$. Then every outer P_i^η-bridge of $\eta(B_i)$ is trivial.*

The next lemma shows how we can construct a Tutte path P of K iteratively using maximal 2-separators. We will provide the details of an efficient implementation in Section 4.

▶ **Lemma 15** (Construction of P)**.** *For every $1 \leq i \leq l$, a v_{i-1}-α_i-v_i-path P_i of B_i can be constructed such that no non-trivial outer P_i-bridge of B_i is part of an inductive call of Theorem 1.*

Proof. The proof proceeds by induction on the number of vertices in B_i. If B_i is just an edge or a triangle, the claim follows directly. For the induction step, we therefore assume that B_i contains at least four vertices. If $\alpha_i = v_{i-1}v_i$, we set $P_i := \alpha_i$, so assume $\alpha_i \neq v_{i-1}v_i$. In particular, $\eta(B_i) \neq \alpha_i$ and α_i is no v_{i-1}-α_i-v_i-path of $\eta(B_i)$. As $|V(\eta(B_i))| < n$, we may apply an inductive call of Theorem 1 to $\eta(B_i)$, which returns a v_{i-1}-α_i-v_i-path $P_i^\eta \neq \alpha_i$ of $\eta(B_i)$. This does not violate the claim, since $\eta(B_i)$ does not contain any non-trivial outer P_i^η-bridge by Lemma 14.

Now we extend P_i^η iteratively to the desired v_{i-1}-α_i-v_i-path P_i of B_i by restoring the subgraphs that were split off along maximal 2-separators one by one. For every edge $cd \in C_{\eta(B_i)}$ such that $\{c, d\}^*$ is a maximal 2-separator of B_i (in arbitrary order), we distinguish the following two cases: If $cd \notin P_i^\eta$, we do not modify P_i^η, as in B_i the subgraph B_{cd}^+ will be a valid outer bridge. If otherwise $cd \in P_i^\eta$, we consider the subgraph B_{cd}^+ of B_i. Clearly, $B := B_{cd}^+ \cup \{cd\}$ is a block. Define that the *boundary points* of B are c, d and the two endpoints of some arbitrary edge $\alpha_B \neq cd$ in C_B. This introduces the boundary parts of B in the standard way, and hence defines $\eta(B)$. Note that B may contain several maximal 2-separators in cC_Bd that in B_i were suppressed by $\{c, d\}^*$, as $\{c, d\}^*$ is not a 2-separator of B. In consistency with Lemma 12, which ensures that no two maximal 2-separators of B_i interlace, we have to ensure that no two maximal 2-separators of B interlace in our case $\alpha_i \neq v_{i-1}v_i$, as otherwise $\eta(B)$ would be ill-defined. This is however implied by Lemma 9, as $\alpha_B \neq cd$. Since $|V(\eta(B))| < |V(B_i)|$, a c-α_B-d-path P_B of B can be constructed such that no non-trivial outer P_B-bridge of B is part of an inductive call of Theorem 1. Since $\alpha_B \neq cd$, P_B does not contain cd. We now replace the edge cd in P_i^η by P_B. This gives the desired path P_i after having restored all subgraphs B_{cd}^+. ◀

Applying Lemma 15 on all blocks of K and taking the union of the resulting paths gives P. In the next step, we will modify Q such that $P \cup \{\alpha\} \cup Q$ becomes the desired Tutte path of G. By Lemma 15, no non-trivial outer P-bridge of K was part of any inductive call of Theorem 1 so far, which allows us to use these bridges inductively for the following modification of Q (the existence proof in [29] used these arbitrarily large bridges in inductive calls for both constructing P and modifying Q).

3.2.3 Modification of Q

We show how to modify Q such that $P \cup \{\alpha\} \cup Q$ is an x-α-y-path of G. To this end, consider a $(P \cup \{\alpha\} \cup Q)$-bridge J of G. Since Lemma 5 cannot be applied, J does not have all of its attachments in Q. On the other hand, if J has all of its attachments in $P \subseteq K$, $J \subseteq K$ follows from the maximality of blocks and therefore J satisfies all conditions for a Tutte path of G. Hence, it suffices to consider $(P \cup \{\alpha\} \cup Q)$-bridges that have attachments in both P and Q. The following lemma showcases some of their properties.

▶ **Lemma 16.** *Let J be a $(P \cup \{\alpha\} \cup Q)$-bridge of G that has an attachment in P. Then $J \cap K$ is either exactly one vertex in P or exactly one non-trivial outer P-bridge L of K. In particular, J has at most two attachments in P.*

Let J be a $(P \cup \{\alpha\} \cup Q)$-bridge of G that has attachments in both P and Q and recall that $C(J) = l_J C_G r_J$. Because Lemma 5 is not applicable to G, there is no other $(P \cup \{\alpha\} \cup Q)$-bridge than J that intersects $(J \cup C(J)) - P - \{l_J, r_J\}$; in other words, $J \cup C(J)$ is everything that is enclosed by the attachments of J in G. In order to obtain the Tutte path of Theorem 1, we will thus replace the subpath $C(J)$ with a path $Q_J \subseteq (J \cup C(J)) - P$ from l_J to r_J such that any $(Q_J \cup P)$-bridge of G that intersects $(J \cup C(J)) - P - \{l_J, r_J\}$ has at most three attachments and at most two if it contains an edge of C_G. Since l_J and r_J are contained in Q, no other $(P \cup \{\alpha\} \cup Q)$-bridge of G than J is affected by this "local" replacement, which proves its sufficiency for obtaining the desired Tutte path.

We next show how to obtain Q_J. If $C(J)$ is a single vertex, we do not need to modify Q at all (hence, $Q_J := C(J)$), as then $J \cup C(J)$ does not contain an edge of C_G and has at most three attachments in total (one in Q and at most two in P by Lemma 16). If $C(J)$ is not a single vertex, we have the following lemma.

▶ **Lemma 17** ([29, 4])**.** *Let J be a $(P \cup \{\alpha\} \cup Q)$-bridge of G that has an attachment in P and at least two in Q. Then $(J \cup C(J)) - P$ contains a path Q_J from l_J to r_J such that any $(Q_J \cup P)$-bridge of G that intersects $(J \cup C(J)) - P - \{l_J, r_J\}$ has at most three attachments and at most two if it contains an edge of C_G.*

By Lemma 16, any $(P \cup \{\alpha\} \cup Q)$-bridge J of G intersects K in at most one non-trivial P-bridge of K having attachments c and d. By Lemma 15, this non-trivial P-bridge was never part of an inductive call of Theorem 1 before (in fact, at most its edge cd was). Replacing $C(J)$ with Q_J for every such J, as described in Lemma 17 and before, therefore concludes the constructive proof of Theorem 1.

4 A Quadratic Time Algorithm

In this section, we give an algorithm based on the decomposition shown in Section 3 (see Algorithm 1). It is well known that there are algorithms that compute the blocks of a graph and the block-cut tree of G in linear time, see [25] for a very simple one. Using this on $G - Q$, we can compute the blocks $B_1, \dots, B_l$ of K in time $O(n)$.

We now check if Lemma 4 or 5 is applicable at least once to G; if so, we stop and apply the construction of either Lemma 4 or 5. Checking applicability involves the computation of special 2-separators $\{c, d\}$ of G that are in C_G (e.g., we did assume minimality of $|V(G_R)|$ in Lemma 4). In order to find such a $\{c, d\}$ in time $O(n)$, we first compute the *weak dual* G^* of G, which is obtained from the dual of G by deleting its outer face vertex, and note that such pairs $\{c, d\}$ are exactly contained in the faces that correspond to 1-separators of G^*. Once more, these faces can be found by the block-cut tree of G^* in time $O(n)$ using the above

Algorithm 1 TPATH(G, x, α, y) ▷ method, running time without induction

if G is a triangle or $\alpha = xy$ **then return** the trivial x-α-y path of G ▷ $O(1)$
if Lemma 4 or 5 is applicable at least once to G **then** ▷ weak dual block-cut tree, $O(n)$
    apply TPATH on G_L and G_R as described and **return** the resulting path ▷ $O(1)$
if there is a 2-separator $\{c, d\} \in C_G$ of G **then**
    do simple case 2
Compute the minimal plane chain K of blocks of G ▷ block-cut tree of $G - Q$, $O(n)$
Compute $\eta(K)$ ▷ dyn. progr. on weak dual block-cut tree, $O(n)$
Compute P by the induction of Lemma 15 ▷ dyn. prog. precomputes all possible B^+_{cd}, $O(n)$
Modify Q by the induction of Lemma 17 ▷ traversing outer faces of bridges, $O(n)$
return $P \cup \{\alpha\} \cup Q$

algorithm. Since the block-cut tree is a tree, we can perform dynamic programming on all these 1-separators bottom-up the tree in linear total time, in order to find one desired $\{c, d\}$ that satisfies the respective constraints (e.g. minimizing $|V(G_R)|$, or separating x and α).

Now we compute $\eta(K)$. Since the boundary points of every B_i are known from K, all *maximal* 2-separators can be computed in time $O(n)$ by dynamic programming as described above. We compute in fact the nested tree structure of all 2-separators on boundary parts due to Lemma 12, on which we then apply the induction described in Lemma 15. Hence, no non-trivial outer P-bridge of K is touched in the induction, which allows to modify Q along the induction of Lemma 17.

In our decomposition, every inductive call is invoked on a graph having less vertices than the current graph. The key insight is now to show a good bound on the total number of inductive calls to Theorem 2. In order to obtain good upper bounds, we will restrict the choice of α_i for every block B_i of K such that α_i is an edge of $C_{B_i} - v_{i-1}v_i$. This prevents several situations in which the recursion stops because of the case $\alpha = xy$, which would unease the following arguments. The next lemma shows that only $O(n)$ inductive calls are performed. Its argument is, similarly to one in [5], based on a subtle summation of the Tutte path differences that occur in the recursion tree.

▶ **Lemma 18.** *The number of inductive calls for TPATH(G, x, α, y) is at most $2n - 3$.*

Hence, Algorithm 1 has overall running time $O(n^2)$, which proves our main Theorem 3.

▶ **Corollary 19.** *Let G be a 2-connected plane graph and let α, β, γ be edges of C_G. Then a Tutte cycle of G that contains α, β and γ can be computed in time $O(n^2)$.*

References

1 T. Asano, S. Kikuchi, and N. Saito. A linear algorithm for finding Hamiltonian cycles in 4-connected maximal planar graphs. *Discrete Applied Mathematics*, 7(1):1–15, 1984.
2 G. Brinkmann and C. T. Zamfirescu. A Strengthening of a Theorem of Tutte on Hamiltonicity of Polyhedra. *ArXiv e-prints*, 2016. arXiv:1606.01693.
3 R. Brunet, M. N. Ellingham, Z. C. Gao, A. Metzlar, and R. B. Richter. Spanning planar subgraphs of graphs in the torus and Klein bottle. *Journal of Combinatorial Theory, Series B*, 65(1):7–22, 1995. doi:10.1006/jctb.1995.1041.
4 N. Chiba and T. Nishizeki. A theorem on paths in planar graphs. *Journal of Graph Theory*, 10(4):449–450, 1986.

5 N. Chiba and T. Nishizeki. The Hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. *Journal of Algorithms*, 10(2):187–211, 1989.

6 R. Diestel. *Graph Theory*. Springer, fourth edition, 2010.

7 I. Fabrici, J. Harant, and S. Jendrol. On longest cycles in essentially 4-connected planar graphs. *Discussiones Mathematicae Graph Theory*, 36:565–575, 2016.

8 I. Fabrici, J. Harant, S. Mohr, and J. M. Schmidt. Longer cycles in essentially 4-connected planar graphs. *Discussiones Mathematicae Graph Theory*, to appear.

9 Z. Gao and R. B. Richter. 2-Walks in circuit graphs. *Journal of Combinatorial Theory, Series B*, 62(2):259–267, 1994.

10 Z. Gao, R. B. Richter, and X. Yu. 2-Walks in 3-connected planar graphs. *Australasian Journal of Combinatorics*, 11:117–122, 1995.

11 Z. Gao, R. B. Richter, and X. Yu. Erratum to: 2-Walks in 3-connected planar graphs. *Australasian Journal of Combinatorics*, 36:315–316, 2006.

12 D. Gouyou-Beauchamps. The Hamiltonian circuit problem is polynomial for 4-connected planar graphs. *SIAM Journal on Computing*, 11(3):529–539, 1982.

13 J. Harant and S. Senitsch. A generalization of Tutte's theorem on Hamiltonian cycles in planar graphs. *Discrete Mathematics*, 309(15):4949–4951, 2009. doi:10.1016/j.disc.2008.04.038.

14 F. Harary and G. Prins. The block-cutpoint-tree of a graph. *Publ. Math. Debrecen*, 13:103–107, 1966.

15 B. Jackson and N. C. Wormald. k-Walks of graphs. *Australasian Journal of Combinatorics*, 2:135–146, 1990.

16 B. Jackson and X. Yu. Hamilton cycles in plane triangulations. *Journal of Graph Theory*, 41(2):138–150, 2002.

17 K. Kawarabayashi and K. Ozeki. 4-connected projective-planar graphs are Hamiltonian-connected. *Journal of Combinatorial Theory, Series B*, 112:36–69, 2015.

18 Tom Leighton and Ankur Moitra. Some results on greedy embeddings in metric spaces. *Discrete & Computational Geometry*, 44(3):686–705, Oct 2010. doi:10.1007/s00454-009-9227-6.

19 A. Nakamoto, Y. Oda, and K. Ota. 3-trees with few vertices of degree 3 in circuit graphs. *Discrete Mathematics*, 309(4):666–672, 2009. doi:10.1016/j.disc.2008.01.002.

20 K. Ozeki. A shorter proof of Thomassen's theorem on Tutte paths in plane graphs. *SUT Journal of Mathematics*, 50:417–425, 2014. URL: http://comb.math.keio.ac.jp/ozeki.

21 K. Ozeki and P. Vrána. 2-edge-Hamiltonian-connectedness of 4-connected plane graphs. *European Journal of Combinatorics*, 35:432–448, 2014. doi:10.1016/j.ejc.2013.06.033.

22 D. P. Sanders. On Hamilton cycles in certain planar graphs. *Journal of Graph Theory*, 21(1):43–50, 1996.

23 D. P. Sanders. On paths in planar graphs. *Journal of Graph Theory*, 24(4):341–345, 1997. doi:10.1002/(SICI)1097-0118(199704)24:4<341::AID-JGT6>3.0.CO;2-O.

24 A. Schmid and J. M. Schmidt. Computing 2-walks in polynomial time. In *Proceedings of the 32nd Symposium on Theoretical Aspects of Computer Science (STACS'15)*, pages 676–688, 2015.

25 J. M. Schmidt. A simple test on 2-vertex- and 2-edge-connectivity. *Information Processing Letters*, 113(7):241–244, 2013. doi:10.1016/j.ipl.2013.01.016.

26 R. Thomas and X. Yu. 4-connected projective-planar graphs are Hamiltonian. *Journal of Combinatorial Theory, Series B*, 62(1):114–132, 1994.

27 R. Thomas and X. Yu. Five-connected toroidal graphs are Hamiltonian. *Journal of Combinatorial Theory, Series B*, 69(1):79–96, 1997.

28 R. Thomas, X. Yu, and W. Zang. Hamilton paths in toroidal graphs. *Journal of Combinatorial Theory, Series B*, 94(2):214–236, 2005.

29 C. Thomassen. A theorem on paths in planar graphs. *Journal of Graph Theory*, 7(2):169–176, 1983.

30 W. T. Tutte. A theorem on planar graphs. *Transactions of the American Mathematical Society*, 82:99–116, 1956.

31 W. T. Tutte. Bridges and Hamiltonian circuits in planar graphs. *Aequationes Mathematicae*, 15(1):1–33, 1977.

32 H. Whitney. A theorem on graphs. *Annals of Mathematics*, 32(2):378–390, 1931.

33 X. Yu. Disjoint paths, planarizing cycles, and spanning walks. *Transactions of the American Mathematical Society*, 349(4):1333–1358, 1997.

A New Approximation Guarantee for Monotone Submodular Function Maximization via Discrete Convexity

Tasuku Soma[1]
The University of Tokyo, Tokyo, Japan
tasuku_soma@mist.i.u-tokyo.ac.jp
https://orcid.org/0000-0001-9519-2487

Yuichi Yoshida[2]
National Institute of Informatics, Preferred Infrastructure, Tokyo, Japan
yyoshida@nii.ac.jp
https://orcid.org/0000-0001-8919-8479

Abstract

In monotone submodular function maximization, approximation guarantees based on the curvature of the objective function have been extensively studied in the literature. However, the notion of curvature is often pessimistic, and we rarely obtain improved approximation guarantees, even for very simple objective functions.

In this paper, we provide a novel approximation guarantee by extracting an $\mathrm{M}^\natural$-concave function $h : 2^E \to \mathbb{R}_+$, a notion in discrete convex analysis, from the objective function $f : 2^E \to \mathbb{R}_+$. We introduce a novel notion called the $\mathrm{M}^\natural$-concave curvature of a given set function f, which measures how much f deviates from an $\mathrm{M}^\natural$-concave function, and show that we can obtain a $(1 - \gamma/e - \epsilon)$-approximation to the problem of maximizing f under a cardinality constraint in polynomial time, where γ is the value of the $\mathrm{M}^\natural$-concave curvature and $\epsilon > 0$ is an arbitrary constant. Then, we show that we can obtain nontrivial approximation guarantees for various problems by applying the proposed algorithm.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization

Keywords and phrases Submodular Function, Approximation Algorithm, Discrete Convex Analysis

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.99

Related Version A full version of the paper is available at `https://arxiv.org/abs/1709.02910`.

Acknowledgements The authors thank Yuni Iwamasa for pointing out a reference [8] on ultrametric fitting. The authors also thank Jan Vondrák for telling us a reference [21].

1 Introduction

A set function $f : 2^E \to \mathbb{R}$ is called *monotone* if $f(X) \leq f(Y)$ for any $X \subseteq Y \subseteq E$ and called *submodular* if $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$ for any $X, Y \subseteq E$. In monotone submodular function maximization under a cardinality constraint, given a nonnegative

[1] T. S. was supported by JSPS Grant-in-Aid for Young Scientists (Start-up) Grant Number JP16H06676.
[2] Y. Y. was supported by JSPS KAKENHI Grant Number JP17H04676.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 99; pp. 99:1–99:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

monotone submodular function $f : 2^E \to \mathbb{R}_+$ and $k \in \mathbb{Z}_+$, we want to compute a set $S \subseteq E$ with $|S| \leq k$ that maximizes $f(S)$. This simple problem includes various optimization problems in theory and practice, such as facility location, combinatorial auctions [7, 14], viral marketing in social networks [11], and sensor placement [13, 12].

Although monotone submodular function maximization is an NP-hard problem, the greedy algorithm achieves a $(1 - 1/e)$-approximation, and this approximation ratio is known to be tight [20]. In practice, however, the greedy algorithm exhibits an approximation ratio much better than $1 - 1/e$, sometimes close to one [13]. This gap between the theoretical guarantee and the practical performance in "real-world" instances has been a major mystery in submodular function maximization. The first attempt to explain this gap goes back to Conforti and Cornuéjols [6]. They introduced a parameter called the (total) *curvature* of a monotone submodular function, with which we can derive a tighter approximation ratio.

▶ **Definition 1** (Curvature)**.** The curvature c of a monotone submodular function $f : 2^E \to \mathbb{R}_+$ is

$$c := 1 - \min_{i \in E} \frac{f(i \mid E - i)}{f(i)}.$$

Here, $f(i)$ and $f(i \mid E - i)$ are shorthand for $f(\{i\})$ and $f(E) - f(E - i)$, respectively. We note that $f(i \mid E - i) \leq f(i)$ holds from the submodularity of f; hence, $c \in [0, 1]$ holds. Roughly speaking, the curvature of a function measures how close it is to a modular function, where a set function $g : 2^E \to \mathbb{R}$ is called *modular* if $g(X) + g(Y) = g(X \cap Y) + g(X \cup Y)$ for every $X, Y \subseteq E$. Indeed, we can easily observe that $c = 0$ if and only if f is modular.

It is shown in [6] that the greedy method achieves a $(1 - e^{-c})/c$-approximation, which is at least $1 - 1/e$ and tends to 1 as $c \to 0$. This approximation has recently improved to $1 - c/e - \epsilon$ for any $\epsilon > 0$ using a more sophisticated algorithm by Sviridenko, Vondrák, and Ward [24]. Since the curvature is easy to analyze, it has been shown that we can obtain refined approximation guarantees for various settings by exploiting the curvature [10, 1].

However, we might ask *does the curvature explain the gap completely?* The answer seems to be negative. The concept of curvature is still unsatisfactory because monotone submodular functions in practical applications are often far from modular functions, and the curvature does not explain why we can obtain high approximation ratios. For instance, let us consider a very simple function $f(X) = \sqrt{|X|}$. Since this function has a curvature of $1 - O(1/\sqrt{n})$ for $n = |E|$, the approximation guarantee is (roughly) $1 - 1/e$, while the greedy algorithm obviously finds an optimal solution!

1.1 Our contributions

To narrow the gap discussed above, in this work we consider a larger and richer class of functions that are easy to maximize. For such a class of functions, we exploit $\mathrm{M}^\natural$-concave functions, introduced in the discrete convex analysis literature [18]. A set function $f : 2^E \to \mathbb{R}$ is called *$M^\natural$-concave*[3] if, for any $X, Y \subseteq E$ and $i \in X \setminus Y$, either (i) $f(X) + f(Y) \leq f(X - i) + f(Y + i)$ or (ii) there exists $j \in Y \setminus X$ such that $f(X) + f(Y) \leq f(X - i + j) + f(Y + i - j)$. Intuitively speaking, we can increase the sum $f(X) + f(Y)$ by making X and Y closer, which resembles concave functions. We note that an $\mathrm{M}^\natural$-concave function is submodular, and an $\mathrm{M}^\natural$-concave function can be maximized in polynomial time with the greedy algorithm (see,

[3] For a set function, $\mathrm{M}^\natural$-concave functions are essentially equivalent to *valuated matroids*.

e.g., [16]). Hence, $\mathrm{M}^\natural$-concave functions can be regarded as a class of "easy" submodular functions.

Now, we start with the following observation. For a monotone submodular function $f : 2^E \to \mathbb{R}_+$, we define a modular function h as $h(X) := \sum_{i \in X} f(i \mid E - i)$. Then, f can be decomposed as $f = g + h$, where g is another monotone submodular function. One can show that $h(X) \geq (1-c)f(X)$ for any $X \subseteq E$, where c is the curvature of f. The modular function h can be regarded as an "easy" part of f and g as a "difficult" part, and the curvature measures the contribution of the modular part h to the entire function f. Observing that any modular function is $\mathrm{M}^\natural$-concave, we can think of decompositions into a monotone submodular function and an $\mathrm{M}^\natural$-concave function, which gives the following definition:

▶ **Definition 2** ($\mathrm{M}^\natural$-concave curvature)**.** Suppose that a monotone submodular function $f : 2^E \to \mathbb{R}_+$ is decomposed as $g + h$, where $g : 2^E \to \mathbb{R}_+$ is monotone submodular, and $h : 2^E \to \mathbb{R}_+$ is $\mathrm{M}^\natural$-concave. Then, the *$M^\natural$-concave curvature* $\gamma(g, h)$ of f is the minimum value of γ such that $h(X) \geq (1 - \gamma)f(X)$ for every $X \subseteq E$, or equivalently

$$\gamma(g, h) = 1 - \min_{X \subseteq E} \frac{h(X)}{f(X)} = \max_{X \subseteq E} \frac{g(X)}{f(X)},$$

where we conventionally assume that $\frac{0}{0} := \infty$.

We abbreviate $\gamma(g, h)$ by γ if g and h are clear from the context. We note that the $\mathrm{M}^\natural$-concave curvature is defined for a decomposition $f = g + h$, whereas the standard curvature is defined for the function f itself. By the argument above, we can always find a decomposition $f = g + h$ such that the $\mathrm{M}^\natural$-concave curvature is no greater than the standard curvature.

Our main contribution is the following generalization of the results of [24, 16] in terms of the $\mathrm{M}^\natural$-concave curvature.

▶ **Theorem 3.** *Let $f : 2^E \to \mathbb{R}_+$ be a monotone submodular function that can be decomposed as $f = g + h$, where g is monotone submodular, and h is $M^\natural$-concave. Assume that we have value oracles of g and h. Then, for an integer $k \in \mathbb{Z}_+$ and a constant $\epsilon > 0$, we can find a random subset X of size k such that $\mathbf{E}[f(X)] \geq (1 - \gamma(g, h)/e - \epsilon)f(O)$ in polynomial time, where O is an optimal solution.*

Applications. $\mathrm{M}^\natural$-concave functions include many nontrivial functions such as (weighted) matroid rank functions and laminar concave functions, which are far from modular functions. Our result immediately implies that if the significant part of $f = g + h$ is due to the $\mathrm{M}^\natural$-concave part h, we can obtain a better approximation ratio, although it might not be straightforward to find such a decomposition of f into g and h even if f is a compactly represented function. In Section 4, we provide a general algorithm for finding a decomposition from the value oracle of f. This algorithm always finds a decomposition that is at least as good as the decomposition via the standard curvature. In Section 5, we provide problem-specific decompositions for several problems such as facility location, which yield improved approximation guarantees.

1.2 Proof technique

Our main result (Theorem 3) is proved by modifying the *continuous greedy algorithm*. The continuous greedy algorithm is a powerful and flexible framework for submodular function maximization, which has been applied to a matroid constraint, a knapsack constraint, and even combinations of various constraints [2, 4, 5]. At a high level, the continuous greedy

algorithm generates a sequence $\boldsymbol{x}(t)$ in the convex hull of feasible solutions for $t \in [0,1]$ by the following differential equation: $\frac{d\boldsymbol{x}}{dt} = \boldsymbol{v}(t)$ and $\boldsymbol{x}(0) = \mathbf{0}$, where $\boldsymbol{v}$ is a prescribed velocity vector. Then, we round the final point $\boldsymbol{x}(1)$ into a feasible solution X via a rounding algorithm. The previous result of [24] relies on the property that for a modular function $h = \boldsymbol{w}(X)$, if $\boldsymbol{w}^\top \boldsymbol{v}(t) \geq \alpha$ holds for $t \in [0,1]$, then $\boldsymbol{w}^\top \boldsymbol{x}(1) \geq \alpha$. This property enables us to find a fractional solution $\boldsymbol{x}(1)$ that simultaneously optimizes h and g; that is, $\boldsymbol{x}(1)$ achieves the optimal value for the modular part h and a $(1-1/e)$-approximation for the remaining monotone submodular part g.

Continuous greedy algorithm with a multilinear extension and concave closure. To run the continuous greedy algorithm, we require continuous extensions of set functions. For submodular functions, the *multilinear extension* is typically used. In addition, a modular function is trivially extendable. However, for $\mathrm{M}^\natural$-concave functions, it is nontrivial to choose a continuous extension because if we just use the multilinear extension, we end up with a $(1-1/e)$-approximation. To remedy this, we use a different continuous extension, namely, the *concave closure*. The concave closure is difficult to evaluate in general, but for $\mathrm{M}^\natural$-concave functions, we can evaluate their concave closures in polynomial time thanks to the results of discrete convex analysis. This property enables us to run a modified continuous greedy algorithm with the following strong guarantee; that is, our continuous greedy algorithm finds a fractional solution that achieves the optimal value for the $\mathrm{M}^\natural$-concave part and a $(1-1/e)$-approximation for the remaining monotone submodular part.

Rounding algorithm preserving the values of a multilinear extension and concave closure. A difficulty also arises in the rounding phase. Typically, a rounding algorithm finds a feasible subset that preserves the value of the multilinear extension. However, since our modified continuous greedy algorithm also involves a concave closure, we need to design a rounding algorithm that preserves the values of the multilinear extension and concave closure. To this end, we extend the *swap rounding* [4] algorithm for $\mathrm{M}^\natural$-concave functions. The original swap rounding algorithm is designed for rounding a fractional solution in a matroid base polytope into a matroid base without losing the value of a multilinear extension. We prove that almost the same strategy works for our purpose by exploiting the combinatorial structures of $\mathrm{M}^\natural$-concave functions.

1.3 Related work

The concept of curvature was introduced by Conforti and Cornuéjols, and they proved that the classical greedy algorithm of [20] achieves a $(1-e^{-c})/c$-approximation for a cardinality constraint. In [25], Vondrák introduced a slightly relaxed variant of the curvature, namely, the *curvature with respect to the optimum*. He obtained the same approximation guarantee $(1-e^{-c^*})/c^*$ for a matroid constraint, where c^* is the curvature with respect to the optimum, and also showed that the approximation guarantee is tight for general submodular functions. Later, this result was refined with the total curvature by Sviridenko, Vondrák, and Ward [24]. For a knapsack constraint, Yoshida [26] proved a better approximation ratio of $1-c/e$. Iyer and Bilmes [10] studied a general reduction between submodular function maximization and a *submodular cover*, and they proved that if the curvature of the functions involved is small, the two problems reduce to each other.

Discrete convex analysis originated with Murota [16]. We note that discrete convex functions are usually defined on the integer lattice $\mathbb{Z}^E$, although we focus on set functions in this paper. Discrete convex functions admit various attractive properties such as Fenchel

duality, the separation theorem, and descent-type algorithms, which are analogous to convex analysis in the Euclidean space. The applications of discrete convex analysis include economics, system analysis for electrical circuits, phylogenetic analysis, etc. Shioura [22] studied maximization of the sum of $\mathrm{M}^\natural$-concave set functions subject to a matroid constraint. He showed that pipage rounding [2] can be explained from the viewpoint of discrete convexity. For details, the reader is referred to a monograph [16] or recent survey [17].

The $\mathrm{M}^\natural$-natural concavity is equivalent to the *gross-substitutability* [17], a central concept in economics. Roughgarden, Talgam-Cohen, and Vondrák [21] studied approximation for maximizing set functions that are close to the gross-substitute functions and others. Recently, Chatziafratis, Roughgarden, and Vondrák [3] provided improved approximation guarantees via a different concept called *perturbation stability*. They proved that the greedy and local search algorithms achieve a better approximation for submodular function maximization for various set systems under a stability assumption.

1.4 Organization

In Section 2, we introduce the notation and basic concepts for submodular function maximization and discrete convex analysis. Our main theorem and continuous greedy algorithms are presented in Section 3. We describe a general algorithm for finding a decomposition of a given monotone submodular function into a monotone submodular function and an $\mathrm{M}^\natural$-concave function in Section 4. In Section 5, we provide several examples of problem-specific decompositions and the theoretical bounds of the $\mathrm{M}^\natural$-concave curvature.

2 Preliminaries

For a set $S \subseteq E$, $\mathbf{1}_S$ denotes the characteristic vector of S; that is, $\mathbf{1}_S(i) = 1$ if $i \in S$, and $\mathbf{1}_S(i) = 0$ otherwise. The dimension of the ambient space should be clear from the context. For $a \in \mathbb{R}$, we define $[a]^+ := \max\{a, 0\}$. For a vector $\boldsymbol{x} \in \mathbb{R}^E$ and $X \subseteq E$, we use the shorthand notation: $\boldsymbol{x}(X) := \sum_{i \in X} \boldsymbol{x}(i)$.

A pair consisting of a finite set E and a set family of $\mathcal{I} \subseteq 2^E$ is called a *matroid* if (i) $\varnothing \in \mathcal{I}$; (ii) if $X \in \mathcal{I}$, then any subset of X also belongs to $\mathcal{I}$; and (iii) if $X, Y \in \mathcal{I}$ and $|X| < |Y|$, then there exists $i \in Y \setminus X$ such that $X + i \in \mathcal{I}$. A member of $\mathcal{I}$ is called an *independent set*. A maximal independent set is called a *base*. The *base polytope* of a matroid is the convex hull of all characteristic vectors of its bases. The *rank function* of a matroid is the following set function: $r(X) = \max\{|I| : I \subseteq X, I \in \mathcal{I}\}$. Although it is well-known that matroid rank functions are monotone and submodular, they are indeed included in the more tractable class called $\mathrm{M}^\natural$-concave functions.

▶ **Definition 4** ($\mathrm{M}^\natural$-concave function [19]). A set function $f : 2^E \to \mathbb{R}$ is *$M^\natural$-concave* if for $X, Y \subseteq E$ and $i \in X \setminus Y$, either

$$f(X) + f(Y) \leq f(X - i) + f(Y + i), \tag{1}$$

or there exists $j \in Y \setminus X$ such that

$$f(X) + f(Y) \leq f(X - i + j) + f(Y - i + j). \tag{2}$$

It is known that $\mathrm{M}^\natural$-concave functions are submodular. Note that $\mathrm{M}^\natural$-concave functions are not necessarily monotone. We give several examples of $\mathrm{M}^\natural$-concave functions below.

▶ **Example 5** ($\mathrm{M}^\natural$-concave functions from univariate concave functions)**.** Let $\phi : \mathbb{R}_+ \to \mathbb{R}$ be a concave function. Then, $f(X) := \phi(|X|)$ is an $\mathrm{M}^\natural$-concave function. For example, $X \mapsto \sqrt{|X|}$ and $X \mapsto \min\{|X|, \alpha\}$ $(\alpha > 0)$ are $\mathrm{M}^\natural$-concave. More generally, let $\mathcal{L}$ be a laminar family on E and $\phi_L : \mathbb{R}_+ \to \mathbb{R}$ be a concave function for $L \in \mathcal{L}$. Then, $f(X) := \sum_{L\in\mathcal{L}} \phi_L(|X \cap L|)$ is an $\mathrm{M}^\natural$-concave function. Here, as a special case, a separable concave function $f(X) = \sum_{i\in E} \phi_i(|X \cap \{i\}|)$ is $\mathrm{M}^\natural$-concave.

▶ **Example 6** ($\mathrm{M}^\natural$-concave functions from matroids)**.** Let $\mathbf{M} = (E, \mathcal{I})$ be a matroid and $\boldsymbol{w} \in \mathbb{R}^E_+$. The *weighted matroid rank function* is the following set function: $f(X) = \max\{\boldsymbol{w}(I) : I \subseteq X, I \in \mathcal{I}\}$. That is, $f(X)$ is the maximum weight of independent sets in X. In particular, the matroid rank function is $\mathrm{M}^\natural$-concave.

If we restrict the domain of an $\mathrm{M}^\natural$-concave function to all subsets of the same cardinality, the condition in (1) can be omitted. Such functions are called *M-concave functions*.

For exploiting continuous greedy algorithms, we require continuous extensions of set functions.

▶ **Definition 7** (Multilinear extension)**.** The *multilinear extension* $F : [0,1]^E \to \mathbb{R}$ of a set function $f : 2^E \to \mathbb{R}$ is defined as

$$F(\boldsymbol{x}) = \sum_{X\subseteq E} f(X) \prod_{i\in X} \boldsymbol{x}(i) \prod_{i\in E\setminus X} (1 - \boldsymbol{x}(i)).$$

▶ **Definition 8** (Concave closure)**.** The *concave closure* $\bar{f} : [0,1]^E \to \mathbb{R}$ of a set function $f : 2^E \to \mathbb{R}$ is defined as

$$\bar{f}(\boldsymbol{x}) := \max\left\{ \sum_{Y\subseteq E} \lambda_Y f(Y) : \sum_Y \lambda_Y \mathbf{1}_Y = \boldsymbol{x}, \sum_Y \lambda_Y = 1, \lambda_Y \geq 0 \, (Y \subseteq E) \right\}. \tag{3}$$

If f is an $\mathrm{M}^\natural$-concave function, then $\bar{f}(\boldsymbol{x})$ can be computed in polynomial time for any $\boldsymbol{x} \in [0,1]^E$ [23]. Furthermore, the subgradients of $\bar{f}$ can be computed in polynomial time [23]. This yields a separation oracle for the constraint of the form $\bar{f}(\boldsymbol{x}) \geq \alpha$.

3 Algorithms

In this section, we prove Theorem 3. To this end, we show the following, which simultaneously optimizes g and h:

▶ **Theorem 9.** *Let $f : 2^E \to \mathbb{R}_+$ be a monotone submodular function decomposed as $f = g + h$ for a monotone submodular function $g : 2^E \to \mathbb{R}_+$ and an $M^\natural$-concave function $h : 2^E \to \mathbb{R}$, and let k be a positive integer. Then, there exists a polynomial-time algorithm that finds a random set $X \subseteq E$ of cardinality k such that*

$$\mathbf{E}[g(X)] \geq \left(1 - \frac{1}{e}\right) g(O) - \epsilon M, \quad \mathbf{E}[h(X)] \geq h(O) - \epsilon M,$$

for any constant $\epsilon > 0$, where O is an optimal solution for $\max\{f(X) : |X| \leq k\}$ and $M = \max_{e\in E} \max\{g(e), h(e)\}$.

We note that the special case in which h is modular is proved in [24].

Before proving Theorem 9, we see that Theorem 3 is an easy consequence of Theorem 9:

Algorithm 1 Modified Continuous Greedy Algorithm

Require: $g(O), h(O) \in \mathbb{R}_+$, value oracles of $G, \bar{h} : [0,1]^E \to \mathbb{R}_+$, and a value oracle of $\nabla G : [0,1]^E \to \mathbb{R}_+^E$.

1: Set $\boldsymbol{x}(0) := \mathbf{0} \in \mathbb{R}^E$

2: From $t = 0$ to $t = 1$, compute $\boldsymbol{x}(t) \in [0,1]^E$ following the differential equation $\frac{d\boldsymbol{x}}{dt} = \boldsymbol{v}(t)$, where $\boldsymbol{v} \in [0,1]^E$ is a vector satisfying $\boldsymbol{v}(t)^\top \nabla G(\boldsymbol{x}) \geq g(O) - G(\boldsymbol{x}), \bar{h}(\boldsymbol{v}(t)) \geq h(O)$, and $\boldsymbol{v}(t) \in P_k$.

3: **return** $\boldsymbol{x}(1)$.

Proof of Theorem 3. By Theorem 9, we can compute a random $X \subseteq E$ of size k such that

$$\begin{aligned}\mathbf{E}[f(X)] &\geq \left(1 - \frac{1}{e}\right) g(O) + h(O) - 2\epsilon M = \left(1 - \frac{1}{e}\right) f(O) + \frac{1}{e} h(O) - 2\epsilon M \\ &\geq \left(1 - \frac{1}{e}\right) f(O) + \frac{1-\gamma}{e} f(O) - 2\epsilon M = \left(1 - \frac{\gamma}{e}\right) f(O) - 2\epsilon M \\ &\geq \left(1 - \frac{\gamma}{e} - O(\epsilon)\right) f(O),\end{aligned}$$

where in the last inequality, we used the fact that $M = \max_{e \in E} \max\{g(e), h(e)\} \leq g(O) + h(O)$. ◀

In the rest of this section, we fix f and its decomposition $f = g + h$, and the goal is a proof of Theorem 9.

3.1 Continuous-time algorithm for simultaneous optimization

First, we present a continuous-time version of our algorithm. Although we cannot run this version in polynomial time, it is helpful to grasp the overall idea.

Our algorithm is based on the continuous greedy framework [2]. Let $G : [0,1]^E \to \mathbb{R}_+$ be the multilinear extension of g and $\bar{h} : [0,1]^E \to \mathbb{R}_+$ be the concave closure of h, and let P_k be the convex hull of all characteristic vectors of subsets of size at most k. We assume that we have value oracles of G, ∇G, and $\bar{h}$. We also assume that we know $g(O)$ and $h(O)$, where O is the optimal solution to $\max\{f(X) : |X| \leq k\}$. We discuss how we can remove these assumptions in Section 3.2. The pseudocode of the continuous greedy algorithm is given in Algorithm 1.

▶ Remark. One can find $\boldsymbol{v}$ in Algorithm 1 in polynomial time. First, the problem of finding $\boldsymbol{v}$ is a feasible problem of a convex program. This problem is feasible because $\mathbf{1}_O \in \mathbb{R}^E$ is a feasible solution. Moreover, we have a separation oracle for $\bar{h}(\boldsymbol{v}) \geq h(O)$. Hence, we can find a feasible solution using the ellipsoid method.

The following lemma shows that Algorithm 1 provides the guarantee required in Theorem 9:

▶ **Lemma 10.** *Let $\boldsymbol{x}(t)$ be the sequence computed in Algorithm 1. Then, we have $G(\boldsymbol{x}(1)) \geq (1 - 1/e)g(O)$ and $\bar{h}(\boldsymbol{x}(1)) \geq h(O)$.*

Proof. By adopting the standard analysis of the continuous greedy algorithm, we can see that $G(\boldsymbol{x}(1)) \geq (1 - 1/e)g(O)$ (see, e.g., [2]). On the other hand, since $\bar{h}$ is a concave function, $\bar{h}(\boldsymbol{x}(1)) = \bar{h}\left(\int_0^1 \boldsymbol{v}(t)dt\right) \geq \int_0^1 \bar{h}(\boldsymbol{v}(t))dt \geq h(O)$, where the first inequality follows from Jensen's inequality. ◀

3.2 Discrete-time algorithm for simultaneous optimization

Now, we describe the discrete-time version of Algorithm 1, which can be run in polynomial time. Again, we basically follow the argument in [24] and explain the modifications to Algorithm 1. We define $M = \max_{e \in E} \max\{g(e), h(e)\}$ as the maximum marginal gain of adding a single element. The detail and proof are deferred to the full version.

▶ **Lemma 11.** *For any constant $\epsilon > 0$, there exists a polynomial-time algorithm that finds a vector $\boldsymbol{x} \in P_k$ such that $G(\boldsymbol{x}) \geq (1 - 1/e)g(O) - \epsilon M$ and $\bar{h}(\boldsymbol{x}) \geq h(O) - \epsilon M$.*

3.3 Rounding

In this section, we show the following:

▶ **Lemma 12.** *Let $\boldsymbol{x} \in P_k$ be a vector. Then, there exists a polynomial-time randomized rounding algorithm that outputs $X \subseteq E$ with $|X| \leq k$ such that $\mathbf{E}[g(X)] \geq G(\boldsymbol{x})$ and $\mathbf{E}[h(X)] \geq \bar{h}(\boldsymbol{x})$.*

Theorem 9 is obtained by combining Lemmas 11 and 12. In the following, we show a rounding method, provided that a convex combination of $\bar{h}(\boldsymbol{x})$ is known. The details on how to compute a convex combination is described in the full version.

Suppose that we have a convex combination of $\bar{h}(\boldsymbol{x})$; that is, we have $\lambda_1, \dots, \lambda_m > 0$ and $X_1, \dots, X_m \subseteq E$ of size k with $\sum_i \lambda_i = 1$, $\sum_i \lambda_i \mathbf{1}_{X_i} = \boldsymbol{x}$, and

$$\bar{h}(\boldsymbol{x}) = \sum_{i=1}^{m} \lambda_i h(X_i). \tag{4}$$

Our rounding is a random process, and the condition $|X_i| = k$ ($i \in [m]$) is always preserved throughout the process.

If $X_1 = \cdots = X_m$, then we are done, and we output the set $X := X_1 (= \cdots = X_m)$, which is of size k. Otherwise, find $1 \leq a < b \leq m$ such that $X_a \setminus X_b \neq \varnothing$ and $X_a \setminus X_b \neq \varnothing$. Then, we fix $i \in X_a \setminus X_b$ and find $j \in X_b \setminus X_a$ such that $h(X_a) + h(X_b) \leq h(X_a - i + j) + h(X_b + i - j)$, whose existence is guaranteed by the $\mathrm{M}^\natural$-concavity of h. Then, we replace X_a with $X_a - i + j$ with the probability $\lambda_b/(\lambda_a + \lambda_b)$, replace X_b with $X_b + i - j$ with the complementary probability, and repeat this process again.

We now analyze the expected value of h at the end of the rounding process.

▶ **Lemma 13.** *Let $X \subseteq E$ be the output set. Then, we have $\mathbf{E}[h(X)] \geq \bar{h}(\boldsymbol{x})$.*

Proof. Suppose that we have a chosen $i \in X_a \setminus X_b$ and $j \in X_b \setminus X_a$ in an iteration of the process. Then, the expected change in the value of $\lambda_a h(X_a) + \lambda_b h(X_b)$ at this iteration is

$$\frac{\lambda_a \lambda_b}{\lambda_a + \lambda_b}[h(X_a - i + j) - h(X_a) + h(X_b + i - j) - h(X_b)] \geq 0.$$

Therefore, the expected value of $\sum_{i=1}^{m} \lambda_i h(X_i)$ is at least $\bar{h}(\boldsymbol{x})$. The lemma holds by induction on the number of iterations. ◀

Next, we analyze the expected value of g at the end of the rounding process. Let $\boldsymbol{x}^t = \sum_i \lambda_i \mathbf{1}_{X_i}$ be the random vector after the tth exchange ($t = 0, 1, \dots$). One can check that (i) $\boldsymbol{x}^0 = \boldsymbol{x}$, (ii) $\boldsymbol{x}^{t+1} - \boldsymbol{x}^t$ has at most one positive coordinate and at most one negative coordinate, and (iii) $\mathbf{E}[\boldsymbol{x}^{t+1} \mid \boldsymbol{x}^t] = \boldsymbol{x}^t$. The following lemma establishes that such a random process preserves the value of the multilinear extension of a monotone submodular function.

▶ **Lemma 14** ([4, Lemma VI.2]). *Let $\boldsymbol{x}^t$ be a vector-valued random process satisfying the above conditions (i), (ii), and (iii). Then, for the multilinear extension F of any monotone submodular function $f : 2^E \to \mathbb{R}$, we have $\mathbf{E}[F(\boldsymbol{x}^t)] \geq F(\boldsymbol{x})$ for $t = 0, 1, \ldots$.*

By Lemmas 13 and 14, we have the following lemma.

▶ **Lemma 15.** *Let X be the output of the above rounding process. Then, we have $\mathbf{E}[g(X)] \geq G(\boldsymbol{x})$ and $\mathbf{E}[h(X)] \geq \bar{h}(\boldsymbol{x})$.*

Thus, we can round a fractional solution into an integral solution without losing the objective value (in the expectation).

▶ Remark. Although the continuous greedy step also applies to any down-closed and solvable polytope, the rounding step is able to handle only cardinality constraints. For example, if we consider a matroid constraint, we need to a rounding algorithm that preserves the value of $\bar{h}$ in a matroid polytope. Since the face of $\bar{h}(\boldsymbol{x}) \geq \alpha$ is another matroid polytope, we must essentially deal with a *matroid intersection* constraint. Unfortunately, there is no known rounding algorithm for matroid intersection preserving the value of the multilinear extensions[4]. We can use a contention resolution scheme [5] instead, but this does not preserve the of the value of the multilinear extension and results in an approximation ratio worse than $1 - 1/e$.

4 Decomposition via M$^\natural$-concave Quadratic Functions

In this section, we describe a general decomposition scheme that exploits M$^\natural$-concave quadratic functions.

For a vector $\boldsymbol{d} \in \mathbb{R}^E$, we define $\mathrm{Diag}(\boldsymbol{d}) \in \mathbb{R}^{E \times E}$ as a diagonal matrix such that $\mathrm{Diag}(\boldsymbol{d})_{ii} = \boldsymbol{d}(i)$ for every $i \in E$. The following characterization of M$^\natural$-concave quadratic set functions is known:

▶ **Lemma 16** ([9], also see [17, Theorem 4.3]). *Let $A \in \mathbb{R}^{E \times E}$ be a symmetric matrix, and define $h : 2^E \to \mathbb{R}$ as $h(X) = \frac{1}{2}\mathbf{1}_X^\top A \mathbf{1}_X = \frac{1}{2}\sum_{i \in X} a_{ii} + \sum_{i,j \in X : i \neq j} a_{ij}$. Then, $h(X)$ is M$^\natural$-concave if and only if*

$$A_{ij} \leq 0 \quad \text{for any distinct } i, j \in E, \text{and} \tag{5}$$

$$A_{ij} \leq \max\{A_{ik}, A_{jk}\} \quad \text{for any distinct } i, j, k \in E. \tag{6}$$

Note that the condition in (6) says that A_{ij} must form an *ultrametric*. The above condition is closely related to the concept of a discrete Hessian matrix:

▶ **Definition 17.** The *discrete Hessian* $\mathrm{Hess}_f(X) \in \mathbb{R}^{E \times E}$ of a set function $f : 2^E \to \mathbb{R}$ at a set $X \subseteq E$ is defined as

$$\mathrm{Hess}_f(X)_{ij} = f(X + i + j) + f(X) - f(X + i) - f(X + j). \quad (i, j \in E)$$

We note that $\mathrm{Hess}_f(X)_{ij} = 0$ when i or j belongs to X. By definition, a set function $f : 2^E \to \mathbb{R}$ is submodular if and only if $\mathrm{Hess}_f(X)_{ij} \leq 0$ for any $X \subseteq E$ and distinct $i, j \in E$. A characterization of an M$^\natural$-concave function in terms of a discrete Hessian is also known:

[4] For a special class of submodular functions, a rounding algorithm for a matroid intersection is proposed in [4].

▶ **Lemma 18** ([17, Theorem 4.3]). *A set function $f : 2^E \to \mathbb{R}$ is $M^\natural$-concave if and only if its discrete Hessian satisfies the conditions in* (5) *and* (6) *at any $X \subseteq E$.*

For a monotone submodular function $f : 2^E \to \mathbb{R}_+$, consider a decomposition $f = g + h$, where $h(X) = \frac{1}{2}\mathbf{1}_X^\top A \mathbf{1}_X$ $(X \subseteq E)$ for some matrix $A \in \mathbb{R}^{E \times E}$. In the following, we derive a sufficient condition of A that makes g nonnegative monotone submodular and makes h nonnegative M$^\natural$-concave so that we can apply Theorem 3.

For the monotonicity of g, we must have $g(i \mid X) = f(i \mid X) - h(i \mid X) \geq 0$ for any $i \in E$ and $X \subseteq E - i$. This yields

$$\sum_{k \in X} A_{ik} + \frac{1}{2} A_{ii} \leq f(i \mid X). \quad (i \in E, X \subseteq E - i) \tag{7}$$

We note that, if (7) was satisfied, g becomes automatically nonnegative as $h(\varnothing) = 0$.

For the submodularity of g, we must have $\mathrm{Hess}_g(X)_{ij} = \mathrm{Hess}_f(X)_{ij} - \mathrm{Hess}_h(X)_{ij} \leq 0$ for any distinct $i, j \in E$ and $X \subseteq E \setminus \{i, j\}$. This yields

$$A_{ij} \geq \mathrm{Hess}_f(X)_{ij}. \quad (i, j \in E, i \neq j, X \subseteq E \setminus \{i, j\}) \tag{8}$$

Finally, for the nonnegativity of h, we must have

$$\mathbf{1}_X^\top A \mathbf{1}_X \geq 0 \quad (X \subseteq E). \tag{9}$$

It is difficult to handle these conditions simultaneously. Fortunately, we can show that we need to consider only the condition in (8) and the nonpositivity constraint.

▶ **Lemma 19.** *Suppose that a matrix $A \in \mathbb{R}^{E \times E}$ satisfies the conditions in* (5), (6), *and* (8). *Then, $h(X) = \frac{1}{2}\mathbf{1}_X^\top A \mathbf{1}_X$ is nonnegative $M^\natural$-concave, and $g = f - h$ is nonnegative monotone submodular.*

Trivially, we can take $A = 2\,\mathrm{Diag}(f(i \mid E - i) : i \in E)$, which yields $h(X) = \sum_{i \in X} f(i \mid E - i)$. This decomposition corresponds to that via the standard curvature. In the following, we discuss how we can find a nontrivial matrix A satisfying the conditions in (5), (6), and (8).

4.1 Ultrametric fitting problem

Assume that we can compute H_{ij} such that $\max_{X \subseteq E - i - j} \mathrm{Hess}_f(X)_{ij} \leq H_{ij} \leq 0$ for any distinct $i, j \in E$. Once we are given such H_{ij}, finding a matrix A satisfying the conditions in (5), (6), and (8) boils down to the *ultrametric fitting problem* [8]. In this problem, given $H_{ij} \leq 0$ for distinct $i, j \in E$, we are to find $A_{ij} \in [H_{ij}, 0]$ for distinct $i, j \in E$ satisfying (6) that minimizes $\max_{i \neq j} |A_{ij} - H_{ij}|$. Farach et al. [8] gave an $O(n^2)$-time algorithm for this problem. Therefore, we can find a matrix A satisfying the conditions in (5), (6), and (8) that is close to H_{ij}. Although this method is general, it is difficult to theoretically bound the M$^\natural$-concave curvature of the resulting decomposition.

4.2 Coverage function

Let $G = (E, V; A)$ be a bipartite graph. Note that here we denote by E the one of the vertex sets (i.e., the ground set) for the sake of consistency with the other parts of this paper. The coverage function $f : 2^E \to \mathbb{R}$ associated with G is defined as $f(X) := \sum_{v \in V} \min\{1, |\Gamma(v) \cap X|\}$, where $\Gamma(v) \subseteq E$ is the set of neighbors of $v \in V$. It is well-known that f is a monotone submodular function.

Now, we consider decomposing f into a monotone submodular function and monotone $\mathrm{M}^\natural$-concave function. For each $v \in V$, we define $f_v : 2^E \to \mathbb{R}$ as $f_v(X) = \min\{1, |\Gamma(v) \cap X|\}$. Then, for distinct $i, j \in E$ and $X \subseteq E - i - j$, $\mathrm{Hess}_{f_v}(X)_{ij}$ is equal to -1 if $v \notin \Gamma(X)$ and $i, j \in \Gamma(v)$ and 0 otherwise. Thus, we have $\mathrm{Hess}_f(X)_{ij} = -|\Gamma(i) \cap \Gamma(j) - \Gamma(X)|$. Since $\mathrm{Hess}_f(X)_{ij}$ is obviously nondecreasing, we can compute the exact value of $\max_{X \subseteq E \setminus \{i,j\}} \mathrm{Hess}_f(X)_{ij} = \mathrm{Hess}_f(E - i - j)_{ij} = -c_{ij}$, where $c_{ij} = |\Gamma(i) \cap \Gamma(j) - \Gamma(E - i - j)|$. Note that the quantity c_{ij} is the number of $v \in V$ whose neighbors are exactly i and j. Then, we can run the algorithm presented in Section 4.1 with $H_{ij} = -|\Gamma(i) \cap \Gamma(j) - \Gamma(E - i - j)|$ to obtain a matrix A satisfying the conditions in (5), (6), and (8). We note that the standard curvature can only handle vertices in V that have only one neighbor in E, whereas our argument can handle vertices in V that have two neighbors in E.

5 Applications

In this section, we apply Theorem 3 to obtain better approximation guarantees for the facility location problem and the sum of weighted matroid rank functions. In these applications, we need to use the specific structures of these problems.

5.1 Facility location

In the *facility location problem*, there are a set I of customers and a set E of possible locations of facilities. Each customer $i \in I$ has a revenue $w_{ij} \geq 0$ for the facility $j \in E$. We assume that customers will select the available facility of maximum revenue. The task is to select a set $X \subseteq E$ of cardinality k that maximizes

$$f(X) = \sum_{i \in I} \max_{j \in X} w_{ij},$$

where we conventionally define $f(\varnothing) = 0$. Evidently, f is a nonnegative monotone submodular function.

We can decompose f into g and h as follows. For $i \in I$, we denote $w_{i,\min} := \min_{j \in E} w_{ij}$ and let $\bar{w}_{ij} := w_{ij} - w_{i,\min}$ $(i \in I,\, j \in E)$. Then, we can rewrite f as

$$f(X) = \sum_{i \in I} \max_{j \in X} \bar{w}_{ij} + \left(\sum_{i \in I} w_{i,\min} \right) [X \neq \varnothing],$$

where $[X \neq \varnothing]$ is the indicator function of the nonemptiness of X. Let $\tilde{f}(X) = \sum_{i \in I} \max_{j \in X} \bar{w}_{ij}$ be the first term. We further subtract a modular function $\ell(X) = \sum_{j \in X} \tilde{f}(j \mid E - j)$ from $\tilde{f}$. Note that since $f(j \mid E - j) = \sum_{i \in I} [w_{ij} - \max_{k \neq j} w_{ij}]^+ = \sum_{i \in I} [\bar{w}_{ij} - \max_{k \neq j} \bar{w}_{ij}]^+ = \tilde{f}(j \mid E - j)$ for $j \in E$, this modular term is exactly the same one in the curvature decomposition for the original function f. Then, we define $g(X) := \tilde{f}(X) - \ell(X)$ and $h(X) := \ell(X) + \left(\sum_{i \in I} w_{i,\min}\right) [X \neq \varnothing]$. One can easily check that g is monotone submodular and h is $\mathrm{M}^\natural$-concave.

We can show that $\gamma(g, h)$ is no more than the standard curvature c.

▶ **Lemma 20.** *$\gamma(g, h) \leq c - \dfrac{\sum_{i \in I} w_{i,\min}}{\sum_{i \in I} w_{i,\max}}$, where $w_{i,\max} := \max_{j \in E} w_{ij}$ $(i \in E)$.*

Proof. For any nonempty $X \subseteq E$, we have

$$\frac{h(X)}{f(X)} = \frac{\ell(X)}{f(X)} + \frac{\sum_{i \in I} w_{i,\min}}{f(X)} \geq 1 - c + \frac{\sum_{i \in I} w_{i,\min}}{f(E)} = 1 - c + \frac{\sum_{i \in I} w_{i,\min}}{\sum_{i \in I} w_{i,\max}},$$

where the inequality follows from the definition of the curvature and the monotonicity of f. Therefore, $\gamma_h = 1 - \min_{X \subseteq E} \frac{h(X)}{f(X)} \leq c - \dfrac{\sum_{i \in I} w_{i,\min}}{\sum_{i \in I} w_{i,\max}}$. ◀

5.2 Sums of weighted matroid rank functions

A weighted matroid rank function is an $\mathrm{M}^\natural$-concave function (see Example 6), and its sum is a monotone submodular function. A sum of weighted matroid rank functions includes the coverage function, the facility location function, and many others, but not all monotone submodular functions. Here, we show that if the objective function is a sum of weighted matroid rank functions, we can obtain an improved approximation guarantee.

Let $f(X) = \sum_{i\in I} f_i(X)$, where $f_i(X) = \max\{\boldsymbol{w}_i(J) : J \subseteq X, J \in \mathcal{I}_i\}$ for some vector $\boldsymbol{w}_i \in \mathbb{R}_+^E$, and $\mathcal{I}_i$ is the independent set family of a matroid ($i \in I$). For $i \in I$, let $w_{i,\min} := \min_{e\in E} \boldsymbol{w}_i(e)$. Let $\tilde{f}(X) = \sum_{i\in I} \tilde{g}_i(X)$, where $\tilde{g}_i(X)$ is the weighted matroid rank function with the reduced weight $\boldsymbol{w}_i - w_{i,\min}\mathbf{1}$ for $i \in I$. Define $\ell(X) = \sum_{j\in X} \tilde{f}(j \mid E - j) = \sum_{j\in X} f(j \mid E - j)$, where the second equality follows from the fact that all matroid bases have the same cardinality. Finally, let us define $g(X) := \tilde{f}(X) - \ell(X)$ and $h(X) := \ell(X) + (\sum_{i\in I} w_{i,\min})[X \neq \varnothing]$. The proof of the following lemma is similar to that of Lemma 20.

▶ **Lemma 21.** *$\gamma(g, h) \leq c - \frac{\sum_{i\in I} w_{i,\min}}{\sum_{i\in I} W_i}$, where c is the standard curvature, and $W_i := f_i(E)$ is the maximum weight of the bases in $(E, \mathcal{I}_i)$ for each i.*

In [15], Lin and Bilmes proposed a linear mixture model of simple monotone submodular functions. One of the most general classes in their model is a linear mixture of weighted matroid rank functions. Suppose that a function $f : 2^E \to \mathbb{R}_+$ is obtained by inference for this linear mixture model. Then, we can write $f(X) = \sum_{i\in I} \alpha_i f_i(X)$, where $\alpha_i > 0$ is a mixture coefficient, and $f_i(X)$ is a weighted matroid rank function ($i \in I$). If some coefficient α_{i^*} is dominant, one can consider the following straightforward decomposition: $g(X) = \sum_{i\neq i^*} \alpha_i f_i(X)$ and $h(X) = \alpha_{i^*} f_{i^*}(X)$. Then, we can expect the resulting curvature to be small, although the actual value depends on the form of f_i ($i \in I$).

6 Conclusion and Open Problems

We propose a new concept the $\mathrm{M}^\natural$-concave curvature, which measures how a given submodular function deviates from $\mathrm{M}^\natural$-concave functions. Based on this concept, we designed a polynomial-time algorithm given a decomposition in the form of $f = g + h$, which generalizes the result of [24]. We complemented our algorithm by devising an algorithm for finding such a decomposition, which always yields a decomposition that is as good as a decomposition based on the standard curvature. We showed examples in which the the $\mathrm{M}^\natural$-concave curvature is better than the standard curvature.

We believe that the $\mathrm{M}^\natural$-concave curvature is generally useful for analysing the performance of algorithms for submodular maximization. Although in this paper we focused on the continuous greedy algorithm, it is quite natural to analyze the performance of other algorithms, e.g., local search. Our algorithm is able to handle only a cardinality constraint due to the difficulty in the rounding step. A possible direction for future work is dealing with more complicated constraints such as matroid constraints and knapsack constraints.

References

1 Eric Balkanski, Aviad Rubinstein, and Yaron Singer. The power of optimization from samples. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4017–4025, 2016.

2 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

3 Vaggos Chatziafratis, Tim Roughgarden, and Jan Vondrak. Stability and recovery for independence systems. In *Proceedings of ALGO*, 2017.

4 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *Proceedings of IEEE 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 575–584, 2010.

5 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM Journal on Computing*, 43(6):1831–1879, 2014.

6 Michele Conforti and Gérard Cornuéjols. Submodular set functions, matroids and the greedy algorithm: Tight worst-case bounds and some generalizations of the Rado-Edmonds theorem. *Discrete Applied Mathematics*, 7(3):251–274, 1984.

7 Shahar Dobzinski, Noam Nisan, and Michael Schapira. Approximation algorithms for combinatorial auctions with complement-free bidders. *Mathematics of Operations Research*, 35(1):1–13, 2010.

8 M. Farach, S. Kannan, and Tandy Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13:155–179, 1995. doi:10.1007/BF01188585.

9 Hiroshi Hirai and Kazuo Murota. M-convex functions and tree metrics. *Japan Journal of Industrial and Applied Mathematics*, 21(3):391–403, 2004.

10 Rishabh Iyer and Jeff Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2436–2444, 2013.

11 David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 137–146, 2003.

12 Andreas Krause and Carlos Guestrin. Submodularity and its applications in optimized information gathering. *ACM Transactions on Intelligent Systems and Technology*, 2(4):32–20, 2011.

13 Andreas Krause, Ajit Paul Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, pages 235–284, 2008.

14 Benny Lehmann, Daniel J Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55:270–296, 2006.

15 Hui Lin and Jeff Bilmes. Learning mixtures of submodular shells with application to document summarization. In *Uncertainty in Artificial Intelligence (UAI)*, 2012.

16 Kazuo Murota. *Discrete Convex Analysis*. Society for Industrial and Applied Mathematics, 2003.

17 Kazuo Murota. Discrete convex analysis: A tool for economics and game theory. *Journal of Mechanism and Institution Design*, 1(1):151–273, 2016.

18 Kazuo Murota and Akiyoshi Shioura. M-convex function on generalized polymatroid. *Mathematics of Operations Research*, 24(1):95–105, 1999.

19 Kazuo Murota and Akiyoshi Shioura. Extension of m-convexity and L-convexity to polyhedral convex functions. *Advances in Applied Mathematics*, 25(4):352–427, nov 2000.

20 G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Mathematical Programming*, 14(1):265–294, 1978.

21 Tim Roughgarden, Inbal Talgam-Cohen, and Jan Vondrák. When are welfare guarantees robust? In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM).*, 2017.

22 Akiyoshi Shioura. On the pipage rounding algorithm for submodular function maximization — a view from discrete convex analysis—. *Discrete Mathematics, Algorithms and Applications*, 1(1):1–23, 2009.

23 Akiyoshi Shioura. Polynomial-time approximation schemes for maximizing gross substitutes utility under budget constraints. *Mathematics of Operations Research*, 40(1):192–225, 2015.

24 Maxim Sviridenko, Jan Vondrák, and Justin Ward. Optimal approximation for submodular and supermodular optimization with bounded curvature. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1134–1148, 2015.

25 Jan Vondrák. Submodularity and curvature: the optimal algorithm. *RIMS kokyuroku Bessatsu*, 23:253–266, 2010.

26 Yuichi Yoshida. Maximizing a monotone submodular function with a bounded curvature under a knapsack constraint. *arXiv preprint*, 2016.

Ring Packing and Amortized FHEW Bootstrapping

Daniele Micciancio
Department of Computer Science and Engineering, University of California - San Diego, CA, USA
daniele@cs.ucsd.edu

Jessica Sorrell
Department of Computer Science and Engineering, University of California - San Diego, CA, USA
jlsorrel@cs.ucsd.edu

Abstract

The FHEW fully homomorphic encryption scheme (Ducas and Micciancio, Eurocrypt 2015) offers very fast homomorphic NAND-gate computations (on encrypted data) and a relatively fast refreshing procedure that allows to homomorphically evaluate arbitrary NAND boolean circuits. Unfortunately, the refreshing procedure needs to be executed after every single NAND computation, and each refreshing operates on a single encrypted bit, greatly decreasing the overall throughput of the scheme. We give a new refreshing procedure that simultaneously refreshes n FHEW ciphertexts, at a cost comparable to a single-bit FHEW refreshing operation. As a result, the cost of each refreshing is amortized over n encrypted bits, improving the throughput for the homomorphic evaluation of boolean circuits roughly by a factor n.

2012 ACM Subject Classification Security and privacy → Cryptography

Keywords and phrases homomorphic encryption, bootstrapping, lattice-based cryptography

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.100

Related Version A full version of the paper is available at `https://eprint.iacr.org/2018/532.pdf`.

Funding Research supported in part by the Defense Advanced Research Project Agency (DARPA) and the U.S. Army Research Office under the SafeWare program, and the National Science Foundation (NSF) under grant CNS-1528068. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect reflect the views, position or policy of the Government.

1 Introduction

Since Gentry's first construction of a Fully Homomorphic Encryption (FHE) scheme [13], much research has been done to improve both the security and efficiency of FHE. On the security front, a line of works [14, 8, 4, 6, 19] has led to a FHE scheme of Brakerski and Vaikuntanathan [9] based on learning with errors for polynomial approximation factors and therefore essentially *as secure as* regular (non-homomorphic) lattice-based *public-key encryption* [23].

On the efficiency front, major progress has been achieved too, but we are still very far from reaching the ideal goal of an FHE scheme *as efficient as public key encryption*. Brakerski, Gentry, and Vaikuntanathan [6] give a scheme for homomorphic evaluation of circuits of depth

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 100; pp. 100:1–100:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

L and security parameter λ requiring $\tilde{O}(\lambda \cdot L^3)$ per-gate computation. Gentry, Halevi, and Smart [18] used similar techniques to achieve homomorphic evaluation of width-$\Omega(\lambda)$ circuits with only polylog(λ) per-gate computation. However, these schemes place restrictions on circuit depth or size and additionally rely on the hardness of RingLWE for quasi-polynomial approximation factors, a stronger assumption than that used for public-key encryption.

Gentry's bootstrapping technique [13] is still the only known method to achieve *fully* homomorphic encryption, i.e., an encryption scheme capable of evaluating homomorphically arbitrary circuits. We recall that Gentry's bootstrapping technique involves the homomorphic computation of the decryption function on an encryption of the decryption key, a rather complex operation, and this operation needs to be performed on all wires, for every few layers of the circuit. Therefore, improving the effectiveness of bootstrapping has been the main goal of many papers aimed at making FHE faster.

Improvements to bootstrapping have been pursued following two different approaches. The first approach, extensively studied in [5, 18, 17, 16, 15, 24, 20, 21], involves the construction of FHE schemes that can pack several messages into a single ciphertext, and operate on them in parallel. While bootstrapping such a scheme may still be very expensive, it can simultaneously refresh a large number of ciphertexts in a single bootstrapping execution. This reduces the total number of times that the bootstrapping procedure needs to be executed, and the amortized cost of bootstrapping over a large (and sufficiently wide) circuit.

A newer approach, explored in [1, 2, 12, 10], works towards reducing the cost of bootstrapping a single ciphertext as much as possible, even at the price of having to perform a bootstrapping operation for every gate of the circuit. Alperin-Sheriff and Peikert [2] introduced a bootstrapping technique requiring $\tilde{O}(\lambda)$ homomorphic operations. Building upon this technique, Ducas and Micciancio [12] brought the running time of a single bootstrapping execution down to a fraction of a second, with further improvements from Chillotti, Gama, Georgieva, and Izabachène [10]. However, it comes with the limitation that the bootstrapping procedure needs to be executed for essentially every gate of the circuit, without packing several messages into a single ciphertext. So bootstrapping is much faster than, say, in HElib [20, 21], but the amortized cost per gate is still quite high.

The goal of this work is to combine the advantages of these two approaches, and show how to simultaneously refresh $O(n)$ messages (where $n = \tilde{O}(\lambda)$), but at a cost comparable to that of [1, 2, 12, 10]. Our starting point is the FHEW bootstrapping method of [12]. We remark that FHEW has been improved in some follow-up works: [3] extended the FHEW scheme to larger gates, and [10] further reduced the running time of bootstrapping, partly at the cost of making a stronger security assumption on Ring-LWE with binary secrets.[1] However, while practically relevant, both improvements are asymptotically modest: the method of [3] is limited to gates with at most $O(\log n)$ input wires, and the speed-up achieved in [10] is at most polylogarithmic. In fact, in this paper, we will make bootstrapping even slower than [12], by a factor $O(n^\epsilon)$. The advantage is that, while the bootstrapping cost gets slightly higher, we will simultaneously refresh $O(n)$ messages, reducing the amortized bootstrapping cost per message by almost a factor of n to $\tilde{O}(3^{1/\epsilon} n^\epsilon)$ and for $\epsilon < 1/2$.

In the remainder of the introduction, we provide a detailed description of the FHEW bootstrapping problem, followed by a technical overview of the high level structure of our solution.

[1] In [10], and in this work, Ring-LWE is used with binary secrets, which may be justifiable based on the best known cryptanalysis methods, but would still benefit from more theoretical investigations.

1.1 The FHEW bootstrapping problem

The starting point of this work is the "FHEW" fully homomorphic encryption scheme of [12]. In this overview, we assume some basic familiarity with LWE encryption, and use notation $\mathsf{LWE}_n^{t/q}[m,\delta]$ for the LWE encryption of a message $m \in \mathbb{Z}_t$, with secret key in $\mathbb{Z}_q^n$ and noise level δ. Similarly, we write $\mathsf{RingLWE}_d^{t/q}[m,\delta]$ for Ring LWE ciphertexts over the dth cyclotomic ring encrypting a polynomial $m(X)$ of degree $\varphi(d)$.[2] The reader is referred to Section 2.4 for background information on LWE, and for a formal definition of the notation. In its most basic form, FHEW uses a function $\mathsf{HomNAND}\colon \mathsf{LWE}_n^{4/q}[\delta] \times \mathsf{LWE}_n^{4/q}[\delta] \to \mathsf{LWE}_n^{2/q}[\Delta]$, mapping the encryption of two bits $b_0, b_1 \in \{0,1\} \subset \mathbb{Z}_4$ (encoded as integers modulo 4), to the encryption of their logical "NAND", $b_0 \barwedge b_1 = \neg(b_0 \wedge b_1)$, but with somewhat larger noise Δ and encoded as an integer modulo 2. (We refer the reader to the full version of the paper for a formal definition and analysis of the $\mathsf{HomNAND}$ function.) Since these operations are not immediately composable, to evaluate arbitrary circuits on encrypted data, [12] also provides a refreshing procedure, $\mathsf{Refresh} : \mathsf{LWE}_n^{2/q}[\Delta] \to \mathsf{LWE}_n^{4/q}[\delta]$, that maps noisy ciphertexts modulo 2, back to ciphertexts modulo 4 with low noise δ. This refreshing involves the homomorphic computation of the decryption function, and it is rather costly: on a first approximation, it involves $\tilde{O}(n)$ homomorphic modular multiplications on data encrypted under a ring/symmetric-key variant of the GSW cryptosystem [19], which we recall in Section 2.5.

Our goal is to show that one can *simultaneously* refresh a large number of ciphertexts (say, $O(n)$) at a cost comparable to a single FHEW refreshing: approximately $O(n^{1+\epsilon})$ homomorphic modular multiplications on GSW ciphertexts. This reduces the amortized cost of refreshing to just $O(n^\epsilon)$ homomorphic (GSW) multiplications per ciphertext, rather than $O(n)$ as in the original FHEW cryptosystem.

▶ **Theorem 1.** *For every $0 < \epsilon < 1/2$, there exists an algorithm* $\mathsf{Refresh}$ *which on input $O(n)$* $\mathsf{LWE}_n^{2/q}[\Delta]$ *ciphertexts, refreshes them to* $\mathsf{LWE}_n^{4/q}[\delta]$ *ciphertexts of larger message space and smaller error, using $\tilde{O}(3^{1/\epsilon} n^{1+\epsilon})$ homomorphic operations, for $0 < \epsilon < 1/2$.*

1.2 High level outline

Our scheme involves a number of different parameters. As in the FHEW cryptosystem, we will use a "small" modulus q and dimension $n = 2^{l-1}$ as parameters for the input ciphertexts. (We write $n = 2^{l-1}$ as we will frequently need to refer to $l = \log n + 1$). A larger modulus Q is used by intermediate ciphertexts. We will give a procedure to simultaneously refresh $\varphi(d) = 2 \cdot 3^{k-1}$ FHEW ciphertexts, where $Q > q > n > d$. Details follow.

We start with $\varphi(d)$ (high noise) ciphertexts in $\mathsf{LWE}_n^{2/q}[\Delta]$, as produced by the FHEW $\mathsf{HomNAND}$ operation, working on LWE encryption in dimension n. The key idea required to simultaneously refresh all of them is to first combine them into a single RingLWE ciphertext, in a polynomial ring of degree $\varphi(d)$. Specifically, as a first step, we use a variant of the key switching technique from [8] to evaluate a function

$$\mathsf{PackLWE} : \left[\mathsf{LWE}_n^{2/q}[m_i, \Delta]\right]_{i<\varphi(d)} \to \mathsf{RingLWE}_d^{2/q}[m(X), \Delta'] \tag{1}$$

[2] We will be using primarily only two cyclotomic rings $\mathcal{R}_d = \mathbb{Z}[X]/(X^{d/2}+1) \equiv \mathbb{Z}^{\varphi(d)}$ for $d = 2^k$ and $\varphi(d) = d/2$, and $\mathcal{R}_d = \mathbb{Z}[X]/(X^{2d/3} + X^{d/3} + 1) \equiv \mathbb{Z}^{\varphi(d)}$ for $d = 3^k$ and $\varphi(d) = 2d/3$.

which maps $\varphi(d)$ *arbitrary* LWE ciphertexts (encrypting scalar messages $m_0, \ldots, m_{\varphi(d)-1}$) to a single ciphertext encrypting the polynomial $m(X) = \sum_{i<\varphi(d)} m_i \cdot X^i$. The details of this packing step are given in Section 3.

Looking ahead, the final step of the homomorphic decryption procedure will produce LWE ciphertexts of dimension equal to half the modulus of its input ciphertext. Therefore, to simplify the composition of $\mathsf{HomNAND}$ and $\mathsf{Refresh}$ operations, we use modulus switching after packing to reduce the modulus of the packed ciphertext to $2n$. We use a ring version of the modulus switching technique of [8] to compute a function

$$\mathcal{R}\text{-}\mathsf{ModSwitch}\colon \mathsf{RingLWE}_d^{2/q}[m(x), \Delta'] \to \mathsf{RingLWE}_d^{2/2n}[m(x), \frac{n}{q}\Delta' + \omega(\sqrt{d \log d}) \cdot \|z\|] \tag{2}$$

mapping ciphertexts modulo q under key $z \in \mathcal{R}_d/q$ to ciphertexts modulo $2n$, with the stated error bound (with high probability over the randomized rounding). Details of the modulus switching operation are provided in the full version.

We can now move on to homomorphically decrypt this Ring LWE ciphertext. Following the general bootstrapping framework of [13], refreshing is performed by evaluating the decryption function homomorphically, on an encryption of the secret key. The homomorphic registers encrypting the entries of the secret key are implemented using a symmetric-key/ring variant of the GSW cryptosystem, that we denote abstractly as $\mathsf{REG}^{2n/Q}[\cdot]$. Note that the message modulus $2n$ matches the ciphertext modulus of $\mathsf{RingLWE}_d^{2/2n}$, which is required for entrywise encryption of the $\mathsf{RingLWE}^{2/2n}$ decryption key.

Using this notation, we can describe the refreshing procedure as the combination of two steps. The first is the primary technical contribution of this work, the homomorphic decryption function

$$\mathsf{RingDecrypt} : \mathsf{RingLWE}_d^{2/2n}[m(x), \Delta''] \to \left[\mathsf{REG}^{2n/Q}[\tilde{m}_i, \delta']\right]_{i<\varphi(d)} \tag{3}$$

which takes (as an implicit parameter) the encryption $\mathsf{REG}^{2n/Q}[\mathsf{encode}(z)]$ of a suitably encoded version of the $\mathsf{RingLWE}^{2/2n}$ secret key $z \in \mathbb{Z}_{2n}^{\varphi(d)}$. The $\mathsf{RingDecrypt}$ function is homomorphic in z, and it is computed simply by evaluating the linear component of the $\mathsf{RingLWE}$ decryption function homomorphically on $\mathsf{REG}^{2n/Q}[\mathsf{encode}(z)]$. This entails the multiplication of an encrypted polynomial by a known polynomial, which introduces some technical challenges.

We recall that FHEW accumulators (and the cryptographic registers used in this paper) encode a message $v \in \mathbb{Z}_N$ using a ring variant of the GSW cryptosystem with, as a message space, the set of polynomials in a formal variable X of degree bounded by $\varphi(N)$. These polynomials are used to encode scalar values, mapping each $v \in \mathbb{Z}_N$ to the monomial X^v. Encoding v in the exponent limits the operations available to a candidate refreshing algorithm. Addition may be performed, using the multiplicatively homomorphic property of the GSW cryptosystem to compute $X^v \cdot X^w = X^{v+w}$, but other operations are not so straightforward. Multiplication by (known) scalars (mapping $X^v \mapsto X^{vc}$) requires homomorphic exponentiation, and even a simple subtraction or negation (mapping $X^v \mapsto X^{-v}$) would require homomorphic inversion, all operations unsupported by the GSW or any other known cryptosystem.

Standard FFT algorithms, on the other hand, require both the evaluation of addition and subtractions (in each "butterfly" of the FFT), as well as scalar multiplication by "twiddle" factors, i.e., powers of the root of unity used to compute the FFT. In order to support subtraction, we represent each register in a redundant way, holding both an encryption of X^v

and an encryption of X^{-v}. To reduce the number of scalar multiplications required by our refreshing procedure, we resort to a variant of Nussbaumer negacyclic convolution algorithm [22].

In its original formulation, the Nussbaumer algorithm operates on polynomials in X (where $X^{2^k} = 1$) by writing them as bivariate polynomials in X, Y, with both X and Y of degree at most $\sqrt{n} = \sqrt{2^k}$. Alternatively, one can look at these bivariate polynomials as univariate polynomials in X with coefficients in $\mathbb{Z}[Y]$. By taking $Y = X^{\sqrt{n}}$, the coefficients belong to the ring $R = \mathbb{Z}[Y]/(Y^{\sqrt{n}} + 1)$, which admits Y as a $2\sqrt{n}$th root of unity and can be used to compute the FFT of polynomials in $R[X]$. Multiplication by roots of unity can now be expressed simply by additions, subtractions and rotations of values in $\mathbb{Z}[Y]/(Y^{\sqrt{n}} + 1)$. Furthermore, because the FFT outputs elements of R, the algorithm can be recursively applied to each of the pointwise multiplications following the FFT.

Our refreshing procedure will require some modifications to the Nussbaumer algorithm as described above, which are detailed in Section 4.4. Most critically, if the algorithm is to be used recursively, we must be careful in bounding its recursive depth. The noise of the refreshing algorithm depends exponentially on the depth of the circuit computing it, and we must restrict ourselves to constant depth to achieve polynomial noise overhead. So rather than setting $Y = X^{\sqrt{n_i}}$ for $n_i = \sqrt[2^i]{n}$ at the ith level of recursion, we fix $Y = X^{n^\epsilon}$ for all steps. Therefore, for all ϵ, the algorithm admits at most $1/\epsilon$ recursive calls. This is the main intuition behind our bootstrapping procedure: the Nussbaumer algorithm reduces polynomial multiplication to multiplications of many lower-degree polynomials, without introducing multiplications by constants or excessive noise overhead. A naive multiplication algorithm can then be used to compute the smaller polynomial products, and the transforms inverted to give the encrypted product required for homomorphic ring decryption.

The second step of the refreshing procedure is the "rounding" of the REG ciphertexts to low-noise $\mathsf{LWE}_n^{4/Q}$ ciphertexts. Since $\mathsf{REG}[\cdot]$ encrypts each coefficient of $\tilde{m}(X)$ individually, the values $\mathsf{REG}^{2n/Q}[\tilde{m}_i, \delta']$ are already refreshed, low-noise ciphertexts of the original messages $m_0, \ldots, m_{d-1}$, but using a (noisy) input encoding $\tilde{m}_i$, a different cryptosystem and a large modulus Q. Each one of them is very similar to the intermediate output of the original FHEW refreshing procedure, as if we had computed it on each $\mathsf{LWE}_{2n}^{2/q}[m_i, \Delta]$ ciphertext individualy. So, they can be mapped to $\mathsf{LWE}_{2n}^{4/q}$ ciphertexts as in the original FHEW scheme by calling a "most-significant-bit" extraction function $\mathsf{msbExtract}\colon \mathsf{REG}^{2n/Q}[\tilde{m}_i, \delta'] \to \mathsf{LWE}_n^{4/Q}[m_i, \delta']$ and the standard modulus switching procedure

$$\mathsf{ModSwitch}\colon \mathsf{LWE}_n^{4/Q}[\delta'] \to \mathsf{LWE}_n^{4/q}[\frac{q}{Q}\delta' + \sqrt{2\pi(1 + \|\mathbf{s}\|^2)}] \tag{4}$$

for a LWE ciphertext under key $\mathbf{s} \in \mathbb{Z}_Q^{2n}$, which increases the noise by a small additive term . Parameters will be set in such a way that the resulting noise is small enough to apply the $\mathsf{HomNAND}$ function and keep computing on encrypted data. This completes the high level description of our boootstrapping method.

2 Preliminaries

2.1 Basic notation

We write column vectors over a ring $\mathcal{R}$ with bold font $\mathbf{a} \in \mathcal{R}^n$. Matrices are similarly written in capitalized bold font as $\mathbf{A} \in \mathcal{R}^{n \times m}$. The L^2 norm of a vector $\mathbf{a} = (a_1, \ldots, a_n) \in \mathcal{R}^n$ is $\|\mathbf{a}\| = \sqrt{\sum_i |a_i|^2}$. The concatenation of elements $a, b, \ldots$ into a row vector is written as $[a, b, \ldots]$. We write $(a, b, \ldots)$ for concatenation as a column vector.

2.2 Distributions

A random variable X has *subgaussian* distribution over $\mathbb{R}$ of parameter α if its tails are dominated by a Gaussian of parameter α, so that $\Pr\{|X| \geq t\} \leq 2e^{-\pi t^2/\alpha^2}$ for all $t \geq 0$. A subgaussian variable X with parameter $\alpha > 0$ satisfies $E[e^{2\pi tX}] \leq e^{\pi\alpha^2 t^2}$, for all $t \in \mathbb{R}$. We note that a centered random variable X, where $|X| \leq \beta$ always holds, is subgaussian, specifically with parameter $\beta\sqrt{2\pi}$. For example the randomized rounding function $\lceil(x)\rfloor_\$$ (which takes value $\lfloor x \rfloor$ with probability $\lceil x \rceil - x$, and equals $\lceil x \rceil$ otherwise) is $\sqrt{2\pi}$-subgaussian. A random vector $\mathbf{x}$ of dimension n is subgaussian of parameter α if for all unit vectors $\mathbf{u} \in \mathbb{R}^n$, its one-dimensional marginals $\langle \mathbf{u}, \mathbf{x} \rangle$ are also subgaussian of parameter α. This extends to random matrices, where $\mathbf{X}^{m\times n}$ is subgaussian of parameter α if for all unit vectors $\mathbf{u} \in \mathbb{R}^m, \mathbf{v} \in \mathbb{R}^n$, $\mathbf{u}^t\mathbf{X}\mathbf{v}$ is subgaussian of parameter α. It follows immediately from these definitions that the concatenation of independent subgaussian vectors, all with parameter α, interpreted as either a vector or matrix, is also subgaussian with parameter α.

2.3 Cyclotomic Rings

For any positive integer N, let $\Phi_N(X) = \prod_{j\in\mathbb{Z}_N^*}(X - \omega_N^j)$ be the Nth cyclotomic polynomial, where $\omega_N = e^{2\pi \mathsf{i}/N} \in \mathbb{C}$ is the complex Nth principal root of unity, and $\mathbb{Z}_N$ is the group of invertible integers modulo N. We recall that $\Phi_N(X) \in \mathbb{Z}[X]$ is a monic polynomial of degree $\varphi(N) = |\mathbb{Z}_N^*|$ with integer coefficients. The corresponding ring $\mathcal{R}_N = \mathbb{Z}[X]/\Phi_N(X)$ of integer polynomials modulo Φ_N is called the Nth cyclotomic ring. This ring can be identified with $\mathcal{R}_N \equiv \mathbb{Z}^{\varphi(N)}$ (as additive groups) representing each element $a \in \mathcal{R}_N$ by a polynomial of degree less than $\varphi(N)$, and mapping this polynomial $a(X) = \sum_{j<\varphi(N)} a_j \cdot X^j$ to its coefficient vector $\boldsymbol{(}a\boldsymbol{)} = (a_0, \ldots, a_{\varphi(N)-1}) \in \mathbb{Z}^{\varphi(N)}$. For any ring element $a \in \mathcal{R}_N$, $\|a\|$ is taken to mean the L^2 norm of the corresponding vector $\boldsymbol{(}a\boldsymbol{)} \in \mathbb{Z}^{\varphi(N)}$. Ring elements $a, b \in \mathcal{R}_N$ also admit a matrix representation $\mathbf{M}_a \in \mathbb{Z}^{\varphi(N)\times\varphi(N)} = [\boldsymbol{(}a \cdot X^0\boldsymbol{)}, \boldsymbol{(}a \cdot X^1\boldsymbol{)}, \ldots, \boldsymbol{(}a \cdot X^{\varphi(N)-1}\boldsymbol{)}]$ (used in Section 4.5,) such that $\mathbf{M}_a \cdot \boldsymbol{(}b\boldsymbol{)} = \boldsymbol{(}a \cdot b\boldsymbol{)}$. For any positive integer q, we write $\mathcal{R}_N/q$ for the quotient $\mathcal{R}_N/(q \cdot \mathcal{R}_N)$, i.e., the ring of polynomials $\mathcal{R}_N$ with coefficients reduced modulo q. Notice that $\mathcal{R}_N/q \equiv \mathbb{Z}_q^{\varphi(N)}$ as additive groups.

For concreteness, in this paper we only use cyclotomic rings $\mathcal{R}_N$ for two special types of the index N: $N = n = 2^l$, giving the polynomial ring $\mathcal{R}_n = \mathbb{Z}[X]/(X^{n/2} + 1)$ of degree $\varphi(n) = n/2$, and $N = d = 3^k$, giving the polynomial ring $\mathcal{R}_d = \mathbb{Z}[X]/(X^{2d/3} + X^{d/3} + 1)$ of degree $\varphi(d) = 2d/3$. In particular, $\mathcal{R}_d/q \equiv \mathbb{Z}_q^{2d/3}$ (for $d = 3^k$) and $\mathcal{R}_n/q \equiv \mathbb{Z}_q^{n/2}$ (for $n = 2^l$).

▶ **Fact 2** (Recall from [11], Fact 6)**.** *If $\mathcal{D}$ is a subgaussian distribution of parameter α over $\mathcal{R}_N$, and $\mathbf{R} \leftarrow \mathcal{D}^{w\times k}$ has independent coefficients drawn from $\mathcal{D}$, then, with overwhelming probability, we have $s_1(\mathbf{R}) \leq \alpha\sqrt{N} \cdot O(\sqrt{w} + \sqrt{k} + \omega(\sqrt{\log N}))$.*

2.4 (Ring) LWE Symmetric Encryption

In this subsection we introduce notation and working definitions for the basic LWE encryption scheme that our bootstrapping procedure operates on. For complete definitions, we refer the reader to the full version.

▶ **Definition 3** ((Ring) LWE ciphertexts)**.** The set of all (Ring) LWE ciphertexts over (cyclotomic) ring $\mathcal{R}_N$, encrypting message $m \in \mathcal{R}_N/t$, under key $\mathbf{s} \in \mathcal{R}_N^n$, modulo q and with error bound β is denoted $\mathcal{R}_N\text{-}\mathsf{LWE}_{\mathbf{s}}^{t/q}[m, \beta] = \{(\mathbf{a}, b) \mid \mathbf{a} \in \mathbb{R}_N^n/q, \|\mathbf{a} \cdot \mathbf{s} - mq/t\| \leq \beta\}$. When the value of the key $\mathbf{s} \in \mathcal{R}_N^n$ is clear from the context or unimportant, we simply write $\mathcal{R}_N\text{-}\mathsf{LWE}_n^{t/q}[m, \beta]$, where the subscript n refers to the dimension of the secret $\mathbf{s} \in \mathcal{R}_N^n$.

We use some abbreviated notation in the following important special cases: When $N = 1$, we omit $\mathcal{R}_N = \mathbb{Z}$, and write $\mathsf{LWE}^{t/q}_{\mathbf{s}}[m, \beta]$ (or $\mathsf{LWE}^{t/q}_{n}[m, \beta]$) for the set of standard ($\mathbb{Z}$-) LWE ciphertexts with n-dimensional secret $\mathbf{s} \in \mathbb{Z}^n$. When $n = 1$, and $\mathbf{s} = s \in \mathcal{R}_N$ is a single ring element, we write $\mathsf{RingLWE}^{t/q}_{N}$ as an abbreviation for $\mathcal{R}_N$-$\mathsf{LWE}^{t/q}_{1}$.

The (Ring) LWE decryption procedure plays a central role in our FHE bootstrapping process, specifically in the $\mathsf{RingDecrypt}$ procedure. The decryption of an $\mathcal{R}_N$-$\mathsf{LWE}^{t/q}_{\mathbf{s}}$ ciphertext $(\mathbf{a}, b) \in (\mathcal{R}^n_N, \mathcal{R}_N)/q$ is $\mathsf{Dec}(\mathbf{s}, (\mathbf{a}, b)) = \lfloor t(b - \mathbf{a} \cdot \mathbf{s})/q \rceil \bmod t \in \mathcal{R}_N/t \equiv \mathbb{Z}_t^{\varphi(N)}$. It is easy to check that for all $(\mathbf{a}, b) \in \mathsf{LWE}^{t/q}_{\mathbf{s}}[m, q/2t]$, the decryption procedure correctly recovers the encrypted message.

2.5 Ring-GSW encryption

The cryptographic accumulators of [12] (and the extended cryptographic registers defined in our work) make use of a ring variant of the GSW encryption scheme [19], which we now briefly describe. Let $\mathcal{R}_{N/Q}$ be the Nth cyclotomic ring, modulo some suitably large integer Q. The Ring-GSW cryptosystem, encrypts a message $m \in \mathbb{Z}_N$ under key $z \in \mathcal{R}_{N/Q}$ as $\mathsf{GSW}^{N/Q}_z(m) = [\mathbf{a}, \mathbf{a} \cdot z + \mathbf{e}] + m \cdot \mathbf{g} \otimes \mathbf{I}_2$ where $\mathbf{g} = (B^0, B^1, B^2, \ldots, Q/B)$ for some base B. Similarly as for LWE, we write $\mathsf{GSW}^{N/Q}_z[m, \beta]$ for the set of ciphertexts encrypting m under z with error at most $\|\mathbf{e}\| \leq \beta$. Decryption follows from the observation that the last row of a ciphertext is a Ring-LWE encryption of m under z.

The Ring-GSW cryptosystem supports homomorphic addition and multiplication, and we will primarily use the latter. $\mathsf{GSW}^{N/Q}_z(m_0 \cdot m_1) = C_0 \times C_1$ is computed by first expressing $C_0 = \sum_i B^i C_{0,i}$ as a sum of matrices with B-bounded (polynomial) entries, and then computing the matrix product $[C_{0,0}, \ldots, C_{0,\log Q}] \cdot C_1$. Letting $\mathbf{e}_0$ (resp. $\mathbf{e}_1$) denote the error vector of C_0 (resp. C_1), the result can be written $[\mathbf{a}, \mathbf{a} \cdot z + \mathbf{e}] + m_0 m_1 \cdot \mathbf{g} \otimes \mathbf{I}_2$, where $\mathbf{e} = \sum_i C_{0,i} e_{1,i} + m_1 \mathbf{e}_0$ depends asymmetrically on the error of the inputs. To minimize the error growth resulting from a sequence of multiplications of GSW ciphertexts (with similar initial error), then, the multiplications should be evaluated in a right-associative sequence.

3 Ciphertext Packing

We describe a variant of the LWE key-switching technique that can be used to convert a set of $\varphi(d)$ LWE ciphertexts $\{(\mathbf{a}_i, b_i)\}$, each encrypting a message m_i, to a single "packed" Ring-LWE ciphertext encrypting the message $m(X) = \sum_i m_i X^{i-1}$.

▶ **Lemma 4.** *There exists a quasi-linear time algorithm that on input $\varphi(d)$ ciphertexts $c_i \in \mathsf{LWE}^{t/q}_{\mathbf{s}}(m_i, \Delta)$ (for $i = 0, \ldots, \varphi(d) - 1$, all under the same secret key $\mathbf{s} \in \mathbb{Z}^n_q$) and a packing key consisting of Ring-LWE encryptions $K_{j,l} \in \mathsf{RingLWE}^{q/q}_{\tilde{z}}[s_l 2^j, \beta_P]$ (for $l = 0, \ldots, n-1$, $j = 0 \ldots, \lceil \log q \rceil - 1$ and key $\tilde{z} \in R_{d,q}$,) outputs a Ring-LWE encryption $c \in \mathsf{RingLWE}^{t/q}_{\tilde{z}}[m(X), \beta]$ of $m(X) = \sum_i m_i X^i$ under $\tilde{z}$ with error at most $\beta = O(\sqrt{d}\Delta + \sqrt{dn \log q}\beta_P)$.*

The proof of Lemma 4 and accompanying pseudocode may be found in the full version, but we summarize its conclusion here. Let $K_{j,l} = (\tilde{a}'_{j,l}, \tilde{b}'_{j,l})$ be the entries of the packing key and $c_i = (\mathbf{a}_i, b_i)$ be the input ciphertexts. Defining $\sum_{j < \log q} \tilde{\mathbf{a}}_j 2^j = \sum_i \mathbf{a}_i \cdot X^{i-1} \in \mathcal{R}^n_{d,q}$ and taking $\tilde{a}'' = -\sum_{j,l} \tilde{a}_{j,l} \tilde{a}'_{j,l}$ and $\tilde{b}'' = \tilde{b} - \sum_{j,l} \tilde{a}_{j,l} \cdot \tilde{b}'_{j,l}$ gives the desired packed ciphertext $(\tilde{a}'', \tilde{b}'')$ encrypting $m(X)$.

The homomorphic decryption procedure produces LWE ciphertexts of dimension $n = q/2$ equal to the modulus of the packed ciphertext. This is problematic because on subsequent refreshing operations, the resulting error bound will become larger than $q \log q$, and the

ciphertexts will no longer be decryptable. Before proceeding, we switch to a smaller modulus which we take to be $2n$ for simple composability. The details of the modulus switching procedure and its associated error bounds are omitted here, but provided in the full version of the paper.

4 Homomorphic Decryption

The homomorphic decryption procedure takes as input a single Ring-LWE ciphertext $(a, b) \in \mathsf{RingLWE}_{\tilde{z}}^{t/2n}(m, \beta) \subseteq (\mathcal{R}_d/2n)^2$ and the encryption of some function of the secret key $\tilde{z} \in \mathcal{R}_d/2n$. The decryption procedure needs to compute the ring element $b - a \cdot \tilde{z} \in \mathcal{R}_d/2n$, and then "round" the result to $\varphi(d)$ LWE ciphertexts. All this should be done homomorphically, given the encryption of (some function of) $\tilde{z}$. We can represent the computation of this ring element as an arithmetic circuit C with inputs in $\mathbb{Z}_{2n}$, but to begin designing such a circuit, we must first consider the cryptographic registers on which we will be computing. We begin this section with a description of the registers used in our ciphertext refreshing procedure, to be followed by a description and analysis of the components of our homomorphic decryption procedure.

4.1 Homomorphic Registers

We use a symmetric/ring variant of the GSW cryptosystem to implement the cryptographic registers used by the homomorphic decryption procedure, similar to the accumulators of FHEW. Registers supporting arithmetic modulo $2n$ are implemented using the $\mathsf{GSW}_N^{2n/Q}$ cryptosystem based on the Nth cyclotomic ring, for N a power of 2 with $2n|N$.

We recall that in FHEW, a value $v \in \mathbb{Z}_{2n}$ is represented by $\mathsf{GSW}_N^{2n/Q}(Y^v)$, where $Y = X^i$ is a primitive $2n$th root of unity. In this scheme we take $N = 2n$, and therefore X is our root of unity. To reduce redundancy given this choice of parameters, we omit the subscript N when referring to GSW ciphertexts, writing $\mathsf{GSW}^{2n/Q}$. This choice of parameters is more thoroughly justified in Section 4.5, but as the homomorphic decryption procedure will produce LWE ciphertexts of dimension $N/2$, taking $N/2 = n$, where n is the original dimension of the LWE ciphertexts, allows us to omit an additional step of key switching back to dimenion n.

These GSW registers support the following operations:

- Initialization $(v \leftarrow w)$: $uX^w\mathbf{G}$, with $u \in \mathbb{Z}_Q$, $u \approx Q/2t$, and invertible mod Q.
- Increment $(v \leftarrow v + c)$: $\mathbf{C} \mapsto \mathbf{C} \cdot X^c$
- Addition: $\mathsf{GSW}^{2n/Q}(uX^v) \times \mathsf{GSW}^{2n/Q}(uX^w) = \mathsf{GSW}_z^{2n/Q}(uX^{v+w})$.
- Extraction: map the accumulator to an LWE ciphertext.

To support subtraction, we represent a value $v \in \mathbb{Z}_{2n}$ as a pair $(\mathsf{GSW}(uX^v), \mathsf{GSW}(uX^{-v}))$. Addition is computed componentwise: $(C_0, C_0') + (C_1, C_1') = (C_0 \times C_1, C_0' \times C_1')$. We implement negation simply by swapping the elements of a pair, and subtraction by combining the two operations. This gives us cryptographic registers supporting all operations required by our refreshing algorithm. To avoid explicitly writing these pairs, we define

$$\mathsf{REG}_z^{q/Q}(v, \beta) = (\mathsf{GSW}_N^{q/Q}(uX^v), \mathsf{GSW}_N^{q/Q}(uX^{-v})).$$

4.2 Slow Multiplication

The use of the REG scheme restricts our arithmetic circuits to use only the operations described above. Given the asymmetric error growth of the underlying GSW operations, we must also be careful in how we design our circuit, as we don't want both inputs to any

addition or subtraction gate to have already accumulated significant error. For the rest of this section, however, we omit explicit references to the REG scheme wherever possible, to simplify the presentation of the homomorphic decryption algorithm. We instead use the notation $[\![c]\!]$ to denote a register or registers encrypting value(s) c.

Our goal, then, is to specify an efficient circuit which is parameterized by the input ciphertext (a, b), meets our restrictions, and outputs an encryption of the desired ring element $[\![b - a \cdot \tilde{z}]\!]$ that will then allow each coefficient to be homomorphically rounded to an LWE ciphertext. We discuss the rounding procedure in Section 4.5. In the next few sections, we specify an arithmetic circuit computing $[\![b - a \cdot \tilde{z}]\!]$, where this circuit takes as input a function of the secret key, $[\![f(\tilde{z})]\!]$, and computes the linear decryption step $C_{a,b}([\![f(\tilde{z})]\!]) = [\![b - a \cdot \tilde{z}]\!]$. We will start with a comparatively straightforward, but inefficient, such construction in which we compute $a \cdot \tilde{z}$ homomorphically with a slow multiplication algorithm.

Let $l = \log n + 1$ and $f(\tilde{z}) \in \mathbb{Z}_{2n}^{l \times \varphi(d)}$ be defined by $f(\tilde{z})_{j,k} = \tilde{z}_k 2^j$. Let $a_{i,j}$ be the jth bit of the binary decomposition of a_i so that $a_i = \sum_{j=0}^{l-1} a_{i,j} 2^j$. We may express multiplication of $\tilde{z}_k$ by a_i by computing $\tilde{z}_k \cdot a_i = \sum_{j=0}^{l-1} a_{i,j} \tilde{z}_k 2^j$. Then we define $C_{a_i}([\![f(\tilde{z})]\!], k) = \sum_{j=0}^{l-1} a_{i,j} [\![f(\tilde{z})_{j,k}]\!]$ to be a circuit computing this multiplication homomorphically using only additions, as the $a_{i,j}$ values are binary. We may then define a circuit C_a, computing a slow multiplication algorithm (mod Φ_d) using these C_{a_i} subcircuits, addition, and subtraction gates.

▶ **Lemma 5.** *Let B be the base of the geometric progression defining* $\mathbf{g}$ *in* GSW *encryption, and let $d_b = \lceil \log_B Q \rceil$. There is an algorithm*

$\mathsf{SlowMult} : a \in \mathcal{R}_d/2n \times (\mathsf{REG}_s(f(\tilde{z})_{j,k}, \beta))_{j,k} \to (\mathsf{REG}_s((a \cdot \tilde{z})_i, \beta'))_i$

requiring $\tilde{O}(d^2)$ homomorphic operations, and where $\beta' \leq \tilde{O}(\beta B \sqrt{n d_B d})$ with high probability.

The proof of Lemma 5 appears in the full version, and the analysis of error growth is similar to that of [11]. To briefly justify its stated complexity, we observe that SlowMult is just naive polynomial multiplication algorithm using only additions, and therefore taking time $O(l)$ per scalar multiplication. Therefore its complexity is $\tilde{O}(d^2)$. The original FHEW refreshing procedure requires only $\tilde{O}(n)$ homomorphic additions per ciphertext, so we already see this algorithm offers no improvement over sequential refreshing of d ciphertexts using FHEW. We will instead use variants of existing fast multiplication algorithms for the homomorphic computation of $a \cdot \tilde{z}$, using SlowMult as a subroutine.

4.3 Homomorphic DFT

We briefly recall and introduce notation for the discrete Fourier transform. We denote the discrete Fourier transform of a length m sequence of elements $x \in \mathcal{R}^m$, where ring $\mathcal{R}$ has mth principal root of unity ω_m, by $\bar{x}_i = DFT(x)_i = \sum_{k=0}^{m-1} x_k \omega_m^{ik}$ and its inverse $x_k = DFT^{-1}(\bar{x})_k = \sum_{i=0}^{m-1} \bar{x}_i \omega_m^{-ik}$. The polynomial product $a \cdot \tilde{z}$ for $a, \tilde{z} \in \mathcal{R}_d/2n$ may then be computed as

$$(a * \tilde{z} \mod X^m - 1) \mod \Phi_d = DFT^{-1}(\frac{1}{m} DFT(a) \cdot DFT(\tilde{z})) \mod \Phi_d$$

provided an mth principal root of unity ω_m exists in $\mathcal{R}_d/2n$ (and $m > \frac{4}{3} d \geq deg(a \cdot \tilde{z})$).

But to compute the DFT homomorphically, we need to be able to homomorphically compute multiplication by ω_m^{-ik}. If we take $\omega_m \in \mathbb{Z}_{2n}$, each multiplication by ω_m^{-ik} requires l homomorphic operations per coefficient (as described in Section 4.2). Furthermore, reducing the quadratic complexity of the DFT requires FFT techniques, recursing to depth $\log m$. At each step of recursion, then, we perform homomorphic operations on registers produced from

the last step, with some increase in error from the previous set of operations. We therefore cannot take advantage of the asymmetric error growth of the GSW scheme underlying our registers and the error growth of our algorithm will become quasi-polynomial in n, exceeding the desired polynomial bound on error overhead. This brings us to the last component of the algorithm, which avoids these scalar multiplications, and achieves efficient multiplication without sacrificing polynomial error growth.

4.4 Nussbaumer Transform

In order to efficiently compute the polynomial product $a \cdot \tilde{z} \in \mathcal{R}_d/2n$, we define a variation of the Nussbaumer transform, suited for multiplication of polynomials modulo a power of 3 cyclotomic. Informally, the transform first maps an element $a \in \mathcal{R}_d/2n$ to a bivariate polynomial in such a way that it may be represented by a coefficient vector of dimension smaller than $\varphi(d)$, over $\mathcal{R}_{d^{1-\epsilon}}/2n$. Taking the DFT of this coefficient vector allows us to reduce the computation of $a \cdot \tilde{z}$ to the pointwise multiplication of two vectors with entries in $\mathcal{R}_{d^{1-\epsilon}}/2n$. The inverse map on the inverse DFT of the smaller polynomial products yields $a \cdot \tilde{z} \in \mathcal{R}_d/2n$. We now give a more detailed description of the algorithm.

Let $d = 3^k$, $m = d^\epsilon$ with $d \geq 3m^2$, and $r = \varphi(d)/m = \varphi(d^{1-\epsilon})$. To multiply two polynomials, the transform maps each polynomial to a bivariate polynomial by the isomorphism

$$\psi : \mathbb{Z}_{2n}[X]/(\Phi_d) \to (\mathbb{Z}_{2n}[Y]/(\Phi_{d/m}(Y)))[X]/(X^m - Y)$$
$$a(X) = \sum_{i=0}^{\varphi(d)} a_i X^i \mapsto \sum_{j=0}^{m-1} \sum_{i=0}^{r-1} a_{mi+j} Y^i X^j \text{ where } Y = X^m.$$

Because $\psi(a)$ and $\psi(\tilde{z})$ have degree at most $m-1$ in X, computing $\psi(a) \cdot \psi(b)$ modulo any polynomial of degree greater than $2m-2$ in X prior to reducing by $X^m - Y$ will not change the result. We also note that Y is a principal d/mth root of unity in $\mathbb{Z}_{2n}[Y]/(\Phi_{d/m}(Y))$, and therefore $Y^{d/3m^2} = Y^{3^{k(1-2\epsilon)-1}}$ is a $3m$th root which can also be shown to be principal.

This allows us to efficiently compute $\psi(a) \cdot \psi(\tilde{z})$ first modulo $X^{3m} - 1$ by pointwise multiplication of the respective DFTs, followed by a reduction modulo $(X^m - Y)$. Since the "points" of the DFT pointwise multiplication step are elements of $\mathbb{Z}_{2n}[Y]/(\Phi_{d/m}(Y))$, these multiplications can be performed by recursive application of the transform or SlowMult.

Without recursion, this gives a key preprocessing function

$$f(\tilde{z}) = (1, 2, 4, ..., n) \otimes [\frac{1}{3m} DFT(\psi(\tilde{z}))_i]_{i<3m} \in (\mathcal{R}_{d/m}/2n)^{l \times 3m} \tag{5}$$

where the DFT evaluates $\psi(\tilde{z})$ at root of unity $\omega_{3m} = Y^{d/3m^2}$.

Let $\bar{a}_i = DFT(\psi(a))_i$ be the ith of the $3m$ degree $< r$ polynomials produced by the Nussbaumer transform. Let $C_{\bar{a}_i}$ denote a circuit computing SlowMult of known polynomial $\bar{a}_i$ (of degree $< r$) with an encrypted polynomial given as input (along with encryptions of all power of 2 multiples of the polynomial's coefficients, for $2^l \leq n$, as in Section 4.2). Let C_{F^*} be a circuit homomorphically computing the inverse DFT for length $3m$ vectors of encrypted polynomials in $\mathcal{R}_{d/m}/2n$. Then we may specify a circuit computing $[\![a \cdot \tilde{z}]\!]$

$$C_a([\![f(\tilde{z})]\!] = C_{F^*}([C_{\bar{a}_i}([\![f(\tilde{z})_{(\cdot),i}]\!])]_i) \mod X^m - Y = [\![a \cdot \tilde{z}]\!].$$

$[\![b - a \cdot \tilde{z}]\!]$ is then computed by negating each of the registers $\mathsf{REG}((a \cdot \tilde{z})_i)$ and incrementing each one by the corresponding b_i, computing $C_{a,b}([\![f(\tilde{z})]\!]) = -C_a([\![f(\tilde{z})]\!]) + b$.

The map ψ is purely representational, so requires no computation. The forward and inverse DFT steps require evaluating polynomials at the roots of unity $\omega_{3m}^i = Y^{id/3m^2}$,

which is implemented by rotation of the coefficient vectors with negation, addition, and subtraction to implement the reduction in $\mathcal{R}_{d/m}/2n$. SlowMult was defined using only the operations of addition and subtraction, and the final reduction modulo $X^m - Y$ similarly only requires additions and subtractions. This circuit then satisfies our criteria, allowing for the homomorphic computation of $a \cdot \tilde{z}$ without use of multiplication gates.

The Nussbaumer transform admits a recursive algorithm that gives tradeoffs between runtime and error growth of the cryptographic registers, but we defer consideration of the recursive formulation to Section 4.7.

▶ **Lemma 6.** *Let $\mathcal{K}^R$ be a refreshing key*

$$\mathcal{K}^R = K^R_{i,j,k} = \mathsf{REG}_z^{N/Q}([f(\tilde{z})_{i,j,k}]) = \mathsf{REG}_z^{N/Q}\left[\frac{1}{3m} DFT(\psi(\tilde{z}))_{i,k} 2^j, \beta_R\right] \text{ (from Eq. 5)}$$

where $DFT(\psi(\tilde{z}))_{i,k}$ indicates the kth coefficient of the ith polynomial output by the Nussbaumer transform (giving $i < 3m, j < l, k < r$). For every $0 < \epsilon < \frac{1}{2}$, there is an algorithm RingDecrypt *that on input $\mathcal{K}^R$ and* RingLWE *ciphertext $(a, b) \in \mathsf{RingLWE}_{\tilde{z}}^{2/2n}[m, \beta]$ under $\tilde{z} \in \mathcal{R}_d/2n$, outputs $\varphi(d)$ ciphertexts $\left[\mathsf{REG}_z^{N/Q}[\tilde{m}_i, \beta']\right]_{i<\varphi(d)}$ with $\beta' < \tilde{O}\left(\beta B^4 (n d_B)^{3.5} d^{\epsilon+.5}\right)$, and requiring $\tilde{O}(d^{2-\epsilon})$ homomorphic operations.*

The proof of Lemma 6 may be found in the full version of the paper, but the stated complexity is justified here informally. Lemma 5 states that SlowMult requires $\tilde{O}(d^2)$ operations on input polynomials of degree $O(d)$. The RingDecrypt algorithm multiplies $3m$ polynomials of degree $O(d/m)$, so these multiplications contribute $\tilde{O}(d^{2-\epsilon})$ homomorphic operations. The complexity of the DFT step, which performs $3m$ summations of $3m$ polynomials with degree $O(d/m)$, will then be dominated by that of SlowMult for all permissible ϵ, and therefore RingDecrypt will require $\tilde{O}(d^{2-\epsilon})$ many homomorphic operations.

Once we have performed the RingDecrypt procedure, it remains to round the resulting vector of ciphertexts to yield $\varphi(d)$ refreshed LWE ciphertexts.

4.5 msbExtract

Once we have computed the sequence of registers $[\mathsf{REG}_z^{2n/Q}[b_i - (a \cdot \tilde{z})_i]]_{i<\varphi(d)}$, we must homomorphically perform the rounding step of decryption and recover a sequence of reduced-error LWE ciphertexts encrypting each m_i. This can be accomplished as in FHEW, by applying the msbExtract procedure of [12] to each of the $\varphi(d)$ registers produced by RingDecrypt. As in FHEW, this procedure produces refreshed ciphertexts with the GSW modulus Q, so must be followed by ModSwitch to convert them to the smaller intial modulus q.

We note that our msbExtract procedure differs somewhat from that of FHEW, in that we omit the step of key switching. In FHEW, the procedure takes as additional input a switching key. This key enables the LWE encryptions under key (z) that are recovered at an intermediate stage of the rounding algorithm to be converted to LWE encryptions under the initial key $\mathbf{s}$. Choosing our REG key z such that $(z) = \mathbf{s}$ obviates the need for key switching, as the intermediate LWE ciphertexts will already be encrypted under the proper key. This choice of key requires assuming the security of RingLWE with binary secrets, as assumed in [10], but this assumption may be removed at the cost of the additional key switching step. The proof of Lemma 7 may be found in the full version, and is similar to that of [12].

▶ **Lemma 7.** *There is an algorithm* msbExtract *that, given a cryptographic register of the form $\mathsf{REG}_z^{2n/Q}(b_i - (a \cdot \tilde{z})_i, \beta)$ as input, with $(z) = \mathbf{s}$, outputs a LWE ciphertext $\mathsf{LWE}_{\mathbf{s}}^{4/Q}(m_i, \sqrt{n} \cdot \beta)$.*

4.6 Refreshing Algorithm

We now present the amortized bootstrapping algorithm from start to finish and give an analysis of its runtime and error growth.

The algorithm takes as input $\varphi(d)$ ciphertexts for $d = 3^k$, under the same key $s \in \mathbb{Z}_q^n$ and with error at most Δ, to be simultaneously refreshed. It also requires key material for the PackLWE and RingDecrypt procedures, described in their respective sections, but recalled here for reference. The packing key $\mathcal{K}^P = K^P_{j,l}$ is required to pack the LWE ciphertexts into a single RingLWE ciphertext under a new key $\tilde{z} \in \mathcal{R}_d/q$, and is given by $K^P_{j,l} = \mathsf{RingLWE}^{q/q}_{\tilde{z}}(s_l 2^j, \beta_P)$. The refreshing key $\mathcal{K}^R = K^R_{i,j,k}$ encrypts a function of the RingLWE secret $f(\tilde{z})$ under a REG key $z \in \mathcal{R}_{2n}/Q$, and is required for the homomorphic decryption of the resulting RLWE ciphertext. Its entries are given by $K^R_{i,j,k} = \mathsf{REG}^{2n,Q}_z\left(\frac{1}{3m} DFT(\psi(\tilde{z}))_{i,k}) 2^j, \beta_R\right)$.

▶ **Theorem 8.** *For every $0 < \epsilon < \frac{1}{2}$, there exists an algorithm* Refresh *that, on the input described above, produces $\varphi(d)$* LWE *ciphertexts with error $\tilde{O}\left(\|s\| + \frac{q}{Q} \cdot \beta_R (Bn)^4 d_B^{3.5} d^{\epsilon+.5}\right)$, and requires $\tilde{O}(d^{2-\epsilon})$ homomorphic operations.*

Proof. From Lemma 6, we already have that the homomorphic complexity of RingDecrypt is $\tilde{O}(d^{2-\epsilon})$, and this dominates the complexity of RingDecrypt.

The correctness of this refreshing scheme will rely on the error bounds at two stages of the algorithm. For msbExtract to recover the correct m_i's from the output of RingDecrypt, the error of each $\tilde{m}_i = \frac{2n}{t} m_i + e$ must not exceed $\frac{n}{t}$ (for $t = 2$, the message space of m_i).

From the error bounds of Theorem 4 and Equation 2, we have that the ciphertext output by $\mathsf{ModSwitch}_{\mathcal{R}}$ will have error bounded by $O\left(\frac{2n}{q}(\sqrt{d}\Delta + \sqrt{dn \log q}\beta_P) + \omega(\sqrt{d \log d}) \cdot \|\tilde{z}\|\right)$. To bound this by $n/2$, we restrict $\tilde{z}$ to $\|\tilde{z}\| = O(\sqrt{d})$. Then so long as the LWE ciphertext error Δ satisfies $\Delta < O(\frac{q}{\sqrt{d}})$, $d\sqrt{\log d} < O(n)$, and $d^2 n < O(\frac{q^2}{\log q})$, the packed ciphertexts will be decryptable with high probability.

We also need to guarantee that the ciphertexts output by Refresh will have error small enough that this scheme is composable. From the bounds of Lemma 6, Lemma 7, and Equation 4, this requires us to bound the error of the ciphertexts output by Refresh by $\beta' = \tilde{O}\left(\|s\| + \frac{q}{Q} \cdot \beta_R (nB)^4 d_B^{3.5} d^{\epsilon+.5}\right) < \frac{q}{\sqrt{d}}$. Letting s be a binary secret, (which does not reduce hardness of the associated LWE instance, as shown in [7]), this gives us that $\frac{Q}{(d_B)^{3.5}} > \beta_R n^4 \sqrt{\log n} B^4 d^{1+\epsilon}$, so taking $B = \Theta(1)$ and $Q > \tilde{O}(\beta_R n^4 d^{1+\epsilon} (\log d)^{3.5})$ will guarantee correct decryption with high probability. ◀

4.7 Recursive optimization

In this section we summarize a recursive formulation of the Nussbaumer transform, and how it can improve the complexity of the RingDecrypt algorithm, at the cost of an increase in the error. The proof of Theorem 9 may be found in the full version.

The Nussbaumer transform as described in Section 4.4 can be thought of as a reduction from a single multiplication of two polynomials in $\mathbb{Z}_{2n}[X]/(\Phi_d(X))$ to $3m$ multiplications of pairs of polynomials in $\mathbb{Z}_{2n}[Y]/(\Phi_{d/m}(Y))$. We may recursively apply this transformation ρ times, provided we have a $3m$th root of unity in the ring $\mathbb{Z}_{2n}[Y]/(\Phi_{d/m^\rho}(Y))$, which will be the case as long as $d/m^\rho \geq 3m$. ϵ is fixed and so this bounds the recursive depth of the algorithm by $\rho < \frac{1}{\epsilon} - 1$.

▶ **Theorem 9.** *The recursive $\mathsf{RingDecrypt}_\rho$ algorithm, with constant parameter ϵ and recursive depth $\rho < \frac{1}{\epsilon} - 2$, requires $\tilde{O}(3^\rho d^{2-\rho\epsilon} + 3^\rho d^{1+\epsilon})$ homomorphic operations and yields an error growth of $\tilde{O}(B^{3\rho+1}(nd_B)^{3\rho}\sqrt{nd_B d^{1+\rho\epsilon}})$.*

References

1 Jacob Alperin-Sheriff and Chris Peikert. Practical bootstrapping in quasilinear time. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 1–20. Springer, Heidelberg, August 2013. doi:10.1007/978-3-642-40041-4_1.

2 Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 297–314. Springer, Heidelberg, August 2014. doi:10.1007/978-3-662-44371-2_17.

3 Jean-François Biasse and Luis Ruiz. FHEW with efficient multibit bootstrapping. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015: 4th International Conference on Cryptology and Information Security in Latin America*, volume 9230 of *Lecture Notes in Computer Science*, pages 119–135. Springer, Heidelberg, August 2015. doi:10.1007/978-3-319-22174-8_7.

4 Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, Heidelberg, August 2012.

5 Zvika Brakerski, Craig Gentry, and Shai Halevi. Packed ciphertexts in LWE-based homomorphic encryption. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 1–13. Springer, Heidelberg, February / March 2013. doi:10.1007/978-3-642-36362-7_1.

6 Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 309–325. Association for Computing Machinery, January 2012.

7 Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 575–584. ACM Press, June 2013.

8 Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.*, 43(2):831–871, 2014. URL: https://doi.org/10.1137/120868669, doi:10.1137/120868669.

9 Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In Moni Naor, editor, *ITCS 2014: 5th Innovations in Theoretical Computer Science*, pages 1–12. Association for Computing Machinery, January 2014.

10 Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 3–33. Springer, Heidelberg, December 2016. doi:10.1007/978-3-662-53887-6_1.

11 Léo Ducas and Daniele Micciancio. Improved short lattice signatures in the standard model. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 335–352. Springer, Heidelberg, August 2014. doi:10.1007/978-3-662-44371-2_19.

12 Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 617–640. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46800-5_24.

13 Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178. ACM Press, May / June 2009.

14 Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 107–109. IEEE Computer Society Press, October 2011.

15 Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P. Smart. Ring switching in BGV-style homomorphic encryption. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12: 8th International Conference on Security in Communication Networks*, volume 7485 of *Lecture Notes in Computer Science*, pages 19–37. Springer, Heidelberg, September 2012.

16 Craig Gentry, Shai Halevi, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 1–16. Springer, Heidelberg, May 2012.

17 Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, Heidelberg, April 2012.

18 Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, Heidelberg, August 2012.

19 Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, Heidelberg, August 2013. doi:10.1007/978-3-642-40041-4_5.

20 Shai Halevi and Victor Shoup. Algorithms in HElib. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 554–571. Springer, Heidelberg, August 2014. doi:10.1007/978-3-662-44371-2_31.

21 Shai Halevi and Victor Shoup. Bootstrapping for HElib. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 641–670. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46800-5_25.

22 Henri J. Nussbaumer. Fast polynomial transform methods for multidimensional dfts. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '80, Denver, Colorado, April 9-11, 1980*, pages 235–237. IEEE, 1980. URL: https://doi.org/10.1109/ICASSP.1980.1170902, doi:10.1109/ICASSP.1980.1170902.

23 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93. ACM Press, May 2005.

24 Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptography*, 71(1):57–81, 2014. URL: https://doi.org/10.1007/s10623-012-9720-4, doi:10.1007/s10623-012-9720-4.

Semi-random Graphs with Planted Sparse Vertex Cuts: Algorithms for Exact and Approximate Recovery

Anand Louis
Indian Institute of Science
Bangalore, India
anandl@iisc.ac.in

Rakesh Venkat[1]
Hebrew University of Jerusalem
Israel
rakesh@cs.huji.ac.il

Abstract

The problem of computing the vertex expansion of a graph is an NP-hard problem. The current best worst-case approximation guarantees for computing the vertex expansion of a graph are a $\mathcal{O}\left(\sqrt{\log n}\right)$-approximation algorithm due to Feige et al. [16], and $\mathcal{O}\left(\sqrt{\mathsf{OPT}\log d}\right)$ bound in graphs having vertex degrees at most d due to Louis et al. [29].

We study a natural semi-random model of graphs with sparse vertex cuts. For certain ranges of parameters, we give an algorithm to recover the planted sparse vertex cut exactly. For a larger range of parameters, we give a constant factor bi-criteria approximation algorithm to compute the graph's balanced vertex expansion. Our algorithms are based on studying a semidefinite programming relaxation for the balanced vertex expansion of the graph.

In addition to being a family of instances that will help us to better understand the complexity of the computation of vertex expansion, our model can also be used in the study of community detection where only a few nodes from each community interact with nodes from other communities. There has been a lot of work on studying random and semi-random graphs with planted sparse edge cuts. To the best of our knowledge, our model of semi-random graphs with planted sparse vertex cuts has not been studied before.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Semi-Random models, Vertex Expansion, Approximation Algorithms, Beyond Worst Case Analysis

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.101

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.09747`.

Acknowledgements We thank Amit Deshpande for many helpful discussions. Rakesh Venkat was visiting Microsoft Research, Bangalore when this research was initiated. Anand Louis is grateful to MSR, Bangalore for supporting this collaboration.

[1] Supported by an I-CORE Algorithms Fellowship

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 101; pp. 101:1–101:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Given a graph $G = (V, E)$, the vertex expansion of a non-empty subset $S \subset V$, denoted by $\phi^{\mathsf{V}}(S)$, is defined as[2]

$$\phi^{\mathsf{V}}(S) \stackrel{\text{def}}{=} |V| \frac{|N(S)| + |N(V \setminus S)|}{|S|\,|V \setminus S|},$$

where $N(S)$, the neighborhood of S, is defined as $N(S) \stackrel{\text{def}}{=} \{j \in V \setminus S : \exists i \in S \text{ such that } \{i, j\} \in E\}$. The vertex expansion of the graph G, denoted by ϕ^{V}_G, is defined as $\phi^{\mathsf{V}}_G \stackrel{\text{def}}{=} \min_{S \subset V,\ S \neq \emptyset} \phi^{\mathsf{V}}(S)$. Computing the vertex expansion of a graph is NP-hard. The complexity of computing various graph expansion parameters are central open problems in theoretical computer science, and inspite of many decades of intensive research, they are yet to be fully understood [8, 9, 26, 10, 16, 39].

Feige et al. [16] gave a $\mathcal{O}\left(\sqrt{\log n}\right)$-approximation algorithm for computing the vertex expansion of a graph. Louis et al. [29] gave an algorithm that computes a set having vertex expansion at most $\mathcal{O}\left(\sqrt{\phi^{\mathsf{V}} \log d}\right)$ in graphs having vertex degrees at most d. We give a brief description of other related works in Section 1.3. In this work, we study a natural semi-random family of graphs, and give polynomial time exact and approximation algorithms for computing the *balanced vertex expansion* (a notion that is closely related to the vertex expansion of a graph, we define it formally in Section 1.1) w.h.p.

In many problems, there is a huge gap betwen theory and practice; the best known algorithms provide a somewhat underwhelming performance guarantee, however simple heuristics perform remarkably well in practice. Examples of this include the simplex algorithm for linear programming [25], SAT [11], sparsest cut [22, 23], among others. In many cases, the underwhelming provable approximation guarantee of an algorithm is a property (hardness of approximation) of the problem itself; even in many such cases, simple heuristics work remarkably well in practice. A possible explanation for this phenomenon could be that for many problems, the instances arising in practice tend to have some inherent structure that makes them "easier" than the worst case instances. Many attempts have been made to understand the structural properties of these instances, and to use them in designing algorithms specifically for such instances, which could perform much better than algorithms for general instances. A fruitful direction of study has been that of modelling real world instances as a family of random and semi-random instances satisfying certain properties. Our work can be viewed as the study of the computation of vertex expansion along this direction.

Often graphs with sparse cuts are used to model communities. For example, the vertices of a graph can be used to represent the members of the communities, and two vertices would have an edge between them if the members corresponding to them are *related* in some way. In such a graph, the sparse cuts indicate the presence of a small number of relations across the members corresponding to the cut, which are likely to be some form of communities within the members. The *stochastic block models* have been used to model such communities. Our model can also be viewed as model for communities where only a few members from each community have a relationship with members from another community.

[2] Other definitions of vertex expansion have been studied in the literature, see Section 1.3.

1.1 Vertex Expansion Block Models

For a graph $G = (V, E)$, its *balanced vertex expansion* $\phi^{\mathsf{V-bal}}$ is defined as

$$\phi_G^{\mathsf{V-bal}} \stackrel{\text{def}}{=} \min_{\substack{S \subset V \\ |S|=|V|/2}} \phi^{\mathsf{V}}(S)\,.$$

Another common notion of vertex expansion that has been studied in the literature is $\phi^{\mathsf{V,a}}(S) \stackrel{\text{def}}{=} (|V|\,|N(S)|\,/\,(|S|\,|V \setminus S|))$, and as before, $\phi_G^{\mathsf{V,a}} \stackrel{\text{def}}{=} \min_{S \subset V} \phi^{\mathsf{V,a}}(S)$. [29] showed that the computation ϕ_G^{V} and $\phi_G^{\mathsf{V,a}}$ is equivalent upto constant factors. In this work, we develop a semi-random model for investigating the balanced vertex expansion of graphs.

We study instances that are constructed as follows. We start with a set of n vertices, and we arbitrarily partition them into two sets S, S' of $n/2$ vertices each. Next, we choose a small subset $T \subset S$ of size εn (resp. $T' \subset S'$) to form the vertex boundary of these sets. On $S \setminus T$ (resp. $S' \setminus T'$), we add an arbitrary graph whose spectral gap[3] is at least λ (a parameter in this model), and whose vertices have roughly the same degree. We add an arbitrary low degree bipartite graph between T and T'. Between each pair of vertices in $(S \setminus T) \times T$, we add edges independently at random with probability p; this is the only part of the construction that is random. Next, we allow a *monotone adversary* to alter the graph : the monotone adversary can arbitrarily add edges that do not change the sparsity of the vertex cut (S, S'), i.e., add edges between any pair of vertices in S (resp. S'), and between any pair in $T \times T'$.

In our model, we allow the sets S and S' to be generated using different sets of parameters, i.e., we use $\varepsilon_1, \lambda_1, p_1$ for S and $\varepsilon_2, \lambda_2, p_2$ for S'. We formally define the vertex expansion block model below (see also Figure 1).

▶ **Definition 1.** An instance of $\mathsf{VBM}(n, \varepsilon_1, \varepsilon_2, p_1, p_2, c, r, \lambda_1, \lambda_2)$ is generated as follows.

1. Let V be a set of n vertices. Partition V into two sets S and S' of $n/2$ vertices each. Partition S into two sets T and $S \setminus T$ of sizes $\varepsilon_1 n$ and $(1/2 - \varepsilon_1)n$ respectively. Similarly, partition S' into two sets T' and $S' \setminus T'$ of sizes $\varepsilon_2 n$ and $(1/2 - \varepsilon_2)n$ respectively.
2. Between each pair in $(S \setminus T) \times T$ (resp. $(S' \setminus T') \times T'$), add an edge independently with probability p_1 (resp. p_2).
3. Between pairs of vertices in $S \setminus T$ (resp. $S' \setminus T'$), add edges to form an arbitrary roughly regular (formally, ratio of the maximum vertex degree and the minimum vertex degree is at most r) of spectral gap[3] at least λ.
4. Between pairs in $T \times T'$, add edges to form an arbitrary bipartite graph of vertex degrees in the range $[1, c]$ (this bipartite graph need not be connected); if $c < 1$, then add no edges in this step. We will use $\mathcal{F}$ to denote this bipartite graph.
5. (*Monotone Adversary*) Arbitrarily add edges between any pair of vertices in S (resp. S'). Arbitrarily add edges between any pair in $T \times T'$.

Output the resulting graph G.

We note that the direct analogue for vertex expansion of *Stochastic Block Models* (see related work in Section 1.3) in the regimes allowing for exact recovery is included in this setting: there, the graphs within S and S' are completely random, and so are the connections between T and T' (before the monotone adversary acts). Our model allows for a lot more adversarial action, while restricting the randomness to only a small portion of the graph.

[3] The spectral gap of a graph is defined as the second smallest eigenvalue of its normalized Laplacian matrix, see Section 1.6 for definition

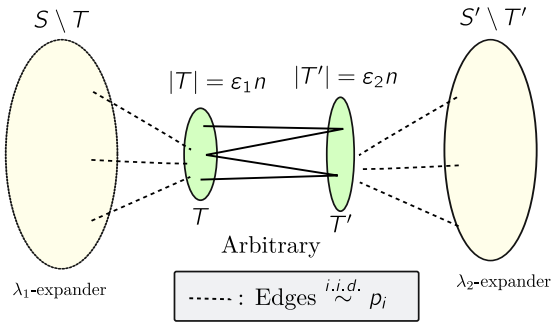


Figure 1 $\mathsf{VBM}(n, \varepsilon_1, \varepsilon_2, p_1, p_2, c, r, \lambda_1, \lambda_2)$. A monotone adversary may further add arbitrary edges within S, S' and between T, T'.

In addition to being a family of instances that will help us to better understand the complexity of the computation of vertex expansion, the vertex expansion block model can also be used in the study of community detection. In the case of two communities, the vertices in S and S' can model the members of the communities. Each community can have a few representatives who interact with the representatives from other communities; these representatives can be modelled using T and T', and their interactions can be modelled by the arbitrary graphs within T and T', and the low degree bipartite graph and the action of the monotone adversary between T and T'. Even though the connections within a community may be arbitrary, usually the members within the community are well connected with each other; this can be modelled by the choosing an appropriate values of λ_1, λ_2 plus the action of the monotone adversary. We can model the connections between community members and their representatives by a sparse random bipartite graph; our model allows the flexibility of choosing p_1 and p_2, and also the action of the monotone adversary.

1.2 Our Results

Our main result is a polynomial time algorithm for exactly recovering S and S' from a graph sampled from $\mathsf{VBM}(n, \varepsilon_1, \varepsilon_2, p_1, p_2, c, r, \lambda_1, \lambda_2)$ for certain ranges of parameters.

▶ **Theorem 2.** *There exist universal constants $c_1 \in \mathbb{R}^+, c_2 \in (0, 1/2), c_3 \in \mathbb{Z}^+, c_4 \in \mathbb{R}^+, c_5 \in (0, 1)$ satisfying the following: there exists a polynomial time algorithm which takes a graph generated from $\mathsf{VBM}(n, \varepsilon_1, \varepsilon_2, p_1, p_2, c, r, \lambda_1, \lambda_2)$, where $p_1, p_2 \in (c_1(\log n)/n, 1]$, $\varepsilon_1, \varepsilon_2 \in [1/n, c_2]$, $c \leqslant c_3$, $r \leqslant c_4$ and $\lambda_1, \lambda_2 \geqslant c_5$, and outputs the sets S and S' with probabilty at least $1 - 1/\mathsf{poly}(n)$.*

We give a description of the steps in proving Theorem 2 in Section 2; in the full version of this paper, we also observe that our proof actually shows a slightly more general result.

We also show that if the instances satisfy a few weaker requirements, then we can obtain a constant factor bi-criteria approximation algorithm for computing the balanced vertex expansion.

We study the case when $S \setminus T$ is an arbitrary graph, i.e., it does not have constant spectral gap. Note that this case is captured by setting $\lambda_1 = 0$ in our model, since the monotone adversary can create any arbitrary graph on $S \setminus T$. Our proof also allows us to let the graph induced on S' be an arbitrary graph. Again, this is captured by setting $p_2 = \lambda_2 = 0$ in our model, since the monotone adversary can create any arbitrary graph on S'. We show that we can use the underlying random bipartite graph between $S \setminus T$ and T to obtain a bi-criteria approximation algorithm in this case, that outputs an almost balanced cut with vertex expansion within a constant factor of the planted one.

▶ **Theorem 3.** *There exist universal constants $c_1 \in \mathbb{R}^+$, $c_2 \in (0,1/2), c_6 \in (0,1/2)$ satisfying the following: there exists a polynomial time algorithm which takes a graph generated from $\mathsf{VBM}(n, \varepsilon_1, \varepsilon_2, p_1, 0, 0, 0, 0, 0)$, where $\varepsilon_1, \varepsilon_2 \in [1/n, c_2]$ and $p_1(\varepsilon_1+\varepsilon_2)n \geqslant c_1 \log n$, and outputs with probability at least $1 - 1/\mathsf{poly}(n)$, a set $A \subset V$ satisfying $|A| \in [c_6 n, (1-c_6)n]$ and $\phi^{\mathsf{V}}(A) \leqslant \mathcal{O}(\varepsilon_1 + \varepsilon_2)$.*

Next, we study the case where the edges between $S \setminus T$ and T are arbitrary, but λ_1 is large. As in the previous case, our proof allows the graph induced on S' to be an arbitrary graph. Again, as before, this case is captured by setting $p_1 = p_2 = \lambda_2 = 0$. In this case, we show that for certain ranges of λ_1, we can obtain a constant factor bi-criteria approximation algorithm for computing the balanced vertex expansion.

▶ **Theorem 4.** *There exist universal constants $c_2 \in (0,1/2), c_5 \in (0,1), c_6 \in (0,1/2)$ satisfying the following: there exists a polynomial time algorithm which takes a graph generated from $\mathsf{VBM}(n, \varepsilon_1, \varepsilon_2, 0, 0, 0, r, \lambda_1, 0)$, where $\varepsilon_1, \varepsilon_2 \in [1/n, c_2]$, and $\lambda_1 \geqslant c_5 r^3(\varepsilon_1 + \varepsilon_2)$, and outputs a set $A \subset V$ satisfying $|A| \in [c_6 n, (1-c_6)n]$ and $\phi^{\mathsf{V}}(A) \leqslant \mathcal{O}(\varepsilon_1 + \varepsilon_2)$.*

In fact, we prove a stronger result: it suffices for $S \setminus T$ to contain a subgraph on $\Omega(n)$ vertices having spectral gap at least $c_5 r^3(\varepsilon_1 + \varepsilon_2)$, to obtain a constant factor bi-criteria approximation algorithm for computing the balanced vertex expansion.

Organization of the rest of the paper

We will next describe related work in Section 1.3, and give our SDP relaxation in Section 1.4. We give an overview of our proofs in Section 1.5, and present a slightly more detailed description of the proof of our main result, Theorem 2, in Section 2. For the proofs of Theorem 3 and Theorem 4, we refer the reader to the full version of the paper.

1.3 Related Work

Stochastic Block Models

Closely related to the vertex expansion of a graph is the notion of edge expansion which is defined as follows.

▶ **Definition 5.** For a weighted graph $G = (V, E, w)$, with non-negative edge weights $w : E \to \mathbb{Q}^+$, the edge expansion of a non-empty set $S \subset V$ is defined as

$$\phi_G(S) \stackrel{\text{def}}{=} \frac{\sum_{e \in E(S, V\setminus S)} w(e)}{\min\{\mathsf{vol}(S), \mathsf{vol}(V \setminus S)\}},$$

where $E(S, V \setminus S) \stackrel{\text{def}}{=} \{\{i,j\} \in E : i \in S, j \notin S\}$ and $\mathsf{vol}(S) \stackrel{\text{def}}{=} \sum_{i \in S} \sum_{j \sim i} w(\{i,j\})$. The edge expansion of the graph is defined as $\phi_G \stackrel{\text{def}}{=} \min_{S \subset V,\ S \neq \emptyset} \phi_G(S)$.

The *Stochastic Block Model* (we will refer to it as the *edge expansion stochastic block model* to differentiate it from our block model) is a randomized model for instances that are generated as follows. A set of n vertices is arbitrarily partitioned into sets S, S' of equal sizes. Between each pair of vertices in S, an edge is added independently with probability p, and between each pair of vertices in $S \times S'$, an edge is added independently with probability q (typically $p > q$).

Starting with work of Holland et al.[20], the works of Boppana [13], who gave a spectral algorithm, and of Jerrum and Sorkin [21], who gave a metropolis algorithm, contributed

significantly to the study of stochastic block models. One of the break through works in the study of SBMs is the work of McSherry [34], who gave a simple spectral algorithm for a certain range of parameters. There has been a lot of recent work related to a certain conjecture regarding SBMs, which stated the regime of parameters p, q for which it was possible to detect the presence of communities. The works [36, 37, 38, 33] have contributed to proving various aspects of the conjecture. In a recent work, Abbe et al.[3] showed that the natural SDP relaxation for balanced edge expansion is integral when there is a sufficient gap between p and q, and $p, q = \Omega(\log n/n)$; Mossel et al.[38] gave an algorithm for a larger regime of parameters which was not based on semidefinite programming. More general SBMs have been studied by Abbe and Sandon [4, 5, 6], Aggarwal et al.[7], etc.

Kim et al.[24] studied a version of SBM for hypergraphs, and gave algorithms for it based on studying a certain "adjacency tensor", the analog of the adjacency matrix for hypergraphs. They also study the sum-of-squares algorithms for this model. [27] gave a reduction from vertex expansion problems to hypergraph expansion problems. We note that applying this reduction to the instances from our models does not give the model studied by [24]: this reduction will only introduce hyperedges between the sets corresponding to T and T', whereas the model studied by [24] adds random hyperedges between S and S'. Moreover, many parts of the graph from our model are adversarially chosen.

Semi-random models for edge expansion problems

Monotone adversarial errors in SBMs are the arbitrary addition of edges between pairs of vertices within S (resp. S'), and the arbitrary deletion of existing edges between S and S'. Feige and Kilian [17] gave an algorithm for the edge expansion model with monotone adversarial errors when the gap between p and q is sufficiently large. Guedon and Vershynin [18] gave an algorithm based on semidefinite programming for partially recovering the communities for certain ranges of parameters. Moitra et al.[35] gave algorithms (based on semidefinite programming) and lower bounds for partial recovery in the stochastic block model with a monotone adversary. Makarychev et al.[32] gave an algorithm for partial recovery for the stochastic block model with a monotone adversarial errors and a small number of arbitrary errors (i.e. non-monotone errors).

Makarychev et al.[30, 31] studied some semi-random models of instances for edge expansion problems. In particular, [30] studied a model analogous to $\mathsf{VBM}(n, \varepsilon_1, \varepsilon_2, 0, 0, 0, r, \lambda_1, 0)$for edge expansion problems; they showed that if the number of edges crossing (S, S') is εm, and if there is a set of m edges E_1 such that (S, E_1) is a regular graph having spectral expansion at least $\Omega(\varepsilon)$, then there is an algorithm to recover a balanced cut of edge expansion $\mathcal{O}(\varepsilon)$. The proof of Theorem 4 and that of the corresponding result in [30] both proceed by using the expansion of the underlying subgraph to show that an $\Omega(n)$ sized subset of the SDP vectors lie in a ball of small radius. [30] use this to recover a constant factor bi-criteria approximation to balanced edge expansion; we adapt this approach to vertex expansion to prove Theorem 4.

The results cited here are only a small sample of the work on the SBMs. Since our model is very different from the edge expansion stochastic block models, we only give a brief survey of the literature here, and we refer the reader to a survey by Abbe [1] for a comprehensive discussion. In general, algorithms for edge expansion problems can not be used for our vertex expansion block model since sparse edge cuts and sparse vertex cuts can be uncorrelated. In particular, the action of the monotone adversary in VBM rules out the use of edge-expansion based algorithms for detecting S and S'. In the full version of this paper, we describe an explicit family graphs generated in the VBM model where these algorithms fail.

Vertex Expansion

There has been some work in investigating vertex expansion (balanced and non-balanced) the worst-case setting. Bobkov et al. [12] gave a Cheeger-type inequality for vertex expansion, where a parameter λ_∞ plays a role analogous to the use of the second eigenvalue λ_2 in Cheeger's inequality for edge expansion. Feige et al.[16] gave a $\mathcal{O}\left(\sqrt{\log n}\right)$-approximation algorithm for the problem of computing the vertex expansion of graphs. Louis et al.[29] gave an SDP rounding based algorithm that computes a set having vertex expansion at most $\mathcal{O}\left(\sqrt{\phi_G^{\mathsf{V}} \log d}\right)$, where d is the maximum vertex degree; they also showed a matching hardness result based on the Small-set expansion hypothesis. Louis and Makarychev [27] gave a bi-criteria approximation for Small-set vertex expansion, a problem related to vertex expansion. Chan et al. [14] studied various parameters related to hypergraphs, including parameters related to hypergraph expansion; they showed that many of their results extend to the corresponding vertex expansion analogues on graphs

Louis and Raghavendra [28] studied a model of instances for vertex expansion similar to ours. In their model, the adversary partitions the vertex set V into two equal sized sets S, S', and chooses a subset T (resp. T') of S (resp. S') of size at most εn. Next, the adversary chooses an arbitrary subset of pairs of vertices in S (resp. S') to form edges such that graph induced on S (resp. S') is an edge expander. The adversary chooses an arbitrary subset of the pairs of vertices in $T \times T'$ to form edges. [28] give an SDP rounding based algorithm to compute a set having vertex expansion $\mathcal{O}\left(\sqrt{\varepsilon}\right)$.

1.4 SDP Relaxation

We use the SDP relaxation for $\phi_G^{\mathsf{V}-\mathsf{bal}}$ (SDP 6), this SDP is very similar to that of [29]. We give the dual of this SDP in SDP 7.

▶ **SDP 6** (Primal).

$$\min \sum_{i \in V} \eta_i$$

subject to

$$\begin{aligned} U_{ii} + U_{jj} - 2U_{ij} &\leqslant \eta_i && \forall i \in V, j \in N(i) \\ U_{ii} &= 1 && \forall i \in V \\ \sum_{i \in V} \sum_{j \in V} U_{ij} &= 0 \\ U &\succeq 0 \end{aligned}$$

▶ **SDP 7** (Dual).

$$\max \sum_{i \in V} B_{ii}$$

subject to

$$\begin{aligned} \sum_{j \in N(i)} Y_{ij} &= 1 \\ Y_{ij} &\geqslant 0 && \forall \{i,j\} \in E \\ B_{ij} &= 0 && \forall i,j \in V, i \neq j \\ L(Y) + \alpha \mathbb{1}\mathbb{1}^T - B &\succeq 0 \end{aligned}$$

Here $\mathbb{1}$ denotes the all-ones vector, and $L(Y)$ denotes the Laplacian matrix of graph weight by the matrix $Y + Y^T$, i.e.

$$L(Y)_{ij} = \begin{cases} \sum_{l \in N(i)} (Y_{il} + Y_{li}) & i = j \\ -(Y_{ij} + Y_{ji}) & j \in N(i) \\ 0 & \text{otherwise} \end{cases}.$$

First, let us see why SDP 6 is a relaxation for $\phi^{\mathsf{V}-\mathsf{bal}}$. Let P be the set corresponding to $\phi_G^{\mathsf{V}-\mathsf{bal}}$, and let $\mathbb{1}_P \in \{-1, 1\}^n$ be a vector such $\mathbb{1}_P(i)$ is equal to 1 if $i \in P$ and -1 otherwise. Note that since $|P| = |V|/2$, we have $\mathbb{1}^T \mathbb{1}_P = 0$. It is easy to verify that

$U := \mathbb{1}_P \mathbb{1}_P^T$ and $\eta_i := \max_{j \in N(i)} \left(\mathbb{1}_P(i) - \mathbb{1}_P(j)\right)^2$ is a feasible solution for SDP 6, and that $\sum_{i \in V} \eta_i = 4\left(|N(P)| + |N(V \setminus P)|\right)$. Therefore, $\phi_G^{\mathsf{V-bal}} = 4\left(\sum_{i \in V} \eta_i\right)/n$. and therefore, SDP 6 is a relaxation for $\phi^{\mathsf{V-bal}}$. Henceforth, we will use $\mathbb{1}_P$ to be the ± 1-indicator vector of a set $P \subset V$, i.e., $\mathbb{1}_P(i)$ is equal to 1 if $i \in P$ and -1 otherwise. We prove the following theorem about SDP 6.

▶ **Theorem 8.** *For the regime of parameters stated in Theorem 2,* $U' \stackrel{\text{def}}{=} \mathbb{1}_S \mathbb{1}_S^T$ *and for each* i, $\eta_i' \stackrel{\text{def}}{=} \max_{j \in N(i)} \left(\mathbb{1}_S(i) - \mathbb{1}_S(j)\right)^2$ *for the set* S *defined in* $\mathsf{VBM}(n, \varepsilon_1, \varepsilon_2, p_1, p_2, c, r, \lambda_1, \lambda_2)$*, is the unique optimal solution to SDP 6 with probabilty at least* $1 - 1/\mathsf{poly}(n)$.

Theorem 8 gives an algorithm to compute the matrix $\mathbb{1}_S \mathbb{1}_S^T$. By factorizing this matrix, one can obtain the vector $\mathbb{1}_S$, using which the set S can be computed. Therefore, Theorem 8 implies Theorem 2.

1.5 Proof Overview

1.5.1 Theorem 2

It is easy to verify that (U', η_i') is a feasible solution to SDP 6. Our goal will be to construct a dual solution (i.e. a feasible solution to SDP 7) which satisfies two properties,

1. The cost of this solution should be same as the cost of this primal solution (U', η').
2. The matrix $\left(L(Y) + \alpha \mathbb{1}\mathbb{1}^T - B\right)$ should have rank $n - 1$.

Using strong duality, (1) will suffice to ensure that (U', η') is an optimal solution of the primal SDP . To show that this is the unique primal optimal solution, we will use the complementary slackness conditions which state that

$$U \cdot \left(L(Y) + \alpha \mathbb{1}\mathbb{1}^T - B\right) = 0\,. \tag{1}$$

Since, $\left(L(Y) + \alpha \mathbb{1}\mathbb{1}^T - B\right)$ will have rank $n - 1$, this will imply that all primal optimal solutions must have rank at most 1, or in other words, there is a unique primal optimal solution (see Lemma 11).

While the approach of using complementary slackness conditions for proving the integrality of the SDP relaxation has been studied for similar problems before ([15, 2, 3, 19, 7]), there is no known generic way of implementing this approach to any given problem. Usually the challenging part in implementing this approach is in constructing an appropriate dual solution, and that, like in most of the works cited above, forms the core of our proof.

We give an outline of how we construct our dual solutions. We begin by setting the Y value for each edge added by the monotone adversary to 0, thus our proof can be viewed as saying that SDP 6 "ignores" all those edges. For the sake of simplicity, let us consider the case when the bipartite graph between T and T' is a c-regular graph. We set $B_{ii} := 4$ if $i \in T \cup T'$ and 0 if $i \notin T \cup T'$. Thus, if we can choose Y such that this choice of B is a feasible solution, then this will ensure that the cost of this dual solution, and the cost of the primal solution (U', η') are both equal to $4(\varepsilon_1 + \varepsilon_2)n$, thereby fullfilling our first requirement.

If U is a rank one matrix, and $\left(L(Y) + \alpha \mathbb{1}\mathbb{1}^T - B\right)$ is a rank $n - 1$ matrix, then (1) implies that $\mathbb{1}_S$ is an eigenvector of $\left(L(Y) + \alpha \mathbb{1}\mathbb{1}^T - B\right)$ with eigenvalue 0. This fact will be extremely useful in setting the Y values for the edges in the bipartite graph between T and T' (Lemma 12). Now, we only have to choose the Y values for the edges fully contained in S (resp. S'). We first prove the following lemma which will help us to choose the Y values.

▶ **Lemma 9** (Informal statement of Lemma 10). *There exists a constant c' such that it suffices to choose Y satisfying*

$$X^T\left(L(Y)\right)X \geqslant \frac{c'}{n}\left(\sum_{i\in S\setminus T}\sum_{t\in T}\left(X_i-X_t\right)^2+\sum_{i\in S'\setminus T'}\sum_{t\in T'}\left(X_i-X_t\right)^2\right) \qquad \forall X\in\mathbb{R}^n\,.$$

The proof of this lemma follows by carefully choosing the value of α, and by exploiting the fact that $\mathbb{1}_S$ is an eigenvector of $L(Y)+\alpha\mathbb{1}\mathbb{1}^T-B$ with eigenvalue 0. Proving the condition in Lemma 9 can be viewed as the problem of choosing capacities for the edges to support the multicommodity flow where each vertex $i\in S\setminus T$ wants to send c'/n amount of flow to each $t\in T$. This idea can work when S (resp. S') is a sufficiently dense graph, but does not work when S (resp. S') is sparse (for more details, we refer the reader to the full version). Our second idea is to use the edge expansion properties of the underlying spanning subgraph. For a d-regular edge expander $H=(V',E')$ having the second smallest normalized Laplacian eigenvalue λ, we get that $\sum_{\{ij\}\in E'}\left(X_i-X_j\right)^2\geqslant(\lambda d/n)\sum_{ij\in E'}\left(X_i-X_j\right)^2$. Since $L(Y)$ is a Laplacian matrix, we get that

$$\begin{aligned}X^TL(Y)X=&\sum_{i,j\in S\{i,j\}\in E}\left(Y_{ij}+Y_{ji}\right)\left(X_i-X_j\right)^2\\&+\sum_{i,j\in S'\{i,j\}\in E}\left(Y_{ij}+Y_{ji}\right)\left(X_i-X_j\right)^2\\&+\sum_{i\in T,j\in T'\{i,j\}\in E}\left(Y_{ij}+Y_{ji}\right)\left(X_i-X_j\right)^2\,.\end{aligned}$$

Now, since S and S' contain an almost regular edge expander as a spanning subgraph, we can adapt the expander argument to this setting and obtain some lower bound on this quantity. This strategy can work in some special cases, but fails in general. Our proof shows that the desired lower bound in Lemma 9 can be obtained using a careful combination of these two ideas, in addition to exploiting the various properties of the random graph between $S\setminus T$ and T (resp. $S'\setminus T'$ and T').

1.5.2 Theorem 3 and Theorem 4

We first solve SDP 6 and obtain a matrix U such that $U\succeq 0$. Therefore, U can be factorized into $U=W^TW$ for some matrix W. Let $u_1,\ldots,u_n$ denote the columns of this matrix W. We give an algorithm to "round" these vectors into a set satisfying the guarantees in the theorem. As in the previous case, we show that we can "ignore" all the edges added by the monotone adversary, and only focus on the edges added in step 2, 3, 4 in Definition 1.

A well known fact for edge expander graphs having roughly equal vertex degrees is that if the value of $\|u_i-u_j\|^2$ averaged over all edges $\{i,j\}$ in the graph is small, then the value of $\|u_i-u_j\|^2$ averaged over all pairs of vertices i,j in the graph is also small. In the proof of Theorem 4, we use the expansion properties of the $\Omega(n)$ sized subset of S coupled with this fact to show that an $\Omega(n)$ sized subset of the vectors $\{u_i:i\in V\}$ must lie in a ball of small diameter; this step is similar to the corresponding step of [30]. We use this to construct an embedding of the graph onto a line, and recover a cut from this embedding using an algorithm of [29]; this step can be viewed as adapting the corresponding step of [30] to vertex expansion.

In the case when $\lambda_1=0$, we show that the lopsided random bipartite graph between $S\setminus T$ and T is an edge expander w.h.p. However, this graph is not close to being regular;

the degrees of the vertices in T would be much higher than the degrees of the vertices in $S \setminus T$. Therefore, we can not directly use the strategy employed in the previous case. But we show that we can use the fact that the measure of $S \setminus T$ under the stationary distribution of the random bipartite graph between $S \setminus T$ and T is $\Omega(1)$, and that the vertices in $S \setminus T$ have roughly equal vertex degrees, to show that $\|u_i - u_j\|^2$ averaged over all pairs of vertices $i, j \in S \setminus T$ is small. From here, we proceed as in the previous case.

1.6 Notation

We denote graphs by $G = (V, E)$, where the vertex set V is identified with $[n] := \{1, 2, \dots n\}$. For any $S \subseteq V$, we denote the induced subgraph on S by $G[S]$. Given $i \in V$ and $T \subseteq V$, define $N_T(i) \stackrel{\text{def}}{=} \{j \in T \ : \ \{i, j\} \in E\}$, and $N(i) = N_V(i)$. We denote $\Delta_T(i) := |N_T(i)|$, and $\Delta(i) = |N(i)|$. For a subgraph $\mathcal{F}$ of G, the degree of i within $\mathcal{F}$ is correspondingly $\Delta_{\mathcal{F}}(i)$.

Given a graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}_{\geqslant 0}$ on its edges, we define the *weighted* degree of a vertex $i \in V$ as $d(i) := \sum_{j \in N(i)} w_{\{i,j\}}$. In our theorems, following SDP 7, we will be assigning directed weights (or capacities) Y_{ij} to edges $\{i, j\} \in E$, and use $L(Y)$ to denote the Laplacian of G with weights $Y_{ij} + Y_{ji}$ on the edges.

Given the normalized Laplacian $\mathcal{L} = I - D^{-1/2} A D^{-1/2}$, the *spectral gap* of G denoted by λ, is the second-smallest eigenvalue of $\mathcal{L}$.

Typically, for a vector $X \in \mathbb{R}^n$, its i-th component is denoted by X_i, or in rare cases for clarity, by $X(i)$. As in the introduction, we use $\mathbb{1}_S$ for any $S \subseteq V$ to denote the vector in $\mathbb{R}^{|V|}$ having entries $\mathbb{1}_S(i) = 1$, if $i \in S$, and -1 otherwise.

2 Exact Recovery for VBM

2.1 A sufficient condition

In order to prove Theorem 8, we start with the following lemma, which outlines a sufficient condition for integrality of the primal optimal SDP solution. For the proof of this lemma, and all subsequent lemmas in this section, we refer the reader to the full version of the paper.

▶ **Lemma 10.** *For a* VBM$(n, \varepsilon_1, \varepsilon_2, p_1, p_2, c, r, \lambda_1, \lambda_2)$ *instance, if we can find a* $Y \in \mathbb{R}^{n \times n}$ *that satisfies the linear constraints on it (*$Y_{ij} \geqslant 0$ *and* $\sum_j Y_{ij} = 1$*), and in addition has:*
(a) $\forall X \in \mathbb{R}^n$,

$$X^T \left(L(Y)\right) X \geqslant \frac{c'}{n} \left(\sum_{i \in S \setminus T} \sum_{t \in T} (X_i - X_t)^2 + \sum_{i \in S' \setminus T'} \sum_{t \in T'} (X_i - X_t)^2 \right), \tag{2}$$

(b) *For every* $i \in T, j \in T'$, *we have* $Y_{ij} = 1/\Delta_{\mathcal{F}}(i)$ *and* $Y_{ji} = 1/\Delta_{\mathcal{F}(j)}$,
(c) *For every* $i \in T, j \in S \setminus T$ *and* $i \in T', j \in S \setminus T'$, *we have* $Y_{ij} = 0$
where $c' = 8c/(1 - \max\{\varepsilon_1, \varepsilon_2\})$, *then SDP 6 has* (U', η') *as defined in Theorem 8 as its unique optimal solution.*

▶ Remark. Along with the SDP 7 linear constraints on Y, the above conditions ensure that we can extend Y to a feasible dual solution (Y, B, α), that satisfies the positive-semidefiniteness constraint *and* is optimal.

We begin by noting a simple consequence of complementary slackness conditions.

▶ **Lemma 11.** *Let* $M := L(Y) + \alpha \mathbb{1}\mathbb{1}^T - B$ *be constructed using an optimal dual solution* (Y, B, α). *The primal optimal solution is integral and unique if* $\mathbb{1}_S$ *is a unique eigenvector of* M *with eigenvalue* 0.

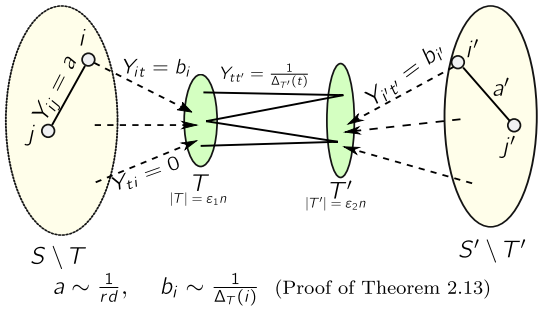


Figure 2 Outline of Y_{ij}'s used in the constructed dual solution.

It is thus sufficient to prove that the conditions in Lemma 10 imply that we can use the given Y to come up with a B and α, such that (Y, B, α) is feasible, *and* $\mathbb{1}_S$ is a unique eigenvector of M with eigenvalue 0. We first find a B (depending on Y) that yields a dual objective value of exactly $4(\varepsilon_1 + \varepsilon_2)n$ and ensures that $\mathbb{1}_S$ is an eigenvector with eigenvalue 0. We use $d_T(i) = \sum_{j \in N(i) \cap T}(Y_{ij} + Y_{ji})$ for the weighted degree of i into $T \subseteq V$.

▶ **Lemma 12.** *Fix some partial candidate dual solution (Y, α). Consider the setting of the diagonal matrix B given by:*

$$B_{ii} = \begin{cases} 2 \cdot d_{T'}(i), & \text{for } i \in T \\ 2 \cdot d_T(i), & \text{for } i \in T' \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Then $\mathbb{1}_S$ is an eigenvector of M with eigenvalue 0. Furthermore, if (Y, α) is feasible for this B and satisfies condition (c) of Lemma 10, then the dual variable assignment (Y, B, α) is optimal, with objective value $4(\varepsilon_1 + \varepsilon_2)n$.

Thus, at this point, we know how to show that a feasible (Y, α) satisfying the conditions in Lemma 10 can be extended to get the dual variable B that ensures that the primal is integral. It now remains to show that given an assignment to Y that satisfies conditions in Lemma 10, and a B constructed thereof using Lemma 12, we can find an appropriate α so that (Y, B, α) is actually feasible.

The setting of B helps us exploit the following fact in showing that (Y, B, α) is feasible for some α.

▶ **Fact 13.** *If $M \in \mathbb{R}^{n \times n}$ is a symmetric matrix with eigenvector v having eigenvalue 0, then $M \succeq 0$ and $\text{rank}(M) = n - 1 \iff \exists l > 0 \ : \ M + l \cdot vv^T \succ 0$.*

Thus, instead of showing that $M := (L + \alpha \mathbb{1}\mathbb{1}^T - B) \succeq 0$, we use the above fact with $v = \mathbb{1}_S$. By our setting for B, $\mathbb{1}_S$ is an eigenvector of M with eigenvalue 0. We show that there is an α that gives us that $M' := L + \alpha\left(\mathbb{1}\mathbb{1}^T + \mathbb{1}_S\mathbb{1}_S^T\right) - B \succ 0$ (by using condition (a) and (b) of Lemma 10).

This ensures M satisfies Lemma 11, completing the proof of Lemma 10. We leave the details to the full version of the paper.

2.2 Satisfying the sufficient condition

We are now left with the task of assigning appropriate weights Y_{ij} to the edges, so that Lemma 10's conditions are satisfied. Effectively, we only have to find weights within $G[S]$ and

$G[S']$, as Lemma 10 already fixes the rest. First, we recall that for any $X \in \mathbb{R}^n$, $X^T L(Y) X = \sum_{\{i,j\} \in E} (Y_{ij} + Y_{ji})(X_i - X_j)^2$. We drop the argument Y henceforth as it will be clear from context. The following observation is useful to keep in mind:

▶ **Observation 14.** *Given a (Y, B, α) dual solution, that satisfies all but the constraints $\sum_{j \in N(i)} Y_{ij} = 1$, having instead that $\forall i \in V \ : \ \sum_{j \in N(i)} Y_{ij} \leqslant 1$, we can produce a feasible solution (Y', B, α) of the same objective value.*

We will henceforth find values for the dual variable Y satisfying just the weaker constraint $\sum_{j \in N(i)} Y_{ij} \leqslant 1$. We first describe the following instructive case, in order to aid intuition.

▶ **Lemma 15.** *When $G[S]$ and $G[S']$ are complete graphs, there is a constant $\beta < 1$ and weights Y, such that when $\varepsilon \stackrel{\text{def}}{=} \max\{\varepsilon_1, \varepsilon_2\} \leqslant \beta$, the sufficient condition in Lemma 10 is satisfied, and hence the primal SDP is integral.*

Proof. For every pair i, j such that $i \in S \setminus T$ and $t \in T$, we set $Y_{it} = b := \frac{1}{\varepsilon n}$. The Y_{ij}'s within S' are set similarly. Let $Y_{ij} = 0$ for all other edges within $G[S]$, and $G[S']$. The constraint $\sum_{j \in N(i)} Y_{ij} \leqslant 1$ is satisfied as $b \times \Delta_T(i) \leqslant 1 \implies b\varepsilon n \leqslant 1$.

In order to prove integrality, we verify that (2) holds for the chosen value of b. Expanding out the term $X^T L X$ gives:

$$X^T L X \ \geqslant \ b \sum_{\substack{i \in S \setminus T \\ t \in T}} (X_i - X_t)^2 \ + \ b \sum_{\substack{i \in S' \setminus T' \\ t \in T'}} (X_i - X_t)^2 .$$

From the condition in Lemma 10, we get that the primal SDP is integral as long as $bn \geqslant 8c/(1-\varepsilon)$, which is true as long as $\frac{1-\varepsilon}{\varepsilon} > 8c$. This is true for ε being less than a small enough constant. ◀

Let us now consider the general case. We will focus on just S henceforth, as similar arguments will work for S' too, and the feasible solution can be constructed independently for either part. Observe that in contrast to the complete graph above, certain terms are missing in the expansion of $X^T L_{|S} X$: these terms are of the form

$$(X_i - X_t)^2 \qquad \forall i \in S \setminus T, t \in T : i \notin N(t) .$$

One way to recover these terms is to make use of the following observation:

▶ **Fact 16.** *For any $x_1, x_2, \ldots x_{l+1} \in \mathbb{R}$, we have $\sum_{i=1}^{l} (x_i - x_{i+1})^2 \geqslant \frac{1}{l}(x_1 - x_{l+1})^2$*

Flow Routing:

Fact 16 gives us a way to generate terms of the form $(X_i - X_t)^2$ using the edges present in the graph $G[S]$. In particular, we can generate a missing term of the form $(X_i - X_t)^2$, as a sum along a path $P = (i_1 = i, i_2, i_3, \ldots, i_l = t)$ in G of the terms $(X_{i_j} - X_{i_{j+1}})^2$, for every $j \in [l-1]$. Each of these terms occurs in the expansion of $X^T L X$. If we use an amount a of the weight of each edge on $\mathcal{P}$ in doing so, the final term has a coefficient of $\frac{a}{l}$, and this can be seen as i attempting to sending a 'flow' of magnitude a to t via $\mathcal{P}$.

Generating all the missing terms can now be formulated as a flow-routing problem using paths of length at most l (for some fixed l). The flows going from i to t generate the term $(X_i - X_t)^2$. Lemma 10 can therefore be restated as the problem of routing at least $c'l/n$ units of flow from every $i \in S \setminus T$ to $t \in T$. The constraint on the (directed) flow edges out of i is determined by the values Y_{ij}. The capacity of the edge $\{i, j\}$ in the direction $i \to j$ is

Y_{ij}, and the outdegree constraint states that every vertex can push out at most one unit of flow in total. Furthermore, a flow of 'a' units travelling along a path of distance l to reach t finally contributes only a/l, due to Fact 16. We state this idea formally below.

▶ **Lemma 17** (Flow routing problem). *Suppose we are given $G[S]$ and $G[S']$ with a feasible assignment Y for the edges. Consider a directed version of $G[S]$, where every edge $\{i,j\} \in E$ is replaced by the directed edges (i,j) and (j,i) with capacities Y_{ij} and Y_{ji} respectively. If for some $l \in \mathbb{N}$, and for every $i \in S \setminus T$ and $t \in T$, we can route a flow of $c'l/n$ from $i \to t$ using paths of length at most l in $G[S]$ (and similarly for $G[S']$), while obeying the (directed) capacity constraints on the edges, then we have:*

$$X^T L X \geqslant \frac{c'}{n}\left(\sum_{i\in S\setminus T}\sum_{t\in T}(X_i - X_t)^2 + \sum_{i\in S'\setminus T'}\sum_{t\in T'}(X_i - X_t)^2\right) \qquad \forall X \in \mathbb{R}^n$$

However, simply routing flows satisfying the above in an arbitrary $G[S]$ (or $G[S']$) turns out to be impossible. We carefully exploit the spectral expansion of $G[S \setminus T]$ (and $G[S' \setminus T']$), along with the randomness on the graph in $S \setminus T \times T$ while routing our flows to find our assignment for Y. Figure 2 summarizes briefly the dual values for Y that we finally use. We leave the details for the full version of the paper.

References

1. Emmanuel Abbe. Community detection and stochastic block models: recent developments. *arXiv preprint arXiv:1703.10146*, 2017.
2. Emmanuel Abbe, Afonso S Bandeira, Annina Bracher, and Amit Singer. Decoding binary node labels from censored edge measurements: Phase transition and efficient recovery. *IEEE Transactions on Network Science and Engineering*, 1(1):10–22, 2014.
3. Emmanuel Abbe, Afonso S Bandeira, and Georgina Hall. Exact recovery in the stochastic block model. *IEEE Transactions on Information Theory*, 62(1):471–487, 2016.
4. Emmanuel Abbe and Colin Sandon. Community detection in general stochastic block models: Fundamental limits and efficient algorithms for recovery. In *IEEE 56th Annual Symp. on Foundations of Computer Science (FOCS), 2015*, pages 670–688. IEEE, 2015.
5. Emmanuel Abbe and Colin Sandon. Recovering communities in the general stochastic block model without knowing the parameters. In *Advances in neural information processing systems*, pages 676–684, 2015.
6. Emmanuel Abbe and Colin Sandon. Detection in the stochastic block model with multiple clusters: proof of the achievability conjectures, acyclic bp, and the information-computation gap, 2017. arXiv: 1512.09080 To Appear in Communications on Pure and Applied Mathematics (2017).
7. Naman Agarwal, Afonso S Bandeira, Konstantinos Koiliaris, and Alexandra Kolla. Multisection in the stochastic block model using semidefinite programming. In *Compressed Sensing and its Applications*, pages 125–162. Springer, 2017.
8. Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986. URL: `http://dx.doi.org/10.1007/BF02579166`, `doi:10.1007/BF02579166`.
9. Noga Alon and Vitali D Milman. λ_1, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, Series B*, 38(1):73–88, 1985.
10. Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM*, 56(2), 2009. (Preliminary version in *36th STOC*, 2004). `doi:10.1145/1502793.1502794`.

11 Roberto Battiti and Marco Protasi. Approximate algorithms and heuristics for max-sat. In *Handbook of Combinatorial Optimization: Volume1–3*, pages 77–148, Boston, MA, 1999. Springer US. URL: http://dx.doi.org/10.1007/978-1-4613-0303-9_2, doi:10.1007/978-1-4613-0303-9_2.

12 Sergey Bobkov, Christian Houdré, and Prasad Tetali. λ_∞, Vertex Isoperimetry and Concentration. *Combinatorica*, 20(2):153–172, 2000.

13 Ravi B. Boppana. Eigenvalues and graph bisection: An average-case analysis. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, SFCS '87, pages 280–285, Washington, DC, USA, 1987. IEEE Computer Society. URL: http://dx.doi.org/10.1109/SFCS.1987.22, doi:10.1109/SFCS.1987.22.

14 T.-H. Hubert Chan, Anand Louis, Zhihao Gavin Tang, and Chenzi Zhang. Spectral Properties of Hypergraph Laplacian and Approximation Algorithms. *J. ACM*, 65(3):15:1–15:48, 2018. URL: http://doi.acm.org/10.1145/3178123, doi:10.1145/3178123.

15 Amin Coja-Oghlan. Colouring semirandom graphs. *Combinatorics, Probability and Computing*, 16(4):515–552, 2007. doi:10.1017/S0963548306007917.

16 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008. URL: https://doi.org/10.1137/05064299X, arXiv:https://doi.org/10.1137/05064299X, doi:10.1137/05064299X.

17 Uriel Feige and Joe Kilian. Heuristics for semirandom graph problems. *Journal of Computer and System Sciences*, 63(4):639–671, 2001.

18 Olivier Guédon and Roman Vershynin. Community detection in sparse networks via grothendieck's inequality. *Probability Theory and Related Fields*, 165(3-4):1025–1049, 2016.

19 Bruce Hajek, Yihong Wu, and Jiaming Xu. Achieving exact cluster recovery threshold via semidefinite programming: Extensions. *IEEE Transactions on Information Theory*, 62(10):5918–5937, 2016.

20 Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

21 Mark Jerrum and Gregory B Sorkin. The metropolis algorithm for graph bisection. *Discrete Applied Mathematics*, 82(1):155–175, 1998.

22 George Karypis and Vipin Kumar. Analysis of multilevel graph partitioning. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, Supercomputing '95, New York, NY, USA, 1995. ACM. URL: http://doi.acm.org/10.1145/224170.224229, doi:10.1145/224170.224229.

23 George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, December 1998. URL: http://dx.doi.org/10.1137/S1064827595287997, doi:10.1137/S1064827595287997.

24 Chiheon Kim, Afonso S. Bandeira, and Michel X. Goemans. Community detection in hypergraphs, spiked tensor models, and sum-of-squares. *arXiv preprint: arXiv:1705.02973 [cs.DS]*, 2017.

25 Victor Klee and George J Minty. How good is the simplex algorithm. In *Shisha, Oved. Inequalities III (Proc. of 3rd Symp. on Inequalities, UCLA)*, pages 159–175. Academic Press, California, 1972.

26 Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, November 1999. URL: http://doi.acm.org/10.1145/331524.331526, doi:10.1145/331524.331526.

27 Anand Louis and Yury Makarychev. Approximation Algorithms for Hypergraph Small Set Expansion and Small Set Vertex Expansion. In *Approximation, Randomization, and*

Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014), volume 28 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 339–355, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.339.

28 Anand Louis and Prasad Raghavendra, 2014. Personal Communication.

29 Anand Louis, Prasad Raghavendra, and Santosh Vempala. The complexity of approximating vertex expansion. In *Proc. of the 54th Annual Symp. on Foundations of Computer Science*, FOCS '13, pages 360–369, Washington, DC, USA, 2013. IEEE Computer Society. URL: http://dx.doi.org/10.1109/FOCS.2013.46, doi:10.1109/FOCS.2013.46.

30 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Approximation algorithms for semi-random partitioning problems. In *Proc. of the 44th Annual ACM Symp. on Theory of Computing*, STOC '12, pages 367–384. ACM, 2012. URL: http://doi.acm.org/10.1145/2213977.2214013, doi:10.1145/2213977.2214013.

31 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Constant factor approximation for balanced cut in the pie model. In *Proc. of the 46th Annual ACM Symp. on Theory of Computing*, STOC '14, pages 41–49, New York, NY, USA, 2014. ACM. URL: http://doi.acm.org/10.1145/2591796.2591841, doi:10.1145/2591796.2591841.

32 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Learning communities in the presence of errors. In *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pages 1258–1291, Columbia University, New York, New York, USA, 23–26 Jun 2016. PMLR. URL: http://proceedings.mlr.press/v49/makarychev16.html.

33 Laurent Massoulié. Community detection thresholds and the weak ramanujan property. In *Proc. of the 46th Annual ACM Symp. on Theory of Computing*, STOC '14, pages 694–703, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591857.

34 Frank D. McSherry. Spectral partitioning of random graphs. In *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 529–537, Washington, DC, USA, 2001. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=874063.875554.

35 Ankur Moitra, William Perry, and Alexander S Wein. How robust are reconstruction thresholds for community detection? In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 828–841. ACM, 2016.

36 Elchanan Mossel, Joe Neeman, and Allan Sly. Belief propagation, robust reconstruction and optimal recovery of block models. In *Conference on Learning Theory*, pages 356–370, 2014.

37 Elchanan Mossel, Joe Neeman, and Allan Sly. Consistency thresholds for the planted bisection model. In *Proc. of the 47th Annual ACM Symp. on Theory of Computing*, STOC '15, pages 69–75, New York, NY, USA, 2015. ACM. URL: http://doi.acm.org/10.1145/2746539.2746603, doi:10.1145/2746539.2746603.

38 Elchanan Mossel, Joe Neeman, and Allan Sly. A proof of the block model threshold conjecture. *Combinatorica*, 2017. URL: https://doi.org/10.1007/s00493-016-3238-8, doi:10.1007/s00493-016-3238-8.

39 Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 755–764, New York, NY, USA, 2010. ACM. URL: http://doi.acm.org/10.1145/1806689.1806792, doi:10.1145/1806689.1806792.

Load Thresholds for Cuckoo Hashing with Overlapping Blocks

Stefan Walzer
Technische Universität Ilmenau, Germany
stefan.walzer@tu-ilmenau.de
https://orcid.org/0000-0002-6477-0106

Abstract

2012 ACM Subject Classification Theory of computation → Data structures design and analysis, Mathematics of computing → Probabilistic algorithms, Theory of computation → Bloom filters and hashing

Keywords and phrases Cuckoo Hashing, Unaligned Blocks, Hypergraph Orientability, Load Thresholds, Randomised Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.102

Related Version Due to size constraints, many technical details are omitted. These details will be included in a full version and can already be found in the preprint [22] (`https://arxiv.org/abs/1707.06855`) of this article.

Acknowledgements I am indebted to my advisor Martin Dietzfelbinger for drawing my attention to this problem as well as for providing a constant stream of useful literature recommendations. When discussing a preliminary version of this work at the Dagstuhl Seminar 17181 on Theory and Applications of Hashing, Michael Mitzenmacher and Konstantinos Panagiotou provided useful comments.

1 Introduction

In **standard cuckoo hashing** [20], a set $X = \{x_1, \dots, x_{cn}\}$ of objects (possibly with associated data) from a universe $\mathcal{U}$ is to be stored in a hash table indexed by $V = \{0, \dots, n-1\}$ of size n such that each object x_i resides in one of two associated memory locations $h_1(x_i), h_2(x_i)$, given by hash functions $h_1, h_2 : \mathcal{U} \to V$. In most theoretic works, these functions are modelled as fully random functions, selected uniformly and independently from $V^{\mathcal{U}}$.

The load parameter $c \in [0, 1]$ indicates the desired space efficiency, i.e. the ratio between objects and allocated table positions. Whether or not a valid placement of the objects in the table exists is well predicted by whether c is above or below the *threshold* $c^* = \frac{1}{2}$: If $c \leq c^* - \varepsilon$ for arbitrary $\varepsilon > 0$, then a placement exists with high probability (whp), i.e. with probability approaching 1 as n tends to infinity, and if $c \geq c^* + \varepsilon$ for $\varepsilon > 0$, then no placement exists whp.

If a placement is found, we obtain a dictionary data structure representing $X \subseteq \mathcal{U}$. To check whether an object $x \in U$ resides in the dictionary (and possibly retrieve associated data), only the memory locations $h_1(x)$ and $h_2(x)$ need to be computed and searched for x. Combined with results facilitating swift creation, insertion and deletion, standard cuckoo hashing has decent performance when compared to other hashing schemes at load factors around $\frac{1}{3}$ [20].

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 102; pp. 102:1–102:10


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Several generalisations have been studied that allow trading rigidity of the data structure – and therefore performance of lookup operations – for load thresholds closer to 1.

- In ***k*-ary cuckoo hashing**, due to Fotakis et al. [8], a general number $k \geq 2$ of hash functions is used.
- Dietzfelbinger and Weidling [6] propose partitioning the table into $\frac{n}{\ell}$ contiguous **blocks of size ℓ** and assign two random blocks to each object via the two hash functions, allowing an object to reside anywhere within those blocks.
- By **windows of size ℓ** we mean the related idea – called "cuckoo-lp" in [6] – where x may reside anywhere in the intervals $[h_1(x), h_1(x) + \ell)$ and $[h_2(x), h_2(x) + \ell)$ (all indices understood modulo n). Compared to the block variant, the values $h_1(x), h_2(x) \in V$ need not be multiples of ℓ, so the possible intervals do not form a partition of V.

The overall performance of a cuckoo hashing scheme is a story of multidimensional trade-offs and hardware dependencies, but based on experiments in [6, 17] roughly speaking, the following empirical claims can be made:

- k-ary cuckoo hashing for $k > 2$ is slower than the other two approaches. This is because lookup operations trigger up to k evaluations of hash functions and k random memory accesses, each likely to result in a cache fault. In the other cases, only the number of key comparisons rises, which are comparatively cheap.
- Windows of size ℓ offer a better tradeoff between worst-case lookup times and space efficiency than blocks of size ℓ.

Although our results are oblivious of hardware effects, they support the second empirical observation from a mathematical perspective.

1.1 Previous Work on Thresholds

Precise thresholds are known for k-ary cuckoo hashing [4, 12, 10], cuckoo hashing with blocks of size ℓ [7, 3], and the combination of both, i.e. k-ary cuckoo hashing with blocks of size ℓ with $k \geq 3, \ell \geq 2$ [9]. The techniques in the cited papers are remarkably heterogeneous and often specific to the cases at hand. Lelarge [18] managed to unify the above results using techniques from statistical physics that, perhaps surprisingly, feel like they grasp more directly at the core phenomena. Generalising further, Leconte, Lelarge, and Massoulié [15] solved the case where each object must occupy $j \in \mathbb{N}$ incident table positions, $r \in \mathbb{N}$ of which may lie in the same block (see also [13]).

Lehman and Panigrahy [17] showed that, asymptotically, the load threshold is $1 - (2/e + o_\ell(1))^\ell$ for cuckoo hashing with blocks of size ℓ and $1 - (1/e + o_\ell(1))^{1.59\ell}$ in the case of windows, with no implication for small constant ℓ. Beyer [2] showed in his master's thesis that for $\ell = 2$ the threshold is at least 0.829 and at most 0.981. To our knowledge, this is an exhaustive list of published work concerning windows.

In a spirit similar to cuckoo hashing with windows, Porat and Shalem [21] analyse a scheme where memory is partitioned into pages and a bucket of size k is a choice of k memory positions from the same page (not necessarily contiguous). The authors provide rigorous lower bounds on the corresponding thresholds as well as empirical results.

1.2 Our Contribution

We provide precise thresholds for k-ary cuckoo hashing with windows of size ℓ for all $k, \ell \geq 2$. In particular this solves the case of $k = 2$ left open in [6, 17]. Note the pronounced improvements in space efficiency when using windows over blocks, for instance in the case of $k = \ell = 2$, where the threshold is at roughly 96.5% instead of roughly 89.7%.

Formally, for any $k, \ell \geq 2$, we identify real analytic functions $f_{k,\ell}$, $g_{k,\ell}$, such that for $\gamma_{k,\ell} = \inf_{\lambda>0}\{f_{k,\ell}(\lambda) \mid g_{k,\ell}(\lambda) < 0\}$ we have

▶ **Main Theorem.** *The threshold for k-ary cuckoo hashing with windows of size ℓ is $\gamma_{k,\ell}$, in particular for any $\varepsilon > 0$,*

(i) *if $c > \gamma_{k,\ell} + \varepsilon$, then no valid placement of objects exists whp and*

(ii) *if $c < \gamma_{k,\ell} - \varepsilon$, then a valid placement of objects exists whp.*

While $f_{k,\ell}$ and $g_{k,\ell}$ are very unwieldy, with ever more terms as ℓ increases, numerical aproximations of $\gamma_{k,\ell}$ can be attained with mathematics software. We provide some values in Table 1.

1.3 Methods

The obvious methods to model cuckoo hashing with windows either give probabilistic structures with awkward dependencies or the question to answer for the structure follows awkward rules. Our first non-trivial step is to transform a preliminary representation into a hypergraph with n vertices, cn uniformly random hyperedges of size k, an added deterministic cycle, and a question strictly about the orientability of this hypergraph.

In the new form, the problem is approachable by a combination of belief propagation methods and the objective method [1], adapted to the world of hypergraph orientability by Lelarge [18] in his insightful paper. The results were further strengthened by a Theorem in [15], which we apply at a critical point in our argument.

As the method is fundamentally about approximate sizes of incomplete orientations, it leaves open the possibility of $o(n)$ unplaced objects; a gap that can be closed in an afterthought with standard methods.

Table 1 Some thresholds $c_{k,\ell}$ as obtained by [20, 3, 7, 4, 12, 11, 9] and values of $\gamma_{k,\ell}$ as obtained from our main theorem.
In both tables, the line for $\ell = 1$ corresponds to plain k-ary cuckoo hashing, reproduced here for comparison.

Thresholds $c_{k,\ell}$ for k-ary cuckoo hashing with blocks of size ℓ:

$\ell \backslash k$	2	3	4	5	6	7
1	0.5	0.9179352767	0.9767701649	0.9924383913	0.9973795528	0.9990637588
2	0.8970118682	0.9882014140	0.9982414840	0.9997243601	0.9999568737	0.9999933439
3	0.9591542686	0.9972857393	0.9997951434	0.9999851453	0.9999989795	0.9999999329
4	0.9803697743	0.9992531564	0.9999720661	0.9999990737	0.9999999721	0.9999999992

Thresholds $\gamma_{k,\ell}$ for k-ary cuckoo hashing with windows of size ℓ:

$\ell \backslash k$	2	3	4	5	6	7
1	0.5	0.9179352767	0.9767701649	0.9924383913	0.9973795528	0.9990637588
2	0.964994923	0.9968991072	0.9996335076	0.9999529036	0.9999937602	0.9999991631
3	0.994422754	0.9998255112	0.9999928198	0.9999996722	0.9999999843	0.9999999992
4	0.998951593	0.9999896830	0.9999998577	0.9999999977	≈ 1	≈ 1

1.4 Further Discussion

In the full version of this paper, we touch on three further issues that complement our results but are somewhat detached from our main theorem.

Numerical approximations of the thresholds. We explain how mathematics software can be used to get approximations for the values $\gamma_{k,\ell}$, which have been characterised only implicitely.

Speed of convergence. We provide experimental results with finite table sizes to demonstrate how quickly the threshold behaviour emerges.

Constructing orientations We examine the LSA algorithm by Khosla for insertion of elements, adapted to our hashing scheme. Experiments suggest an expected constant runtime per element as long as the load is bounded away from the threshold, i.e. $c < \gamma_{k,\ell} - \varepsilon$ for some $\varepsilon > 0$.

2 Definitions and Notation

A cuckoo hashing scheme specifies for each object $x \in X$ a set $A_x \subset V$ of table positions that x may be placed in. For our purposes, we may identitfy x with A_x. In this sense, $H = (V, X)$ is a hypergraph, where table positions are vertices and objects are hyperedges. The task of placing objects into admissible table positions corresponds to finding an *orientation* of H, which assigns each edge $x \in X$ to an incident vertex $v \in x$ such that no vertex has more than one edge assigned to it. If such an orientation exists, H is *orientable*.

We now restate the hashing schemes from the introduction in this hypergraph framework, switching to letters e (and E) to refer to (sets of) edges. We depart in notation, but not in substance, from definitions given previously, e.g. [8, 5, 17]. Illustrations are available in Figure 1.

Concerning ***k*-ary cuckoo hashing** the hypergraph is given as:

$$H_n = H^k_{n,cn} := (\mathbb{Z}_n, E = \{e_1, e_2, \dots, e_{cn}\}), \quad \text{for } e_i \leftarrow \tbinom{\mathbb{Z}_n}{k}, \tag{1}$$

where $\mathbb{Z}_n = \{0, 1, \dots, n-1\}$ and for a set S and $k \in \mathbb{N}$ we write $e \leftarrow \binom{S}{k}$ to indicate that $e = \{s_1, s_2, \dots, s_k\}$ is obtained by picking $s_1, \dots, s_k$ independently and uniformly at random from S.

There is a subtle difference to picking e uniformly at random from $\binom{S}{k}$, the set of all k-subsets of S, as the elements $s_1, \dots, s_k$ need not be distinct. We therefore understand e as a multiset. Also, we may have $e_i = e_j$ for $i \neq j$, so E is a multiset as well.[1]

Assuming the table size n is a multiple of ℓ, ***k*-ary cuckoo hashing with blocks of size ℓ** is modelled by the hypergraph

$$B_n = B^{k,\ell}_{n,cn} := (\mathbb{Z}_n, \{e'_1, e'_2, \dots, e'_{cn}\}), \quad \text{where } e'_i = \bigcup_{j \in e_i} [j\ell, (j+1)\ell) \text{ and } e_i \leftarrow \tbinom{\mathbb{Z}_{n/\ell}}{k}, \tag{2}$$

that is, each hyperedge is the union of k blocks chosen uniformly at random from the set of all blocks, which are the n/ℓ intervals of size ℓ in $\mathbb{Z}_n$ that start at a multiple of ℓ. Note that for $\ell = 1$ we recover H_n.

Similarly, ***k*-ary cuckoo hashing with windows of size ℓ** is modelled by

$$W_n = W^{k,\ell}_{n,cn} := (\mathbb{Z}_n, \{e'_1, e'_2, \dots, e'_{cn}\}), \quad \text{where } e'_i = \bigcup_{j \in e_i} [j, j+\ell) \text{ and } e_i \leftarrow \tbinom{\mathbb{Z}_n}{k}, \tag{3}$$

that is, each hyperedge is the union of k windows chosen uniformly at random from the set of all windows, which are the n intervals of size ℓ in $\mathbb{Z}_n$, this time without alignment

[1] While our choice for the probability space is adequate for cuckoo hashing and convenient in the proof, such details are inconsequential. Choosing H_n uniformly from the set of all hypergraphs with *cn distinct* edges all of which contain k *distinct* vertices would be equivalent for our purposes.

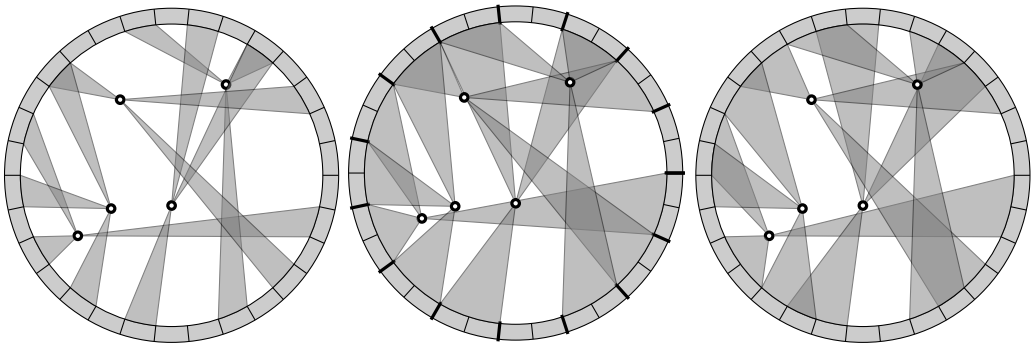


Figure 1 Drawing of possible outcomes for the hypergraphs H_n, B_n and W_n (modelling k-ary cuckoo hashing plain / with blocks / with windows) for $n = 30$, $c = \frac{1}{6}$, $k = 3$ and $\ell = 2$ (ℓ only for B and W). Each edge is drawn as a point and connected to all incident table cells, which are arranged in a circle. In the case of B, thick lines indicate the borders between blocks.

restriction. Note that intervals wrap around at the ends of the set $\{0, \dots, n-1\}$ with no awkward "border intervals". Again, for $\ell = 1$ we recover H_n.

3 Outline of the Proof

Step 1: A tidier problem. The elements of an edge e of B_n and W_n are not independent, as e is the union of k intervals of size ℓ. This poorly reflects the actual tidiness of the probabilistic object. We may obtain a model with independent elements in edges, by switching to a more general notion of what it means to orient a hypergraph.

Formally, given a weighted hypergraph $H = (V, E, \eta)$ with weight function $\eta : V \cup E \to \mathbb{N}$, an *orientation* μ of H assigns to each pair (e, v) of an edge and an incident vertex a number $\mu(e, v) \in \mathbb{N}_0$ such that

$$\forall e \in E\colon \sum_{v \in e} \mu(e, v) = \eta(e), \text{ and } \forall v \in V\colon \sum_{e \ni v} \mu(e, v) \leq \eta(v). \tag{4}$$

We will still say that an edge e *is oriented to* a vertex v (possibly several times) if $\mu(e, v) > 0$. One may be inclined to call $\eta(v)$ a *capacity* for $v \in V$ and $\eta(e)$ a *demand* for $e \in E$, but we use the same letter in both cases as the distinction is dropped later anyway.

Orientability of H, B and W from earlier is also captured in the generalised notion with implicit vertex weights of $\eta \equiv 1$.

A simplified representation of B_n is straightforward to obtain. We provide it mainly for illustration purposes, see Figure 2(a):

$$\hat{B}_n := \hat{B}^{k,\ell}_{n,cn} := (\mathbb{Z}_{n/\ell}, \{e_1, e_2, \dots, e_{cn}\}, \eta), \quad \text{where } e_i \leftarrow \tbinom{\mathbb{Z}_{n/\ell}}{k} \tag{5}$$
$$\text{and } \eta(v) = \ell \text{ for } v \in \mathbb{Z}_{n/\ell} \text{ and } \eta(e_i) = 1 \text{ for } 1 \leq i \leq cn.$$

In $\hat{B}_n$, each group of ℓ vertices of B_n representing one block is now contracted into a single vertex of weight ℓ and edges contain k independent vertices representing blocks instead of $k\ell$ dependent vertices. It is clear that B_n is orientable if and only if $\hat{B}_n$ is orientable.

In a similar spirit we identify a transformed version $\hat{W}_n$ for W_n, but this time the details are more complicated as the vertices have an intrinsic linear geometry, whereas B_n featured essentially an unordered collection of internally unordered blocks. The *ordinary edges* in $\hat{W}_n$

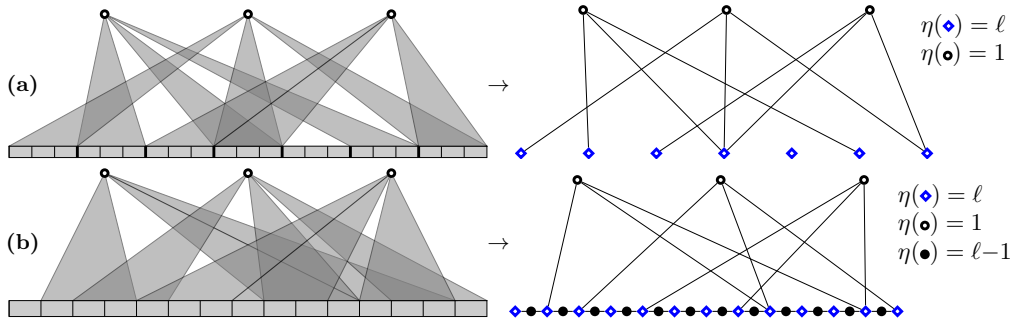


Figure 2 (a) In k-ary cuckoo hashing with blocks of size ℓ (here $k = \ell = 3$), we can contract each block into a single vertex of weight ℓ to obtain a simpler but equivalent representation of the orientation problem.
(b) In k-ary cuckoo hashing with windows of size ℓ, a similar idea can be made to work, but additional helper edges (drawn as $\bullet$) of weight $\ell - 1$ are needed (see Proposition 1).

also have size k instead of size $k\ell$, but we need to introduce additional *helper edges* that capture the linear geometry of $\mathbb{Z}_n$, see Figure 2(b). We define:

$$\begin{aligned}&\hat{W}_n := \hat{W}^{k,\ell}_{n,cn} := (\mathbb{Z}_n, C_n \cup \{e_1, \dots, e_{cn}\}, \eta) \\ &\text{with ordinary edges } e_i \leftarrow \tbinom{\mathbb{Z}_n}{k},\ \text{helper edges } C_n = \{c_i := (i, i+1) \mid i \in \mathbb{Z}_n\}, \\ &\text{and weights } \eta(w) = \ell,\ \eta(h) = \ell - 1,\ \eta(e) = 1 \quad \text{for} \quad w \in \mathbb{Z}_n, h \in C_n, e \in \{e_1, \dots, e_{cn}\}.\end{aligned} \tag{6}$$

Note that formally the graphs W_n and $\hat{W}_n$ are random variables on a common probability space. An outcome $\omega = (e_i)_{1 \le i \le cn}$ from this space determines both graphs.

The following proposition justifies the definition and is proved in the full version of this paper.

▶ **Proposition 1.** *$\hat{W}_n$ is orientable if and only if W_n is orientable.*[2]

An important merit of $\hat{W}_n$ that will be useful in Step 3 is that it is *locally tree-like*, meaning each vertex has a probability of $o(1)$ to be involved in a constant-length cycle. Here, by a cycle in a hypergraph we mean a sequence of distinct edges $e_1, e_2, \dots, e_j$ such that successive edges share a vertex and e_j and e_1 share a vertex.

Note the interesting special case $\hat{W}^{2,2}_{n,cn}$, which is a cycle of length n with cn random chords, unit edge weights and vertices of weight 2. Understanding the orientability thresholds for this graph seems interesting in its own right, not just as a means to understand $W^{2,2}_{n,cn}$.

Step 2: Incidence Graph and Allocations. The next step is by no means a difficult or creative one, we merely perform the necessary preparations needed to apply [15], introducing their concept of an allocation in the process.

This will effectively get rid of the asymmetry between the roles of vertices and edges in the problem of orienting $\hat{W}_n$, by switching perspective in two simple ways. The first is to consider the incidence graph G_n of $\hat{W}_n$ instead of $\hat{W}_n$ itself, i.e. the bipartite graph

$$G_n = G^{k,\ell}_{n,cn} = (\underbrace{C_n}_{A_C} \cup \underbrace{\{e_1, \dots, e_{cn}\}}_{A_R}, \underbrace{\mathbb{Z}_n}_{B}, \underbrace{\text{“}\ni\text{”}}_{E(G_n)}). \tag{7}$$

[2] Formally this should read: The events $\{W_n \text{ is orientable}\}$ and $\{\hat{W}_n \text{ is orientable}\}$ coincide.

We use $A = A_C \cup A_R$ to denote those vertices of G_n that were edges in $\hat{W}_n$ and B for those vertices of G_n that were vertices in $\hat{W}_n$. Vertices $a \in A$ and $b \in B$ are adjacent in G_n if $b \in a$ in $\hat{W}_n$. The weights η on vertices and edges in $\hat{W}_n$ are now vertex weights with $\eta(a_C) = \ell - 1$, $\eta(a_R) = 1$, $\eta(b) = \ell$ for $a_C \in A_C$, $a_R \in A_R$, $b \in B$. The notion of μ being an orientation translates to μ being a map $\mu : E(G_n) \to \mathbb{N}_0$ such that $\sum_{b \in N(a)} \mu(a,b) = \eta(a)$ for all $a \in A$ and $\sum_{a \in N(b)} \mu(a,b) \leq \eta(b)$ for all $b \in B$. Note that vertices from A need to be *saturated* ("$= \eta(a)$" for $a \in A$) while vertices from B need not be ("$\leq \eta(b)$" for $b \in B$). This leads to the second switch in perspective.

Dropping the saturation requirement for A, we say μ is an *allocation* if $\sum_{u \in N(v)} \mu(u,v) \leq \eta(v)$ for all $v \in A \cup B$.

Clearly, any orientation is an allocation, but not vice versa; for instance, the trivial map $\mu \equiv 0$ is an allocation. Let $|\mu|$ denote the size of an allocation, i.e. $|\mu| = \sum_{e \in E} \mu(e)$. By bipartiteness, no allocation can have a size larger than the total weight of A, i.e.

$$\text{for all allocations } \mu\colon |\mu| \leq \eta(A) = \sum_{a \in A} \eta(a) = |A_C| \cdot (\ell - 1) + |A_R| \cdot 1 = (\ell - 1 + c)n$$

and it is precisely the orientations of G_n that have size $\eta(A)$. We conclude:

▶ **Proposition 2.** *Let $M(G_n)$ denote the maximal size of an allocation of G_n. Then*

$$\tfrac{M(G_n)}{n} = \ell - 1 + c \quad \textit{if and only if} \quad G_n \textit{ is orientable} \quad \textit{if and only if} \quad \hat{W}_n \textit{ is orientable.}$$

Step 3: The Limit T of G_n. Reaping the benefits of step 1, we find G_n to have $O(1)$ cycles of length $O(1)$ whp. To capture the local appearance of G_n even more precisely, let the *r-ball* around a vertex v in a graph be the subgraph induced by the vertices of distance at most r from v. Then the r-ball around a random vertex of G_n is distributed, as n gets large, more and more like the r-ball around the root of a random infinite rooted tree $T = T_c^{k,\ell}$. It is distributed as follows, with nodes of types A_C, A_R or B.

- The root of T is of type A_C, A_R or B with probability $\frac{1}{2+c}$, $\frac{c}{2+c}$ and $\frac{1}{2+c}$, respectively.
- If the root is of type A_C, it has two children of type B. If it is of type A_R, it has k children of type B. If it is of type B, it has two children of type A_C and a random number X of children of type A_R, where $X \sim \mathrm{Po}(kc)$. Here $\mathrm{Po}(\lambda)$ denotes the Poisson distribution with parameter λ.
- A vertex of type A_C that is not the root has one child of type B. A vertex of type A_R that is not the root has $k-1$ children of type B.
- A vertex of type B that is not the root has a random number X of children of type A_R, where $X \sim \mathrm{Po}(kc)$. If its parent is of type A_C, then it has one child of type A_C, otherwise it has two children of type A_C.
- Vertices of type A_C, A_R and B have weight $\ell-1$, 1 and ℓ, respectively.

All random decisions should be understood to be independent. A type is also treated as a set containing all vertices of that type. In the full version of this paper we briefly recall the notion of *local weak convergence* and argue that the following holds:

▶ **Proposition 3.** *$(G_n)_{n \in \mathbb{N}} = (G_{n,cn}^{k,\ell})_{n \in \mathbb{N}}$ converges locally weakly to $T = T_c^{k,\ell}$.*

Step 4: The Method of [15]. We are now in a position to apply a powerful Theorem due to Leconte, Lelarge, and Massoulié [15] that characterises $\lim_{n \to \infty} \frac{M(G_n)}{n}$ in terms of solutions to belief propagation equations for T. Put abstractly: The limit of a function of G_n is a function of the limit of G_n. We elaborate on details and deal with the equations in the

full version of this paper. After condensing the results into a characterisation of $\gamma_{k,\ell} \in (0,1)$ in terms of "well-behaved" functions we obtain:

▶ **Proposition 4.**

$$\lim_{n\to\infty} \frac{M(G_{n,cn})}{n} \begin{cases} = \ell - 1 + c & \textit{almost surely} \quad \textit{if } c < \gamma_{k,\ell} \\ < \ell - 1 + c & \textit{almost surely} \quad \textit{if } c > \gamma_{k,\ell} \end{cases}$$

Step 5: Closing the Gap. It is important to note that we are not done, as

$$\lim_{n\to\infty} \frac{M(G_{n,cn})}{n} = \ell - 1 + c \text{ a.s.} \quad \text{does not imply}^3 \quad M(G_{n,cn}) = n \cdot (\ell - 1 + c) \text{ whp.} \quad (8)$$

We still have to exclude the possibility of a gap of size $o(n)$ on the right hand side, imagine for instance $M(G_{n,cn}) = (\ell - 1 + c)n - \sqrt{n}$ to appreciate the difference. In the setting of cuckoo hashing with double hashing (see [16]), it is actually the analogue of this pesky distinction that seems to be in the way of proving precise thresholds for perfect orientability, so we should treat this seriously.

Luckily the line of reasoning by Lelarge [18] can be adapted to our more general setting. The key is to prove that if not all objects can be placed into the hash table, then the configuration causing this problem has size $\Theta(n)$ (and those large overfull structures do not go unnoticed on the left side of (8)).

▶ **Lemma 1.** *There is a constant $\delta > 0$ such that whp no set of $0 < t < \delta n$ vertices in $\hat{W}_n$ (of weight ℓt) induces edges of total weight ℓt or more, provided $c \leq 1$.*

The proof of this Lemma (using first moment methods) and the final steps towards our main theorem are found in the full version of this paper.

4 Conclusion and Outlook

We established a method to determine load thresholds $\gamma_{k,\ell}$ for k-ary cuckoo hashing with (unaligned) windows of size ℓ. In particular, we resolved the cases with $k = 2$ left open in [6, 17], confirming corresponding experimental results by rigorous analysis.

The following four questions may be worthwhile starting points for further research.

Is there more in this method? It is conceivable that there is an insightful simplification of our calculations that yields a less unwieldy characterisation of $\gamma_{k,\ell}$. We also suspect that the threshold for the appearance of the $(\ell+1)$-*core* of $\hat{W}_n$ can be identified with some additional work (for cores see e.g. [19, 14]). This threshold is of interest because it is the point where the simple peeling algorithm to compute an orientation of $\hat{W}_n$ breaks down.

Can we prove efficient insertion? Given our experiments concerning the performance of Khosla's LSA algorithm for inserting elements in our hashing scheme (for details refer to the full version), it seems likely that its runtime is linear. But one could also consider approaches that do not insert elements one by one but build a hash table of load $c = \gamma_{k,\ell} - \varepsilon$ given all elements at once. Something in the spirit of the selfless algorithm [3] or excess degree reduction [4] may offer linear runtime with no performance degradation as ε gets smaller, at least for $k = 2$.

How good is it in practice? This paper does not address the competitiveness of our hashing scheme in realistic practical settings. The fact that windows give higher thresholds than (aligned) blocks for the *same* parameter ℓ may just mean that the "best" ℓ for a particular use case is lower, not precluding the possibility that the associated performance benefit is outweighed by other effects. [6] provide a few experiments in their appendix suggesting slight advantages for windows in the case of unsuccessful searches and slight disadvantages for successful searches and insert operations, in one very particular setup with $k = 2$. Further research could take into account precise knowledge of cache effects on modern machines, possibly using a mixed approach, respecting alignment only insofar as it is favoured by the caches. Ideas from Porat and Shalem [21] could prove beneficial in this regard.

What about other geometries? We analysed linear hash tables where objects are assigned random intervals. One could also consider a square hash table $(\mathbb{Z}_{\sqrt{n}})^2$ where objects are assigned random squares of size $\ell \times \ell$ (with no alignment requirement). We suspect that understanding the thresholds in such cases would require completely new techniques.

References

1. David Aldous and J. Michael Steele. *The Objective Method: Probabilistic Combinatorial Optimization and Local Weak Convergence*, pages 1–72. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. doi:10.1007/978-3-662-09444-0_1.
2. Stephan Beyer. Analysis of the Linear Probing Variant of Cuckoo Hashing. Master's thesis, Technische Universität Ilmenau, 2012. URL: http://gso.gbv.de/DB=2.1/PPNSET?PPN=685166759.
3. Julie Anne Cain, Peter Sanders, and Nicholas C. Wormald. The Random Graph Threshold for k-orientiability and a Fast Algorithm for Optimal Multiple-Choice Allocation. In *Proc. 18th SODA*, pages 469–476, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283433.
4. Martin Dietzfelbinger, Andreas Goerdt, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight Thresholds for Cuckoo Hashing via XORSAT. In *Proc. 37th ICALP (1)*, pages 213–225, 2010. doi:10.1007/978-3-642-14165-2_19.
5. Martin Dietzfelbinger and Christoph Weidling. Balanced Allocation and Dictionaries with Tightly Packed Constant Size Bins. In *Proc. 32nd ICALP*, pages 166–178, 2005. doi:10.1007/11523468_14.
6. Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theor. Comput. Sci.*, 380(1-2):47–68, 2007. doi:10.1016/j.tcs.2007.02.054.
7. Daniel Fernholz and Vijaya Ramachandran. The k-orientability Thresholds for $G_{n,p}$. In *Proc. 18th SODA*, pages 459–468, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283432.
8. Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space Efficient Hash Tables with Worst Case Constant Access Time. *Theory Comput. Syst.*, 38(2):229–248, 2005. doi:10.1007/s00224-004-1195-x.
9. Nikolaos Fountoulakis, Megha Khosla, and Konstantinos Panagiotou. The Multiple-Orientability Thresholds for Random Hypergraphs. In *Proc. 22nd SODA*, pages 1222–1236, 2011. URL: http://www.siam.org/proceedings/soda/2011/SODA11_092_fountoulakisn.pdf.
10. Nikolaos Fountoulakis and Konstantinos Panagiotou. Orientability of Random Hypergraphs and the Power of Multiple Choices. In *Proc. 37th ICALP (1)*, pages 348–359, 2010. doi:10.1007/978-3-642-14165-2_30.

11 Nikolaos Fountoulakis and Konstantinos Panagiotou. Sharp Load Thresholds for Cuckoo Hashing. *Random Struct. Algorithms*, 41(3):306–333, 2012. doi:10.1002/rsa.20426.

12 Alan M. Frieze and Páll Melsted. Maximum Matchings in Random Bipartite Graphs and the Space Utilization of Cuckoo Hash Tables. *Random Struct. Algorithms*, 41(3):334–364, 2012. doi:10.1002/rsa.20427.

13 Pu Gao and Nicholas C. Wormald. Load Balancing and Orientability Thresholds for Random Hypergraphs. In *Proc. 42nd STOC*, pages 97–104, 2010. doi:10.1145/1806689.1806705.

14 Svante Janson and Malwina J. Luczak. A simple solution to the k-core problem. *Random Struct. Algorithms*, 30(1-2):50–62, 2007. doi:10.1002/rsa.20147.

15 M. Leconte, M. Lelarge, and L. Massoulié. Convergence of multivariate belief propagation, with applications to cuckoo hashing and load balancing. In *Proc. 24th SODA*, pages 35–46, 2013. URL: http://dl.acm.org/citation.cfm?id=2627817.2627820.

16 Mathieu Leconte. Double hashing thresholds via local weak convergence. In *51st Annual Allerton Conference on Communication, Control, and Computing*, pages 131–137, 2013. doi:10.1109/Allerton.2013.6736515.

17 Eric Lehman and Rina Panigrahy. 3.5-Way Cuckoo Hashing for the Price of 2-and-a-Bit. In *Proc. 17th ESA*, pages 671–681, 2009. doi:10.1007/978-3-642-04128-0_60.

18 Marc Lelarge. A New Approach to the Orientation of Random Hypergraphs. In *Proc. 23rd SODA*, pages 251–264, 2012. URL: http://dl.acm.org/citation.cfm?id=2095139.

19 Michael Molloy. Cores in random hypergraphs and boolean formulas. *Random Struct. Algorithms*, 27(1):124–135, 2005. doi:10.1002/rsa.20061.

20 Rasmus Pagh and Flemming Friche Rodler. Cuckoo Hashing. *J. Algorithms*, 51(2):122–144, 2004. doi:10.1016/j.jalgor.2003.12.002.

21 Ely Porat and Bar Shalem. A Cuckoo Hashing Variant with Improved Memory Utilization and Insertion Time. In *Proc. 22nd DCC*, 2012. doi:10.1109/DCC.2012.41.

22 Stefan Walzer. Load thresholds for cuckoo hashing with overlapping blocks. *CoRR*, abs/1707.06855, 2017. arXiv:1707.06855.

Brief Announcement: On Secure m-Party Computation, Commuting Permutation Systems and Unassisted Non-Interactive MPC

Navneet Agarwal
Indian Institute of Technology Bombay
navneet@cse.iitb.ac.in

Sanat Anand
Indian Institute of Technology Bombay
sanat@cse.iitb.ac.in

Manoj Prabhakaran
Indian Institute of Technology Bombay
mp@cse.iitb.ac.in

Abstract

A fundamental problem in the theory of secure multi-party computation (MPC) is to characterize functions with more than 2 parties which admit MPC protocols with information-theoretic security against passive corruption. This question has seen little progress since the work of Chor and Ishai (2001), which demonstrated difficulties in resolving it. In this work, we make significant progress towards resolving this question in the important case of aggregating functionalities, in which m parties $P_1, \ldots, P_m$ hold inputs $x_1, \ldots, x_m$ and an aggregating party P_0 must learn $f(x_1, \ldots, x_m)$.

We give a necessary condition and a slightly stronger sufficient condition for f to admit a secure protocol. Both the conditions are stated in terms of an algebraic structure we introduce called Commuting Permutations Systems (CPS), which may be of independent combinatorial interest.

When our sufficiency condition is met, we obtain a perfectly secure protocol with minimal interaction, that fits the model of Non-Interactive MPC or NIMPC (Beimel et al., 2014), but without the need for a trusted party to generate correlated randomness. We define Unassisted Non-Interactive MPC (UNIMPC) to capture this variant. We also present an NIMPC protocol for all functionalities, which is simpler and more efficient than the one given in the prior work.

2012 ACM Subject Classification Theory of computation → Cryptographic protocols, Theory of computation → Complexity classes, Security and privacy → Mathematical foundations of cryptography

Keywords and phrases Secure Multi-Party Computation, Combinatorial Characterization, Latin Hypercube, Permutation Hypercube Complex

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.103

A fundamental problem in the theory of secure multi-party computation (MPC) is to characterize functions with *more than 2 parties* which admit MPC protocols with information-theoretic security against passive corruption. This question has seen little progress since the work of Chor and Ishai [2], which demonstrated difficulties in resolving it.

We report an ongoing work, in which we make significant progress towards resolving this question in the important case of *aggregating functionalities*: In an aggregating functionality, there are m parties $P_1, \ldots, P_m$ with inputs $x_1, \ldots, x_m$ and an aggregating party P_0 must learn $f(x_1, \ldots, x_m)$. Aggregating functionalities form a practically and theoretically important

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 103; pp. 103:1–103:4


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

class. In particular, it has been the subject of an influential line of study that started with the *minimal model for secure computation* of Feige, Kilian and Naor [4]. This model – sometimes refered to as the Private Simultaneous Messages (PSM) model – served as a precursor of important concepts like randomized encodings [5] that have proven useful in a variety of cryptographic applications. Recently, a strengthening of this model, called Non-Interactive MPC (NIMPC) was introduced by Beimel et al. [1], which is closer to standard MPC in terms of the security requirements. In both these models the severe restriction on the communication pattern often leads to simple and elegant protocols. Indeed, for specialized functions (like "Remote-OT" and AND) the original protocols developed in the PSM model [4] can also be shown to be optimal (or very nearly so) in terms of communication and randomness complexity [3, 9]. Similarly, Beimel et al. discovered several elegant NIMPC protocols for special classes of functions [1]. However, these protocols do not directly translate to MPC protocols *as these models include a trusted party* which sends correlated random variables to the parties in a pre-processing phase. The term aggregating functionality was coined in [8].

Our contributions in this work fall into three broad categories: (1) minimal models of computation, (2) algebraic-combinatorial classes of aggregating functionalities, and (3) positive and negative results relating the above two.

Minimal Models of MPC. The previous minimalistic models of MPC – PSM [4] and NIMPC [1] – admit secure protocols for all functions, unlike the full-fledged MPC model. Our minimalistic models (called UNIMPC* and UNIMPC) admit secure protocols only for functions which have secure protocols in the MPC model. While the previous models were proposed in the context of studying communication complexity of information-theoretic MPC, ours is perhaps the first significant model aimed at studying the feasibility of information-theoretic MPC.

UNIMPC stands for *Unassisted NIMPC* and, as the name suggests, removes the assistance from the trusted party in NIMPC: Instead the parties should securely compute the correlated randomness by themselves, in an offline phase. Unlike PSM and NIMPC, which allow trusted parties, *UNIMPC retains the standard security model of MPC*, allowing corruption of any set of parties. While MPC and NIMPC are incomparable in the sense that an MPC protocol does not yield an NIMPC protocol (because of the general communication pattern) and an NIMPC protocol does not yield an MPC protocol (because of the use of a trusted party), UNIMPC could be seen as a common denominator of these two secure computation models.

> *A UNIMPC protocol can be directly interpreted as an MPC protocol as well as an NIMPC protocol.*

UNIMPC* corresponds to a minimalistic version of UNIMPC, with protocols which have a single round of (simultaneous) communication among the parties before they get their inputs, followed by a single message from each party to the aggregator after they receive their input. (UNIMPC allows arbitrarily many rounds of communication prior to receiving inputs.) Understanding the gap between the classes of functionalities with UNIMPC and UNIMPC* protocols is closely related to understanding the power of multiparty secure sampling [7].

Commuting Permutations Systems. We identify an algebraic-combinatorial structure called Commuting Permutations System (CPS) and a sub-class called Commuting Permutation Subgroup Systems (CPSS).

Below S_n denotes the symmetric group – the group of all permutations of n elements.

▶ **Definition 1.** An (n, m)-*Commuting Permutations System* (CPS) is a collection $(X_1, \cdots, X_m)$ where for all $i \in [m]$, $X_i \subseteq S_n$ contains the identity permutation, and for any collection $(\pi_1, \cdots .\pi_m)$ with $\pi_i \in X_i$, and $\rho \in S_m$, $\pi_1 \circ \cdots \circ \pi_m(1) = \pi_{\rho(1)} \circ \cdots \circ \pi_{\rho(m)}(1)$.[1]

It is called an (n, m)-*Commuting Permutation Subgroups System* (CPSS) if each X_i is a subgroup of S_n.

An $(m+1)$-party aggregating functionality $f : X_1 \times \cdots \times X_m \to [n]$ is said to be a *CPS functionality* (resp. CPSS functionality) if $(X_1, \cdots, X_m)$ is an (n, m)-CPS (resp. (n, m)-CPSS) and for all $(\pi_1, \cdots, \pi_m) \in X_1 \times \cdots \times X_m$, $f(\pi_1, \cdots, \pi_m) = (\prod_{i \in [m]} \pi_i)(1)$.

Results. Our main results can be summarized as follows. Writing CPS (or CPSS) for class of functionalities that "embed" into a CPS (respectively, CPSS) functionality, and UNIMPC*, UNIMPC and MPC for classes of functionalities that admit the corresponding secure protocol, we have, for any number of parties,

$$\text{CPSS} \Rightarrow \text{UNIMPC}^* \Rightarrow \text{UNIMPC} \Rightarrow \text{MPC} \Rightarrow \text{CPS}.$$

Note that we leave an intriguing gap between the necessary and sufficient conditions. In particular we leave open the possibility that the set of functionalities with UNIMPC protocols is a strict subset of the set of aggregating functionalities with MPC protocols, and is a strict superset of aggregating functionalities with UNIMPC* protocols. However, these differences disappear for small number of parties: When the number of input parties is 2, we show that UNIMPC* $\Leftrightarrow$ CPS, and when the number of input parties is 3, UNIMPC $\Leftrightarrow$ CPS.

We also obtain a characterization of all "Latin hypercube functionalities" which have an MPC protocol, and show that they all have UNIMPC* protocol. This result relies on the above results, as well as on the existence of NIMPC protocols for every CPS functionality. For the sake of being self-contained we present a simple NIMPC protocol for general functionalities, which in fact turns out to be more efficient than the prior constructions [1, 6].

Our results could be seen as a step towards fully characterizing the functionalities with information-theoretic MPC protocols in various security models. For instance, for characterizing functionalities with UC secure protocols, aggregating functionalities remain the only class to be understood [8], and the sub-classes of aggregating functionalities identified in this work can serve as a starting point for understanding UC security. Similarly, the problem of characterizing symmetric functions (when all parties get the same output) as considered in [2] is still unsolved, but our positive results do present new possibilities there (because a passive-secure MPC protocol for an aggregating functionality can be readily converted into one for a symmetric functionality computing the same function).

References

1 Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 387–404. Springer, 2014. doi:10.1007/978-3-662-44381-1_22.

2 Benny Chor and Yuval Ishai. On privacy and partition arguments. *Information and Computation*, 167(1):2–9, 2001.

[1] Choice of 1 is arbitrary. Requiring identity permutation to always be part of each X_i is w.l.o.g., as a CPS without it will remain a CPS on adding it.

3 Deepesh Data, Manoj Prabhakaran, and Vinod Prabhakaran. On the communication complexity of secure computation. CoRR Report 1311.7584 available from `http://arxiv.org`, 2013.

4 Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *STOC*, pages 554–563. ACM, 1994. `doi:10.1145/195058.195408`.

5 Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, pages 294–304, 2000.

6 Satoshi Obana and Maki Yoshida. An efficient construction of non-interactive secure multiparty computation. In *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*, pages 604–614, 2016.

7 Manoj Prabhakaran and Vinod Prabhakaran. On secure multiparty sampling for more than two parties. In *Proceedings of the 2012 IEEE International Information Theory Workshop (ITW 2012)*, 2012.

8 Manoj Prabhakaran and Mike Rosulek. Cryptographic complexity of multi-party computation problems: Classifications and separations. In David Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 262–279. Springer, 2008. `doi:10.1007/978-3-540-85174-5_15`.

9 Sundara Rajan S, S. Rajakrishnan, A. Thangaraj, and V. Prabhakaran. Lower bounds and optimal protocols for three-party secure computation. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 1361–1365, July 2016.

Brief Announcement: Characterizing Demand Graphs for (Fixed-Parameter) Shallow-Light Steiner Network

Amy Babay
Johns Hopkins University, Baltimore, MD, USA
babay@cs.jhu.edu

Michael Dinitz
Johns Hopkins University, Baltimore, MD, USA
mdinitz@cs.jhu.edu

Zeyu Zhang
Johns Hopkins University, Baltimore, MD, USA
zyzhang92@gmail.com

Abstract

We consider the Shallow-Light Steiner Network problem from a fixed-parameter perspective. Given a graph G, a distance bound L, and p pairs of vertices $\{(s_i, t_i)\}_{i\in[p]}$, the objective is to find a minimum-cost subgraph G' such that s_i and t_i have distance at most L in G' (for every $i \in [p]$). Our main result is on the fixed-parameter tractability of this problem for parameter p. We exactly characterize the demand structures that make the problem "easy", and give FPT algorithms for those cases. In all other cases, we show that the problem is W[1]-hard. We also extend our results to handle general edge lengths and costs, precisely characterizing which demands allow for good FPT approximation algorithms and which demands remain W[1]-hard even to approximate.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Shallow-Light, Steiner Network, Fixed-Parameter Tractability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.104

Funding Supported in part by NSF award 1535887.

1 Introduction

We study *length-bounded* variants of Steiner Tree and Steiner Forest, which are related to (but still quite different from) *directed* variants. The direct setting assumes that the edges in the graph are directed. While in the length-bounded setting, we typically assume that the input graph and demands are undirected but each demand has a distance bound, and a solution is only feasible if every demand is connected within distance at most the given bound (rather than just being connected). One of the most basic problems of this form is the Shallow-Light Steiner Tree problem (SLST), where the demands $\{(s_i, t_i)\}_{i\in[p]}$ form a star with root $r = s_1 = s_2 = \cdots = s_p$ and there is a global length bound L (so in any feasible solution the distance from r to t_i is at most L for all $i \in [p]$). This problem has been studied extensively [8, 9, 6, 5], but if we generalize this problem to arbitrary demand pairs then we get the Shallow-Light Steiner Network problem, which to the best of our knowledge has received essentially no study.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 104; pp. 104:1–104:4


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Definition 1** (Shallow-Light Steiner Network)**.** Given a graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}^+$, a length function $l : E \to \mathbb{R}^+$, a distance bound L, and p pairs of vertices $\{(s_i, t_i)\}_{i\in[p]}$. The objective of SLSN is to find a minimum cost subgraph $G' = (V, S)$, such that for every $i \in [p]$, there is a path between s_i and t_i in G' with length at most L.

Let H be the graph with vertex set $\{s_1, \ldots, s_p, t_1, \ldots, t_p\}$ and edge set $\{(s_i, t_i)\}_{i\in[p]}$. We call H the *demand graph* of the problem.

Both the directed and the length-bounded settings share a dichotomy between considering either star demands (Directed Steiner Tree (DST)/SLST) or totally general demands (Directed Steiner Network (DSN)/SLSN). But this gives an obvious set of questions: what demand graphs make the problem "easy" (in FPT) and what demand graphs make the problem "hard" (W[1]-hard)? Recently Feldmann and Marx [4] gave a complete characterization for this for DSN, proving that if the demand graph is transitively equivalent to an "almost-caterpillar" then the problem is in FPT, and otherwise the problem is W[1]-hard.

While *a priori* there might not seem to be much of a relationship between the directed and the length-bounded settings, there are multiple folklore results that relate them, usually by means of some sort of layered graph. For example, any FPT algorithm for the DST problem can be turned into an FPT algorithm for SLST (with unit edge lengths) and vice versa. However, such a relationship is not known for more general demand graphs.

In light of these relationships between the directed and the length-bounded settings and the recent results of [4], it is natural to attempt to characterize the demand graphs that make SLSN easy or hard. We solve this problem, giving a complete characterization of easy and hard demand graphs. Informally, we show that SLSN is significantly harder than DSN: the only "easy" demand graphs are stars (in which case the problem is just SLST) and constant-size graphs. Even tiny modifications, like a star with a single independent edge, become W[1]-hard (despite being in FPT for DSN).

Connection to Overlay Routing: SLST and SLSN are particularly interesting due to their connection to overlay routing protocols that use so-called *dissemination graphs* for routing rather than traditional paths. Routing on dissemination graphs allows these systems to be significantly more robust, and a length bound corresponds to a latency bound, which is critical for many applications. Recently, Babay et al. [1] designed and built such a system, and demonstrated that it can achieve significantly greater reliability and timeliness than traditional routing with only a slight increase in cost. Finding good solutions to $2\binom{n}{2}$ different SLST instances and $\binom{n}{2}$ different SLSN instances is a crucial piece of this system (as these are the graphs on which routing happens). The search for fast algorithms for these instances was the main motivation behind this work. We refer the interested reader to [1] for a further discussion of this routing system and how it related to SLSN and SLST.

2 Our Results

We first define SLSN with respect to a class (set) of demand graphs.

▶ **Definition 2.** The Shallow-Light Steiner Network problem with restricted demand graph class $\mathcal{C}$ (SLSN$_\mathcal{C}$) is the SLSN problem with the additional restriction that the demand graph H of the problem must be isomorphic to some graph in $\mathcal{C}$.

We define $\mathcal{C}_\lambda$ as the class of all demand graphs with at most λ edges, and $\mathcal{C}^*$ as the class of all star demand graphs (there is a central vertex called the root, and every other vertex in the demand graph is adjacent to the root and only the root). Our main result is that

these are *precisely* the easy classes: We first prove that SLSN is in XP for parameter p (i.e. solvable in $n^{f(p)}$ time for some function f), which immediately implies that $\text{SLSN}_{\mathcal{C}_\lambda}$ can be solved in polynomial time if λ is a constant. Note that $\text{SLSN}_{\mathcal{C}^*}$ is precisely the SLST problem, for which a folklore FPT algorithm exists, thus $\text{SLSN}_{\mathcal{C}^*}$ (while NP-hard) is in FPT for parameter p. We also show that, for any other class $\mathcal{C}$ (i.e., any class which is not just a subset of $\mathcal{C}^* \cup \mathcal{C}_\lambda$ for some constant λ), the problem $\text{SLSN}_{\mathcal{C}}$ is W[1]-hard for parameter p. In other words, if the class of demand graphs includes arbitrarily large non-stars, then the problem is W[1]-hard parameterized by the number of demands. More formally, we prove the following theorems.

▶ **Theorem 3.** *The unit-length arbitrary-cost* SLSN *problem with parameter p is in* XP*, and it can be solved in $n^{O(p^4)}$ time.*

To prove this theorem, we first prove a structural lemma which shows that the optimal solution must be the union of several lowest cost paths with restricted length (these paths may be between steiner nodes, but we show that there cannot be too many). Then we just need to guess all the endpoints of these paths, as well as all the lengths of these paths. We prove that there are only $n^{O(p^4)}$ possibilities, and the running time is also $n^{O(p^4)}$.

▶ **Theorem 4.** *The unit-length arbitrary-cost* $\text{SLSN}_{\mathcal{C}^*}$ *problem is* FPT *for parameter p.*

As mentioned, $\text{SLSN}_{\mathcal{C}^*}$ is exactly the same as SLST, so we use a folklore reduction between SLST and DST to prove this theorem.

▶ **Theorem 5.** *If $\mathcal{C}$ is a recursively enumerable class, and $\mathcal{C} \not\subseteq \mathcal{C}_\lambda \cup \mathcal{C}^*$ for any constant λ, then* $\text{SLSN}_{\mathcal{C}}$ *is* W[1]*-hard for parameter p, even in the unit-length and unit-cost case.*

All of the above results are in the unit-length setting. We extend both our upper bounds and hardness results to handle arbitrary lengths, but with some extra complications. Even if $p = 1$, with arbitrary lengths and arbitrary costs the SLSN problem is equivalent to the Restricted Shortest Path problem, which is known to be NP-hard [7]. Therefore we can no longer hope for an FPT algorithm (with parameter p). So we change our notion of "easy" from "solvable in FPT" to "arbitrarily approximable in FPT": we show $(1+\varepsilon)$-approximation algorithms for the easy cases, and prove that there is no $\left(\frac{5}{4} - \varepsilon\right)$-approximation algorithm for the hard cases in $f(p) \cdot poly(n)$ time for any function f.

▶ **Theorem 6.** *For any constant $\lambda > 0$, there is a fully polynomial time approximation scheme (*FPTAS*) for the arbitrary-length arbitrary-cost* $\text{SLSN}_{\mathcal{C}_\lambda}$ *problem.*

▶ **Theorem 7.** *There is a $(1+\epsilon)$-approximation algorithm in $O(4^p \cdot poly(\frac{n}{\varepsilon}))$ time for the arbitrary-length arbitrary-cost* $\text{SLSN}_{\mathcal{C}^*}$ *problem.*

Our next theorem is analogous to Theorem 5, but since costs are allowed to be arbitrary we can prove stronger hardness of approximation (under stronger assumptions).

▶ **Theorem 8.** *Assume that the (randomized) Gap-Exponential Time Hypothesis [2] (Gap-ETH) holds. Let $\varepsilon > 0$ be a small constant, and $\mathcal{C}$ be a recursively enumerable class where $\mathcal{C} \not\subseteq \mathcal{C}_\lambda \cup \mathcal{C}^*$ for any constant λ. Then there is no $\left(\frac{5}{4} - \varepsilon\right)$-approximation algorithm in $f(p) \cdot n^{O(1)}$ time for* $\text{SLSN}_{\mathcal{C}}$ *for any function f, even with unit-lengths and polynomial-costs.*

Note that this theorem uses a much stronger assumption (Gap-ETH rather than W[1] $\neq$ FPT), which assumes that there is no (possibly randomized) algorithm running in $2^{o(n)}$ time that can distinguish whether a 3SAT formula is satisfiable or at most a $(1-\varepsilon)$-fraction of its clauses can be satisfied. This enables us to utilize the hardness result for a generalized version of the MCC problem from [3], which will allow us to modify our reduction from Theorem 5 to get hardness of approximation.

References

1 Amy Babay, Emily Wagner, Michael Dinitz, and Yair Amir. Timely, reliable, and cost-effective internet transport service using dissemination graphs. In *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017*, pages 1–12, 2017.

2 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-eth to fpt-inapproximability: Clique, dominating set, and more. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 743–754. IEEE, 2017.

3 Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized approximation algorithms for directed steiner network problems. *arXiv preprint arXiv:1707.06499*, 2017.

4 Andreas Emil Feldmann and Dániel Marx. The complexity landscape of fixed-parameter directed steiner network problems. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, volume 55 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

5 Longkun Guo, Kewen Liao, and Hong Shen. On the shallow-light steiner tree problem. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2014 15th International Conference on*, pages 56–60. IEEE, 2014.

6 Mohammad Taghi Hajiaghayi, Guy Kortsarz, and Mohammad R Salavatipour. Approximating buy-at-bulk and shallow-light k-steiner trees. *Algorithmica*, 53(1):89–103, 2009.

7 Refael Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations research*, 17(1):36–42, 1992.

8 Guy Kortsarz and David Peleg. Approximating shallow-light trees. Technical report, Association for Computing Machinery, New York, NY (United States), 1997.

9 Joseph Naor and Baruch Schieber. Improved approximations for shallow-light spanning trees. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 536–541. IEEE, 1997.

Brief Announcement: Zero-Knowledge Protocols for Search Problems

Ben Berger
Weizmann Institute of Science, Rehovot, Israel
ben.berger@weizmann.ac.il

Zvika Brakerski
Weizmann Institute of Science, Rehovot, Israel
zvika.brakerski@weizmann.ac.il

Abstract

We consider natural ways to extend the notion of Zero-Knowledge (ZK) Proofs beyond decision problems. Specifically, we consider *search problems*, and define zero-knowledge proofs in this context as interactive protocols in which the prover can establish the correctness of a solution to a given instance without the verifier learning anything beyond the intended solution, even if it deviates from the protocol.

The goal of this work is to initiate a study of Search Zero-Knowledge (search-ZK), the class of search problems for which such systems exist. This class trivially contains search problems where the validity of a solution can be efficiently verified (using a single message proof containing only the solution). A slightly less obvious, but still straightforward, way to obtain zero-knowledge proofs for search problems is to let the prover send a solution and prove in zero-knowledge that the instance-solution pair is valid. However, there may be other ways to obtain such zero-knowledge proofs, and they may be more advantageous.

In fact, we prove that there are search problems for which the aforementioned approach fails, but still search zero-knowledge protocols exist. On the other hand, we show sufficient conditions for search problems under which some form of zero-knowledge can be obtained using the straightforward way.

2012 ACM Subject Classification Theory of computation → Interactive proof systems, Theory of computation → Cryptographic protocols

Keywords and phrases Zero-Knowledge, Search Problems, Interactive Proofs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.105

Funding Supported by the Israel Science Foundation (Grant No. 468/14), Binational Science Foundation (Grants No. 2016726, 2014276), and by the European Union Horizon 2020 Research and Innovation Program via ERC Project REACT (Grant 756482) and via Project PROMETHEUS (Grant 780701).

Acknowledgements We thank Ofer Grossman and Oded Goldreich for helpful discussions.

1 Introduction

The notion of Zero-Knowledge Proofs (ZK-Proofs) introduced by Goldwasser, Micali and Rackoff [15] is one of the most insightful and influential in the theory of computing. Its tremendous impact came not only from having numerous applications but maybe more importantly from changing the way we think about proofs, communication and how to formalize such intuitive claims as a party "not learning anything" from an interaction. In a nutshell, a ZK-Proof is an interactive proof of some statement, i.e. an interaction between

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 105; pp. 105:1–105:5

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a prover P and a verifier V with the prover attempting to convince the verifier that some instance x belongs to a language L. In addition to the usual completeness and soundness, in the ZK scenario the prover wants to protect itself from revealing "too much information" to the verifier. Surely the verifier needs to learn that indeed $x \in L$, but nothing else beyond this fact should be revealed. Furthermore, even a *malicious* verifier that does not follow the prescribed protocol should not be able to trick the prover into revealing more information than intended. This intuitive statement is formalized using the *simulation paradigm*, the existence of a simulator machine S that takes an input $x \in L$ and a possibly cheating verifier V^* and samples from the view of V^* in the interaction (P, V^*) (up to negligible statistical or computational distance). Since the view of the verifier can essentially be produced (up to negligible distance) knowing only that $x \in L$, it clearly does not reveal anything beyond this fact.

Our Results. In this work we consider a setting where again the prover is concerned about revealing too much information to the verifier, but now in the context of *search problems*. That is, the prover would like to assist the verifier in learning a solution y to an instance x of some search problem, but would like to limit the verifier's ability to learn anything beyond the intended solution (or distribution of solutions).

While one's first intuition of a search problem is of one where it is efficient to verify a solution (i.e. searching for an NP witness), this is actually not the interesting setting here. In fact, in this case the prover can just send the witness, and the verifier verifies locally, so no additional information beyond the solution is revealed. One example one could consider is the isomorphic vertex problem: given two graphs (G_1, G_2) and a vertex v_1 in G_1, find a vertex v_2 in G_2 that is isomorphic to v_1 under *some* isomorphism.

Our first contribution is to formalize this notion using the simulation paradigm, as follows. We require that the prover for the interactive protocol is associated with a family of distributions $\{Y_x\}_x$ over solutions for each input x, intuitively corresponding to the distribution V is allowed to learn. We require that the view of any verifier can be simulated given only a sample y drawn from Y_x. To reduce the number of free parameters in the definition we propose to associate Y_x with the distribution of solutions output by an interaction of an honest prover with an honest verifier (note that importantly this refers to the distribution of solutions y output by the honest verifier and not to the honest verifier's entire view). Thus the zero-knowledge task becomes to ensure that no verifier (including the honest verifier) learns anything except the honest verifier's prescribed output. In terms of soundness, we require that V either outputs some valid solution for the search problem (if such exists), or rejects, except perhaps with small probability, even when interacting with a malicious prover.

Intuitively one could think that in order to achieve search-ZK, the prover should first sample a solution from Y_x, send it to the verifier and then prove in decision-ZK the validity of the solution (that is, that in a sense search-ZK is reducible to decision-ZK). Indeed almost all examples for protocols we have are roughly of this form. We investigate whether it is possible to provide a protocol of this form for any language in search-ZK, or whether there are some cases where other methods can achieve search-ZK but the aforementioned outline cannot. We define the class prefix-ZK to be the class of problems with protocols as above. We show that prefix-ZK has a complete problem (which we are unable to show for general search-ZK) and we show conditions under which some search-ZK systems can be transformed into prefix-ZK (for the same underlying search problem). Finally, we show that, perhaps counter-intuitively, search-ZK contains problems that are not in prefix-ZK, so at least in that sense the study of search-ZK may not be a derivative of the study of decision-ZK. Interestingly, this separation follows from showing that search-PSPACE *does not contain* search-IP, which may be of independent interest.

Lastly, we discuss the relation between search-ZK and the notion of *pseudo-deterministic* algorithms and protocols presented by Gat and Goldwasser [6] and further explored by Goldreich, Goldwasser, Grossman, Holden and Ron [7, 12, 16, 13, 14]. In a pseudo-deterministic protocol, not only should the distribution Y_x be a singleton y_x, but also the soundness requirement is that a malicious prover cannot make an honest verifier output a solution different from y_x (except with small probability). One of the advantages of pseudo-deterministic protocols is that they allow for soundness amplification for search problems. We show that the isomorphic vertex problem indeed has a pseudo-deterministic search-ZK protocol, suggesting that achieving strong soundness together with strong privacy is possible in some interesting cases.

Related Notions. The first related notion is that of secure multiparty computation (MPC) by Yao [20] and Goldreich, Micali and Wigderson [8]. For the purpose of this work, the relevant setting is of secure *two-party* computation where two parties A, B with inputs x_A, x_B wish to compute values y_A, y_B which depend on both inputs. The privacy requirement is that each party does not learn more than its intended output. It would appear that setting $A = P$, $B = V$, and defining F_B appropriately to output what the verifier is allowed to learn, should result in a search-ZK protocol. However, looking more closely, the complexity of an MPC protocol scales with the complexity of the function F_B, which in general scales with the complexity of the prover's functionality. If the prover's functionality is not in NP, then MPC cannot be used. MPC appears to be useful in the restricted case of computational search-ZK for search problems that can be computed as a function of an NP witness. Our isomorphic vertex problem falls into that category (with the NP witness being an isomorphism), however for isomorphic vertex we have a *statistical* search-ZK protocol. For statistical search-ZK, the MPC methodology does not seem to be useful, since information theoretically secure *two-party* computation is not possible [3, 2].

Another related like of work is concerned with privacy of approximation algorithms, initiated by Feigenbaum et al. [5] and Halevi et al. [17], and further studied by Beimel et al. [1]. The setting in these works is quite different from ours. Their ideal setting is where a solution to some search problem is posted without revealing the input (e.g. output a vertex cover for some graph without revealing the edges of the graphs). The problem arises when solving exactly is hard and an approximation algorithm is used instead. Their goal is to show that the approximate solution does not reveal more information than the exact solution. Note that in this setting there is no soundness requirement (in fact, a client cannot be convinced that a solution is correct since it does not have the input).

Future Directions. Our work is far from being an exhaustive study of search-ZK, and we hope to open the door for additional study. One direction of research is designing search-ZK protocols for other problems of interest, and more importantly general approaches for search-ZK for classes of problems. The question of whether search-ZK has complete problems in the computational and statistical setting remains open. Another intriguing line of inquiry, which may also be helpful for resolving the above, is whether we can translate the extensive body of work on statistical ZK protocols [4, 18, 9, 11, 10, 19] into the search regime.

References

1 Amos Beimel, Paz Carmi, Kobbi Nissim, and Enav Weinreb. Private approximation of search problems. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 119–128. ACM, 2006. doi:10.1145/1132516.1132533.

2 Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10, 1988. doi:10.1145/62212.62213.

3 David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 11–19, 1988. doi:10.1145/62212.62214.

4 Ivan Damgård, Oded Goldreich, Tatsuaki Okamoto, and Avi Wigderson. Honest verifier vs dishonest verifier in public coin zero-knowledge proofs. In Don Coppersmith, editor, *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, volume 963 of *Lecture Notes in Computer Science*, pages 325–338. Springer, 1995. doi:10.1007/3-540-44750-4_26.

5 Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 927–938. Springer, 2001. doi:10.1007/3-540-48224-5_75.

6 Eran Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:136, 2011. URL: http://eccc.hpi-web.de/report/2011/136.

7 Oded Goldreich, Shafi Goldwasser, and Dana Ron. On the possibilities and limitations of pseudodeterministic algorithms. In Robert D. Kleinberg, editor, *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, pages 127–138. ACM, 2013. doi:10.1145/2422436.2422453.

8 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

9 Oded Goldreich, Amit Sahai, and Salil P. Vadhan. Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 399–408. ACM, 1998. doi:10.1145/276698.276852.

10 Oded Goldreich, Amit Sahai, and Salil P. Vadhan. Can statistical zero knowledge be made non-interactive? or on the relationship of SZK and NISZK. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 467–484. Springer, 1999. doi:10.1007/3-540-48405-1_30.

11 Oded Goldreich and Salil P. Vadhan. Comparing entropies in statistical zero knowledge with applications to the structure of SZK. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity, Atlanta, Georgia, USA, May 4-6, 1999*, page 54. IEEE Computer Society, 1999. doi:10.1109/CCC.1999.766262.

12 Shafi Goldwasser and Ofer Grossman. Perfect bipartite matching in pseudo-deterministic RNC. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:208, 2015. URL: http://eccc.hpi-web.de/report/2015/208.

13 Shafi Goldwasser and Ofer Grossman. Bipartite perfect matching in pseudo-deterministic NC. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 87:1–87:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.87.

14 Shafi Goldwasser, Ofer Grossman, and Dhiraj Holden. Pseudo-deterministic proofs. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 17:1–17:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ITCS.2018.17.

15 Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. doi:10.1137/0218012.

16 Ofer Grossman. Finding primitive roots pseudo-deterministically. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:207, 2015. URL: http://eccc.hpi-web.de/report/2015/207.

17 Shai Halevi, Robert Krauthgamer, Eyal Kushilevitz, and Kobbi Nissim. Private approximation of np-hard functions. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 550–559. ACM, 2001. doi:10.1145/380752.380850.

18 Tatsuaki Okamoto. On relationships between statistical zero-knowledge proofs. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 649–658. ACM, 1996. doi:10.1145/237814.238016.

19 Amit Sahai and Salil P. Vadhan. A complete problem for statistical zero knowledge. *J. ACM*, 50(2):196–249, 2003. doi:10.1145/636865.636868.

20 Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

Brief Announcement: Relaxed Locally Correctable Codes in Computationally Bounded Channels

Jeremiah Blocki
Department of Computer Science, Purdue University, West Lafayette, Indiana, USA
jblocki@purdue.edu

Venkata Gandikota
Department of Computer Science, Johns Hopkins University, Baltimore, Maryland, USA
gv@jhu.edu

Elena Grigorescu
Department of Computer Science, Purdue University, West Lafayette, Indiana, USA
elena-g@purdue.edu

Samson Zhou
Department of Computer Science, Purdue University, West Lafayette, Indiana, USA
samsonzhou@gmail.com

Abstract

We study variants of locally decodable and locally correctable codes in computationally bounded, adversarial channels, under the assumption that collision-resistant hash functions exist, and with no public-key or private-key cryptographic setup. Specifically, we provide constructions of *relaxed locally correctable* and *relaxed locally decodable codes* over the binary alphabet, with constant information rate, and poly-logarithmic locality. Our constructions compare favorably with existing schemes built under much stronger cryptographic assumptions, and with their classical analogues in the computationally unbounded, Hamming channel. Our constructions crucially employ *collision-resistant hash functions* and *local expander graphs*, extending ideas from recent cryptographic constructions of memory-hard functions.

2012 ACM Subject Classification Theory of computation → Error-correcting codes

Keywords and phrases Relaxed locally correctable codes, computationally bounded channels, local expanders

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.106

Related Version We defer all proofs to [5], `https://arxiv.org/abs/1803.05652`, whose results are described in this announcement.

Introduction

An error-correcting code is a tuple $(\mathsf{Enc}, \mathsf{Dec})$, where a sender encodes a *message* m of k symbols from an alphabet Σ, into a *codeword* c of block-length n, consisting of symbols over the same alphabet, using encoding algorithm $\mathsf{Enc} : \Sigma^k \to \Sigma^n$; a receiver uses decoding algorithm $\mathsf{Dec} : \Sigma^n \to \Sigma^k$ to recover the message m from a received word $w \in \Sigma^n$. Codes with both large *information rate*, defined as k/n, and large *error rate*, which is the tolerable fraction of errors in the received word, are most desirable.

In modern uses of error-correcting codes, one may only need to recover small portions of the message, such as a single bit. Given an index $i \in [n]$, and oracle access to w, a local decoder must make only $q = o(n)$ queries into w, and output the bit m_i. The *locality* of the decoder is defined to be q. Codes that admit such fast decoders are called *locally decodable*

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 106; pp. 106:1–106:4


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

codes (LDCs) [12, 15]. A related notion is that of *locally correctable codes* (LCCs), where the local decoder must output bits of the codeword c, instead of bits of the message m.

Ben-Sasson *et al.* [4] propose the notion of *relaxed locally decodable codes* (RLDCs) as a way to remedy the dramatic tradeoffs of classical LDCs. In this notion the decoding algorithm is allowed to output "$\perp$" sometimes; however, it should not output an incorrect value too often. More formally, given $i \in [k]$, and oracle access to the received word w, which is assumed to be relative close to some codeword $c = \mathsf{Enc}(m) \in \Sigma^n$, the local decoder: (1) outputs m_i if $w = c$; (2) outputs either m_i or $\perp$ with probability 2/3, otherwise; and, (3) the set of indices i such that the decoder outputs m_i (the correct value) with probability 2/3, has size at least $\rho \cdot k$ for some constant $\rho > 0$. The relaxed definition allows them to achieve RLDCs with constant query complexity and blocklength $n = k^{1+\epsilon}$.

Recently, Gur *et al.* [9] introduce the analogous notion of *relaxed locally correctable codes* (RLCCs). The results in [9] obtain significantly better parameters for RLCCs than for classical LCCs; namely, they construct RLCCs with constant query complexity, polynomial block length, and constant error rate, and RLCCs with quasipolynomial query complexity, linear blocklength (constant rate), with the caveat that the error rate is subconstant. These results immediately extend to RLDCs, since their codes are *systematic*, meaning that the initial part of the encoding consists of the message itself.

Computationally bounded, adversarial channels

All the above constructions of local codes assume a channel that may introduce a bounded number of adversarial errors, and the channel has as much time as it needs to decide what positions to corrupt (i.e., the standard Hamming channel). In this work we study RLDCs and RLCCs in the *computationally bounded, adversarial channel* model, introduced by Lipton [13]. In this model we require that the adversary who determines which bits of the codeword to corrupt must run in probabilistic polynomial time. Existing constructions of locally correctable codes in the computationally bounded channel model typically require preliminary trusted setup [14, 10, 11, 7] (e.g., the sender and receiver have established cryptographic keys). By contrast, our results do not require the sender and the receiver to share a secret key for a symmetric cipher, nor do we assume the existence of a public key infrastructure (PKI). Instead our constructions are based on the existence of collision-resilient hash functions, a standard cryptographic assumption. Because the parameters of a collision-resistant hash function are public, *any* party (sender/receiver/attacker) is able to evaluate it.

Our Contributions

We now define our model. Our codes interact with an adversarial channel, so their strength is measured both in their error correction and locality capabilities (as for RLCCs/RLDCs), and in the security they provide against the channel.

▶ **Definition 1.** A *computational adversarial channel* $\mathcal{A}$ with error rate τ is an algorithm that interacts with a local code $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ of rate k/n in rounds, as follows. In each round of the execution, given a security parameter λ,

(1) Generate $s \leftarrow \mathsf{Gen}(1^\lambda)$; s is public, so Enc, Dec, and $\mathcal{A}$ have access to s

(2) The channel $\mathcal{A}$ on input s hands a message x to the sender.

(3) The sender computes $c = \mathsf{Enc}(s, x)$ and hands it back to the channel (in fact, the channel can compute c without this interaction).

(4) The channel $\mathcal{A}$ corrupts at most τn entries of c to obtain a word $w \in \Sigma^n$; w is given to the receiver's Dec with query access, together with a challenge index $i \in [n]$

(5) The receiver outputs $b \leftarrow \mathsf{Dec}^w(s, i)$.

(6) We define $\mathcal{A}(s)$'s *probability of fooling* Dec on this round to be $p_{\mathcal{A},s} = \Pr[b \notin \{\perp, c_i\}]$, where the probability is taken only over the randomness of the $\mathsf{Dec}^w(s, i)$. We say that $\mathcal{A}(s)$ is γ-successful *at fooling* Dec if $p_{\mathcal{A},s} > \gamma$. We say that $\mathcal{A}(s)$ is ρ-successful *at limiting* Dec if $|\mathsf{Good}_{\mathcal{A},s}| < \rho \cdot n$, where $\mathsf{Good}_{\mathcal{A},s} \subseteq [n]$ is the set of indices j such that $\Pr[\mathsf{Dec}^w(s, j) = c_j] > \frac{2}{3}$. We use $\mathsf{Fool}_{\mathcal{A},s}(\gamma, \tau, \lambda)$ (resp. $\mathsf{Limit}_{\mathcal{A},s}(\rho, \tau, \lambda)$) to denote the event that the attacker was γ-successful at fooling Dec (resp. ρ-successful at limiting Dec) on this round.

▶ **Definition 2** ((Computational) Relaxed Locally Correctable Codes (CRLCC))**.** A local code $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a $(q, \tau, \rho, \gamma, \mu(\cdot))$-CRLCC *against a class* $\mathbb{A}$ of adversaries, if Dec^w makes at most q queries to w and satisfies the following:

(1) For all public seeds s if $w \leftarrow \mathsf{Enc}(s, x)$ then $\mathsf{Dec}^w(s, i)$ outputs $b = (\mathsf{Enc}(s, x))_i$.

(2) For all $\mathcal{A} \in \mathbb{A}$ we have $\Pr[\mathsf{Fool}_{\mathcal{A},s}(\gamma, \tau, \lambda)] \leq \mu(\lambda)$, where the randomness is taken over the selection of $s \leftarrow \mathsf{Gen}(1^\lambda)$ as well as $\mathcal{A}$'s random coins.

(3) For all $\mathcal{A} \in \mathbb{A}$ we have $\Pr[\mathsf{Limit}_{\mathcal{A},s}(\rho, \tau, \lambda)] \leq \mu(\lambda)$, where the randomness is taken over the selection of $s \leftarrow \mathsf{Gen}(1^\lambda)$ as well as $\mathcal{A}$'s random coins.

When $\mu(\lambda) = 0$ and $\mathbb{A}$ is the set of all (computationally unbounded) channels we say that the code is a (q, τ, ρ, γ)-RLCC. When $\mu(\cdot)$ is a negligible function *and* $\mathbb{A}$ is restricted to the set of all probabilistic polynomial time (PPT) attackers we say that the code is a (q, τ, ρ, γ)-CRLCC (computational relaxed locally correctable code). We say that a code that satisfies conditions 1 and 2 is a *Weak CRLCC*, while a code satisfying 1, 2 and 3 is a *Strong CRLCC*.

Results and Techniques. At a technical level our constructions use *local expander graphs* and *collision resistant hash functions* (CRHF) as main building blocks.

Local expanders have several nice properties that have been recently exploited in the design and analysis of secure memory hard functions [8, 1, 2, 6, 3]. Given a graph $G = (V, E)$ and distinguished subsets $A, B \subseteq V$ of nodes such that A and B are disjoint and $|A| = |B|$, we say that the pair (A, B) contains a δ-*expander* if for all $X \subseteq A$ and $Y \subseteq B$ with $|X| > \delta|A|$ and $|Y| > \delta|B|$, there is an edge connecting X and Y. A δ-*local expander* is a directed acyclic graph G with n nodes $V(G) = \{1, \ldots, n\}$ with the property that for *any* radius $r > 0$ and *any* node $v \geq 2r$ the sets $A = \{v - 2r + 1, \ldots, v - r\}$ and $B = \{v - r + 1, \ldots, v\}$ contain a δ-expander. For any constant $\delta > 0$ it is possible to construct a δ-local expander with the property that $\mathsf{indeg}(G) \in \mathcal{O}(\log n)$ and $\mathsf{outdeg}(G) \in \mathcal{O}(\log n)$ [8, 3].

A CRHF function is a pair (GenH, H) of PPT algorithms, where for security parameter 1^λ, GenH outputs a public seed $s \in \{0,1\}^*$ that is passed as the first input to $H : \{0,1\}^* \times \Sigma^* \to \Sigma^{\ell(\lambda)}$. The *length* of the hash function is $\ell(\lambda)$. (GenH, H) is said to be collision-resistant if any PPT adversary can produce a collision with only negligible probability.

Using local expander graphs we first construct Weak CRLCCs and then Strong CRLCCs against PPT adversaries, under the assumption that CRHF*s* exist. Our constructions are systematic, so they immediately imply the existence of CRLDCs with the same parameters.

▶ **Theorem 3.** *Assuming the existence of a CRHF* (GenH, H) *with length* $\ell(\lambda)$*, there exist constants* $0 < \tau, \rho, \gamma < 1$ *and a negligible function* μ*, such that there exists a constant rate* $(\mathsf{polylog}\, n, \tau, \rho, \gamma, \mu(\cdot))$*-Strong CRLCC of blocklength* n *over the binary alphabet. In particular, if* $\ell(\lambda) = \mathsf{polylog}\, \lambda$ *and* $\lambda \in \Theta(n)$ *then the code is a* $(\mathsf{polylog}\, n, \tau, \rho, \gamma, \mu(\cdot))$*-Strong CRLCC.*

The classical RLCCs of [9] achieve $(\log n)^{\mathcal{O}(\log \log n)}$ query complexity, constant information rate, but subconstant error rate, in the Hamming channel.

References

1. Joël Alwen, Jeremiah Blocki, and Ben Harsha. Practical graphs for optimal side-channel resistant memory-hard functions. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 1001–1017. ACM Press, / nov 2017.
2. Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 3–32. Springer, Heidelberg, 2017.
3. Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. In *Advances in Cryptology - EUROCRYPT - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*, 2018. (to appear).
4. Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust pcps of proximity, shorter pcps, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006. A preliminary version appeared in the Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC).
5. Jeremiah Blocki, Venkata Gandikota, Elena Grigorescu, and Samson Zhou. Relaxed locally correctable codes in computationally bounded channels. *CoRR*, abs/1803.05652, 2018. `arXiv:1803.05652`.
6. Jeremiah Blocki and Samson Zhou. On the depth-robustness and cumulative pebbling cost of Argon2i. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 445–465. Springer, Heidelberg, 2017.
7. Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 402–414, 1999.
8. Paul Erdös, Ronald L. Graham, and Endre Szemeredi. On sparse graphs with dense long paths. Technical report, Stanford University, Stanford, CA, USA, 1975.
9. Tom Gur, Govind Ramnarayan, and Ron D. Rothblum. Relaxed locally correctable codes. In *9th Innovations in Theoretical Computer Science Conference, ITCS*, pages 27:1–27:11, 2018.
10. Brett Hemenway and Rafail Ostrovsky. Public-key locally-decodable codes. In *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Proceedings*, pages 126–143, 2008.
11. Brett Hemenway, Rafail Ostrovsky, Martin J. Strauss, and Mary Wootters. Public key locally decodable codes with short keys. In *14th International Workshop, APPROX, and 15th International Workshop, RANDOM, Proceedings*, pages 605–615, 2011.
12. Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 80–86, 2000.
13. Richard J. Lipton. A new approach to information theory. In *STACS, 11th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, pages 699–708, 1994.
14. Rafail Ostrovsky, Omkant Pandey, and Amit Sahai. Private locally decodable codes. In *Automata, Languages and Programming, 34th International Colloquium, ICALP, Proceedings*, pages 387–398, 2007.
15. Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom generators without the XOR lemma (abstract). In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity, Atlanta, Georgia, USA, May 4-6, 1999*, page 4, 1999.

Brief Announcement: Approximation Schemes for Geometric Coverage Problems

Steven Chaplick

Lehrstuhl für Informatik I, Universität Würzburg, Germany
steven.chaplick@uni-wuerzburg.de
https://orcid.org/0000-0003-3501-4608

Minati De[1]

Department of Mathematics, Indian Institute of Technology Delhi, India
minati@maths.iitd.ac.in

Alexander Ravsky

Pidstryhach Institute for Applied Problems of Mechanics and Mathematics, National Academy of Science of Ukraine, Lviv, Ukraine
oravsky@mail.ru

Joachim Spoerhase[2]

Lehrstuhl für Informatik I, Universität Würzburg, Germany;
Institute of Computer Science, University of Wrocław, Poland.
joachim.spoerhase@uni-wuerzburg
https://orcid.org/0000-0002-2601-6452

Abstract

In this announcement, we show that the classical Maximum Coverage problem (MC) admits a PTAS via local search in essentially all cases where the corresponding instances of Set Cover (SC) admit a PTAS via the local search approach by Mustafa and Ray [7]. As a corollary, we answer an open question by Badanidiyuru, Kleinberg, and Lee [1] regarding half-spaces in $\mathbb{R}^3$ thereby settling the existence of PTASs for essentially all natural cases of geometric MC problems. As an intermediate result, we show a color-balanced version of the classical planar subdivision theorem by Frederickson [5]. We believe that some of our ideas may be useful for analyzing local search in other settings involving a hard cardinality constraint.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases balanced separators, maximum coverage, local search, approximation scheme, geometric approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.107

Related Version A full version of the paper is available at [2], `https://arxiv.org/abs/1607.06665`.

[1] Partially supported by DST-INSPIRE Faculty Grant (DST-IFA-14-ENG-75)
[2] Partially supported by Polish National Science Centre grant 2015/18/E/ST6/00456

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 107; pp. 107:1–107:4

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction and Contribution

Let U be a set of ground elements, $\mathcal{F} \subseteq 2^U$ be a family of subsets of U and k be a positive integer. The Maximum Coverage (MC) problem asks for a k-subset $\mathcal{F}'$ of $\mathcal{F}$ such that the number $|\bigcup \mathcal{F}'|$ of ground elements covered by $\mathcal{F}'$ is maximized. In the closely related Set Cover problem (SC), the goal is to cover all ground elements using as few sets as possible. Both problems are among the most fundamental NP-hard optimization problems and their approximability is in general well understood [3, 4].

We examine the approximability of *geometric* MC problems. Badanidiyuru et al. [1] provided fixed-parameter approximation schemes for the large class of MC problems with bounded VC dimension. The running times of these schemes are polynomial in $|U|$ and $|\mathcal{F}|$ but *exponential* in k, in the VC dimension, and in the reciprocal of the error parameter $\epsilon > 0$. Prior to their work [1], Mustafa and Ray [7] had shown that local search yields PTASs for many geometric SC problems where a naturally defined *exchange graph* between a locally optimum solution and a globally optimum solution is *planar*. While the range of settings solvable by the fixed-parameter approximation approach of Badanidiyuru et al. is in principle even broader, the approach of Mustafa and Ray gives PTASs with strictly polynomial running time while still encompassing essentially all of the natural cases of (unweighted) SC that are not known to be APX-hard. It is thus an interesting question if their approach gives PTASs (with strictly polynomial running times) also for the corresponding MC problems.

A difficulty in carrying over this approach from SC to MC lies in the hard cardinality constraint in MC. For SC, Mustafa and Ray used the planar subdivision theorem by Frederickson [5] to subdivide the above-mentioned planar exchange graph into *small* pieces where each one provides a candidate swap. This subdivision may, however, be arbitrarily unbalanced with respect to the two feasible solutions forming the node set of the exchange graph. Hence a direct application of this approach would be in conflict with the cardinality constraint. Another difficulty comes from the different objective functions of MC and SC and that the analysis of Mustafa and Ray exploits that *all* ground elements are covered.

In this announcement we summarize how to overcome these issues (see [2] for the full version). A key step in our proof is that the pieces of an (unbalanced) subdivision obtained Frederickson's theorem [5] can be recombined in a careful way to obtain a *color-balanced* version of that theorem (see Theorem 2). Also the subsequent analysis of the performance guarantee requires some new ideas because of the other above-mentioned difficulties, and because our colored subdivision cannot not achieve perfect but only rough balance.

In a b-local search for MC, we start with any feasible solution. We perform a profitable *swap* of cardinality b as long as there is one. The timing is polynomial for constant b.

2 Our Results

For a graph G, a subset S of $V(G)$ is an *α-balanced separator* when its removal breaks G into two collections of connected components such that each collection contains at most an α fraction of $V(G)$ where $\alpha \in [\frac{1}{2}, 1)$ and α is a constant. The size of a separator S is the number of vertices it contains. For a non-decreasing sublinear function f, a subgraph-closed class of graphs is said to be *f-separable* if there is an $\alpha \in [\frac{1}{2}, 1)$ such that for any $n > 2$, any n-vertex graph in the class has an α-balanced separator of size at most $f(n)$.

▶ **Definition 1.** A class $\mathcal{C}$ of instances of MC is called *f-separable* if for any two disjoint feasible solutions $\mathcal{F}$ and $\mathcal{F}'$ of any instance in $\mathcal{C}$ there exists an f-separable graph G with node set $\mathcal{F} \cup \mathcal{F}'$ with the following *exchange* property. If there is a ground element $u \in U$

that is covered both by $\mathcal{F}$ and $\mathcal{F}'$ then there exists an edge (S, S') in G with $S \in \mathcal{F}$ and $S' \in \mathcal{F}'$ with $u \in S \cap S'$.

Note that a class of MC instances where each instance admits planar exchange graphs is $O(\sqrt{n})$-separable.

▶ **Theorem 2.** *Let $\mathcal{G}$ be a subgraph-closed f-separable graph class and G be a 2-colored n-vertex graph in $\mathcal{G}$ with color classes Γ_1, Γ_2 such that $|\Gamma_2| \geq |\Gamma_1|$. For any q and $r \ll n$ where r is suitably large, there is an integer $t \in \Theta(\frac{n}{q \cdot r})$ such that V can be partitioned into $t+1$ sets $\mathcal{X}, V_1, \ldots, V_t$ where c_1, c_2 are constants (depending only on f) and there is an integer $q' \in [q, 2q-1]$ satisfying the following properties.*

- **(i)** *$N(V_i) \cap V_j = \emptyset$ for each $i \neq j$ and $\mathcal{X} = \bigcup_i N(V_i)$,*
- **(ii)** *$|V_i| \geq \frac{q' \cdot r}{2}$ and $|V_i| \leq 2 \cdot (q'+1) \cdot r$ for each i,*
- **(iii)** *$|N(V_i)| \leq c_1 \cdot q \cdot f(r)$ for each i (thus, $|\mathcal{X}| \leq \sum_{i=1}^{t} |N(V_i)| \leq \frac{c_2 \cdot f(r) \cdot n}{r}$),*
- **(iv)** *$\left| |V_i \cap \Gamma_1| - \frac{|\Gamma_1|}{|\Gamma_2|} \cdot |V_i \cap \Gamma_2| \right| \leq 4 \cdot r$ for each i.*

▶ **Theorem 3.** *For any non-decreasing strictly sublinear function f, every f-separable class of MC instances (closed under removing elements and sets) admits a PTAS via local search.*

The following theorem describes several cases of MC that can be solved by our approach. Therein, we refer to several maximization versions of classical minimum covering problems (such as Vertex Cover). For example, in Maximum Vertex Cover, we are given a graph G and number k and we want to find a k vertices which *cover* as many edges as possible. The others are defined analogously. For a definition of 1.5D Terrain Guarding we refer to Krohn et al. [6].

▶ **Theorem 4.** *Local search gives a PTAS for:*

(V) *the* Maximum Vertex Cover *problem on f-separable and subgraph-closed graph classes,*

(T) *the* Maximum 1.5D Terrain Guarding *problem.*

and the following classes of MC problems:

(C_1) *the set of ground elements is a set of points in $\mathbb{R}^3$, and the family of subsets is induced by a set of halfspaces in $\mathbb{R}^3$.*

(C_2) *the set of ground elements is a set of points in $\mathbb{R}^2$, and the family of subsets is induced by a set of convex pseudodisks (a set of convex objects where any two objects can have at most two intersections in their boundary).*

and the following Maximum Hitting Set *problems:*

(H_1) *the set of ground elements is a set of points in $\mathbb{R}^2$, and the set of ranges is induced by a set of r-admissible regions (this includes pseudodisks, same-height axis-parallel rectangles, circular disks, translates of convex objects).*

(H_2) *the set of ground elements is a set of points in $\mathbb{R}^3$, and the set of ranges is induced by a set of halfspaces in $\mathbb{R}^3$.*

and Maximum Dominating Set *problems in each of the following graph classes:*

(D_1) *intersection graphs of homothetic copies of convex objects (which includes arbitrary squares, regular k-gons, translated and scaled copies of a convex object).*

(D_2) *non-trivial minor-closed graph classes.*

References

1 Ashwinkumar Badanidiyuru, Robert Kleinberg, and Hooyeon Lee. Approximating low-dimensional coverage problems. In *Symp. Computational Geometry (SoCG'12)*, pages 161–170, 2012. doi:10.1145/2261250.2261274.

2 Steven Chaplick, Minati De, Alexander Ravsky, and Joachim Spoerhase. Approximation schemes for geometric coverage problems. *CoRR*, 2016. arXiv:1607.06665.

3 Gérard Cornuéjols, George L. Nemhauser, and Laurence A. Wolsey. Worst-case and probabilistic analysis of algorithms for a location problem. *Operations Research*, 28(4):847–858, 1980. doi:10.1287/opre.28.4.847.

4 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

5 Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987.

6 Erik Krohn, Matt Gibson, Gaurav Kanade, and Kasturi R. Varadarajan. Guarding terrains via local search. *J. of Computational Geometry*, 5(1):168–178, 2014. URL: http://jocg.org/index.php/jocg/article/view/128.

7 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.

Brief Announcement: Bayesian Auctions with Efficient Queries

Jing Chen
Department of Computer Science, Stony Brook University, Stony Brook, NY 11794, USA
jingchen@cs.stonybrook.edu

Bo Li
Department of Computer Science, Stony Brook University, Stony Brook, NY 11794, USA
boli2@cs.stonybrook.edu

Yingkai Li
Department of Computer Science, Stony Brook University, Stony Brook, NY 11794, USA
yingkli@cs.stonybrook.edu

Pinyan Lu
Institute for Theoretical Computer Science, Shanghai University of Finance and Economics, Shanghai 200433, China
lu.pinyan@mail.shufe.edu.cn

Abstract

Generating good revenue is one of the most important problems in Bayesian auction design, and many (approximately) optimal dominant-strategy incentive compatible (DSIC) Bayesian mechanisms have been constructed for various auction settings. However, most existing studies do not consider the complexity for the seller to carry out the mechanism. It is assumed that the seller knows "each single bit" of the distributions and is able to optimize perfectly based on the entire distributions. Unfortunately this is a strong assumption and may not hold in reality: for example, when the value distributions have exponentially large supports or do not have succinct representations.

In this work we consider, for the first time, the *query complexity* of Bayesian mechanisms. We only allow the seller to have limited oracle accesses to the players' value distributions, via *quantile queries* and *value queries*. For a large class of auction settings, we prove *logarithmic* lower-bounds for the query complexity for any DSIC Bayesian mechanism to be of any constant approximation to the optimal revenue. For single-item auctions and multi-item auctions with unit-demand or additive valuation functions, we prove *tight* upper-bounds via efficient query schemes, without requiring the distributions to be regular or have monotone hazard rate. Thus, in those auction settings the seller needs to access much less than the full distributions in order to achieve approximately optimal revenue.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design

Keywords and phrases The complexity of Bayesian mechanisms, quantile queries, value queries

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.108

Related Version A full version of the paper is available at [9], https://arxiv.org/abs/1804.07451.

Acknowledgements This work has been partially supported by NSF CAREER Award No. 1553385, NSF of China Grant No. 61741209 and the Fundamental Research Funds for the Central Universities. Part of this work was done when the first three authors were visiting Shanghai University of Finance and Economics.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 108; pp. 108:1–108:4

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

An important problem in Bayesian mechanism design is to design auctions that (approximately) maximize the seller's expected revenue. More precisely, in a Bayesian multi-item auction a seller has m heterogenous items to sell to n players. Each player i has a private value for each item j, v_{ij}; and each v_{ij} is independently drawn from some prior distribution $\mathcal{D}_{ij}$. When the prior distribution $\mathcal{D} \triangleq \times_{ij}\mathcal{D}_{ij}$ is of *common knowledge* to both the seller and the players, optimal Bayesian incentive-compatible (BIC) mechanisms have been discovered for various auction settings [16, 11, 4, 5], where all players reporting their true values forms a Bayesian Nash equilibrium. When there is no common prior but the seller knows $\mathcal{D}$, many (approximately) optimal dominant-strategy incentive-compatible (DSIC) Bayesian mechanisms have been designed [16, 17, 7, 14, 19, 6], where it is each player's *dominant strategy* to report his true values.

However, the *complexity* for the seller to carry out such mechanisms is largely unconsidered in the literature. Most existing Bayesian mechanisms require that the seller has full access to the prior distribution $\mathcal{D}$ and is able to carry out all required optimizations based on $\mathcal{D}$, so as to compute the allocation and the prices. Unfortunately the seller may not be so knowledgeable or powerful in real-world scenarios. If the supports of the distributions are exponentially large (in m and n), or if the distributions are continuous and do not have succinct representations, it is hard for the seller to write out "each single bit" of the distributions or precisely carry out arbitrary optimization tasks based on them. Even with a single player and a single item, when the value distribution is irregular, computing the optimal price in time that is much smaller than the size of the support is not an easy task. Thus, a natural and important question to ask is *how much the seller should know about the distributions in order to obtain approximately optimal revenue.*

In this work we consider, for the first time, the *query complexity* of Bayesian mechanisms. In particular, the seller can only access the distributions by making oracle queries. Two types of queries are allowed, *quantile queries* and *value queries*. That is, the seller queries the oracle with specific quantiles (respectively, values), and the oracle returns the corresponding values (respectively, quantiles) in the underlying distributions. These two types of queries happen a lot in market study. Indeed, the seller may wish to know what is the price he should set so that half of the consumers would purchase his product; or if he sets the price to be 200 dollars, how many consumers would buy it. Another important scenario where such queries naturally come up is in databases. Indeed, although the seller may not know the distribution, some powerful institutes, say the Office for National Statistics, may have such information figured out and stored in its database. As in most database applications, it may be neither necessary nor feasible for the seller to download the whole distribution to his local machines. Rather, he would like to access the distribution via queries to the database. Other types of queries are of course possible, and will be considered in future works.

In this work we focus on *non-adaptive* queries. That is, the seller makes all oracle queries simultaneously, before the auction starts. This is also natural in both database and market study scenarios, and adaptive queries will be considered in future works.

2 Main Results

We would like to understand both lower- and upper-bounds for the query complexity of approximately optimal Bayesian auctions. In this work, we mainly consider three widely studied settings: single-item auctions and multi-item auctions with unit-demand or additive valuation functions. Our main results are summarized in Table 1.

Table 1 Our main results. Here $h(\cdot) < 1$ is the tail function in the small-tail assumptions. For single-item auctions, the revenue is a $(1+\epsilon)$-approximation to the optimal BIC revenue, with ϵ sufficiently small. For multi-item auctions with unit-demand or additive valuation functions, the revenue is a c-approximation for some constant c.

	Query Complexity	Distributions		
		Bounded in $[1, H]$		Unbounded & Small Tail
Auctions	Single-Item	$\Theta(n\epsilon^{-1}\log H)$		$O(-n\epsilon^{-1}\log h(\frac{2\epsilon}{3(1+\epsilon)}))$
	Unit-Demand	$\forall c > 1$: $\Omega(\frac{mn\log H}{\log c})$	$\forall c > 24$: $O(\frac{mn\log H}{\log(c/24)})$	$\forall c > 24$: $O(-\frac{mn\log h(\frac{2c-48}{3c})}{\log(c/24)})$
	Additive	$\forall c > 1$: $\Omega(\frac{mn\log H}{\log c})$	$\forall c > 8$: $O(\frac{mn\log H}{\log(c/8)})$	$\forall c > 8$: $O(-\frac{m^2 n\log h(\frac{c-8}{10c})}{\log(c/8)})$
	Single-Item	Regular Distributions: $\Omega(n\epsilon^{-1})$, $O(n\epsilon^{-1}\log\frac{n}{\epsilon})$		

Note that we allow arbitrary unbounded distributions that satisfy *small-tail assumptions*, which means the expected revenue generated from the "tail" of the distributions is negligible compared to the optimal revenue. Similar assumptions are widely adopted in sampling mechanisms [18, 12], to deal with irregular distributions with unbounded supports. Since distributions with bounded supports automatically satisfy the small-tail assumptions, the lower-bounds listed for the former apply to the latter as well.

Also note that our lower- and upper-bounds on query complexity are *tight* for bounded distributions. In the full version of the paper [9], we show that our lower-bounds allow the seller to make both value and quantile queries, and apply to any multi-player multi-item auctions where each player's valuation function is *succinct sub-additive*. The lower-bounds also allow randomized queries and randomized mechanisms.

For the upper-bounds, all our query schemes are deterministic and only make one type of queries: value queries for bounded distributions and quantile queries for unbounded ones. We show that our schemes, despite of being very efficient, only loses a small fraction of revenue compared with the cases where the seller has full access to the distributions.

3 Discussion and Future Directions

In the full version, we will elaborate on the connections between our work and related studies. For example, a closely related area is sampling mechanisms [10, 13, 15, 12, 3]. It assumes that the seller does not know $\mathcal{D}$ but observes independent samples from $\mathcal{D}$ before the auction begins. The *sample complexity* measures how many samples the seller needs so as to obtain a good approximation to the optimal Bayesian revenue. Our results show that query complexity can be exponentially smaller than sample complexity: the former is *logarithmic* in the "size" of the distributions, while the latter is known to be polynomial. We will also discuss other related studies such as [2, 1, 3, 8].

Finally, we point out some interesting further directions. As mentioned, we focus on non-adaptive queries in this work. One can imagine more powerful mechanisms using *adaptive* queries, where the seller's later queries depend on the oracle's responses to former ones. It is intriguing to design approximately optimal Bayesian mechanisms with lower query complexity using adaptive queries, or prove that even with such queries, the query complexity cannot be much better than our lower-bounds. Another interesting direction is when the answers of the oracle contain noise. In this case, the distributions learnt by the seller may be within a small distance from the "true distributions" defined by oracle answers without noise. It would be interesting to design mechanisms to handle such noise.

References

1. Pablo Azar, Constantinos Daskalakis, Silvio Micali, and S Matthew Weinberg. Optimal and efficient parametric auctions. In *24th Symposium on Discrete Algorithms (SODA'13)*, pages 596–604, 2013.
2. Pablo Azar and Silvio Micali. Parametric digital auctions. In *4rd Innovations in Theoretical Computer Science Conference (ITCS'13)*, pages 231–232, 2013.
3. Yang Cai and Constantinos Daskalakis. Learning multi-item auctions with (or without) samples. In *58th Symposium on Foundations of Computer Science (FOCS'17)*, pages 516–527, 2017.
4. Yang Cai, Constantinos Daskalakis, and S Matthew Weinberg. An algorithmic characterization of multi-dimensional mechanisms. In *44th Annual ACM Symposium on Theory of Computing (STOC'12)*, pages 459–478, 2012.
5. Yang Cai, Constantinos Daskalakis, and S Matthew Weinberg. Optimal multi-dimensional mechanism design: Reducing revenue to welfare maximization. In *53rd Symposium on Foundations of Computer Science (FOCS'12)*, pages 130–139, 2012.
6. Yang Cai, Nikhil R Devanur, and S Matthew Weinberg. A duality based unified approach to Bayesian mechanism design. In *48th Annual ACM Symposium on Theory of Computing (STOC'16)*, pages 926–939, 2016.
7. Shuchi Chawla, Jason D Hartline, David L Malec, and Balasubramanian Sivan. Multi-parameter mechanism design and sequential posted pricing. In *43th ACM Symposium on Theory of Computing (STOC'10)*, pages 311–320, 2010.
8. Jing Chen, Bo Li, and Yingkai Li. From Bayesian to crowdsourced Bayesian auctions. *arXiv:1702.01416*, 2016.
9. Jing Chen, Bo Li, Yingkai Li, and Pinyan Lu. Bayesian auctions with efficient queries, full version. *arXiv:1804.07451*, 2018.
10. Richard Cole and Tim Roughgarden. The sample complexity of revenue maximization. In *46th Annual ACM Symposium on Theory of Computing (STOC'14)*, pages 243–252, 2014.
11. Jacques Cremer and Richard P McLean. Full extraction of the surplus in Bayesian and dominant strategy auctions. *Econometrica*, 56(6):1247–1257, 1988.
12. Nikhil R. Devanur, Zhiyi Huang, and Christos-Alexandros Psomas. The sample complexity of auctions with side information. In *48th Annual ACM Symposium on Theory of Computing (STOC'16)*, pages 426–439, 2016.
13. Zhiyi Huang, Yishay Mansour, and Tim Roughgarden. Making the most of your samples. In *16th ACM Conference on Economics and Computation (EC'15)*, pages 45–60, 2015.
14. Robert Kleinberg and S Matthew Weinberg. Matroid prophet inequalities. In *44th Annual ACM Symposium on Theory of Computing (STOC'12)*, pages 123–136, 2012.
15. Jamie Morgenstern and Tim Roughgarden. Learning simple auctions. In *29th Conference on Learning Theory (COLT'16)*, pages 1298–1318, 2016.
16. Roger B Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1):58–73, 1981.
17. Amir Ronen. On approximating optimal auctions. In *3rd ACM Conference on Electronic Commerce (EC'01)*, pages 11–17, 2001.
18. Tim Roughgarden and Okke Schrijvers. Ironing in the dark. In *17th ACM Conference on Economics and Computation (EC'16)*, pages 1–18, 2016.
19. Andrew Chi-Chih Yao. An n-to-1 bidder reduction for multi-item auctions and its applications. In *26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*, pages 92–109, 2015.

Brief Announcement: Hamming Distance Completeness and Sparse Matrix Multiplication

Daniel Graf
Department of Computer Science, ETH Zürich, Switzerland
daniel.graf@inf.ethz.ch

Karim Labib
Department of Computer Science, ETH Zürich, Switzerland
labibk@student.ethz.ch

Przemysław Uznański
Department of Computer Science, ETH Zürich, Switzerland
przemyslaw.uznanski@inf.ethz.ch

Abstract

We show that a broad class of $(+, \diamond)$ vector products (for binary integer functions $\diamond$) are equivalent under one-to-polylog reductions to the computation of the Hamming distance. Examples include: the dominance product, the threshold product and ℓ_{2p+1} distances for constant p. Our results imply equivalence (up to $\operatorname{poly}\log n$ factors) between complexity of computation of All Pairs: Hamming Distances, ℓ_{2p+1} Distances, Dominance Products and Threshold Products. As a consequence, Yuster's (SODA'09) algorithm improves not only Matoušek's (IPL'91), but also the results of Indyk, Lewenstein, Lipsky and Porat (ICALP'04) and Min, Kao and Zhu (COCOON'09). Furthermore, our reductions apply to the pattern matching setting, showing equivalence (up to $\operatorname{poly}\log n$ factors) between pattern matching under Hamming Distance, ℓ_{2p+1} Distance, Dominance Product and Threshold Product, with current best upperbounds due to results of Abrahamson (SICOMP'87), Amir and Farach (Ann. Math. Artif. Intell.'91), Atallah and Duket (IPL'11), Clifford, Clifford and Iliopoulous (CPM'05) and Amir, Lipsky, Porat and Umanski (CPM'05). The resulting algorithms for ℓ_{2p+1} Pattern Matching and All Pairs ℓ_{2p+1}, for $2p+1 = 3, 5, 7, \ldots$ are new.

Additionally, we show that the complexity of AllPairsHammingDistances (and thus of other aforementioned AllPairs- problems) is within $\operatorname{poly}\log n$ from the time it takes to multiply matrices $n \times (n \cdot d)$ and $(n \cdot d) \times n$, each with $(n \cdot d)$ non-zero entries. This means that the current upperbounds by Yuster (SODA'09) cannot be improved without improving the sparse matrix multiplication algorithm by Yuster and Zwick (ACM TALG'05) and vice versa.

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

Keywords and phrases fine-grained complexity, matrix multiplication, high dimensional geometry, pattern matching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.109

1 Introduction

Many classical algorithmic problems received new attention when formulated as algebraic problems. In pattern matching we can define a similarity score between two strings and ask for this score between the pattern $\mathbf{P}$ of length m and every m-substring of the text $\mathbf{T}$ of length $n \geq m$. For example, scores of Hamming distance or L_1 distance between numerical strings generalize the classical pattern matching. All those problems share an additive structure, i.e. for an input pattern $\mathbf{P}$ and text $\mathbf{T}$, the score vector $\mathbf{O}$ is such that $\mathbf{O}[i] = \sum_j \mathbf{P}[j] \diamond \mathbf{T}[i+j]$

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 109; pp. 109:1–109:4


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Summary of different score functions and the corresponding problems. $\mathbf{1}[\varphi]$ is 1 iff φ and 0 otherwise.

Name	Score function	Pattern Matching problem	All Pairs problem
Hamming	$\mathbf{1}[x \neq y]$	$\mathbf{O}[i] = \lvert\{j : \mathbf{P}[j] \neq \mathbf{T}[i+j]\}\rvert$	$O[i][j] = \lvert\{k : \mathbf{A}_i[k] \neq \mathbf{B}_j[k]\}\rvert$
Dominance	$\mathbf{1}[x \leq y]$	$\mathbf{O}[i] = \lvert\{j : \mathbf{P}[j] \leq \mathbf{T}[i+j]\}\rvert$	$O[i][j] = \lvert\{k : \mathbf{A}_i[k] \leq \mathbf{B}_j[k]\}\rvert$
δ-Threshold	$\mathbf{1}[\lvert x-y\rvert \geq \delta]$	$\mathbf{O}[i] = \lvert\{j : \lvert\mathbf{P}[j] - \mathbf{T}[i+j]\rvert > \delta\}\rvert$	$O[i][j] = \lvert\{k : \lvert\mathbf{A}_i[k] - \mathbf{B}_j[k]\rvert > \delta\}\rvert$
ℓ_1 distance	$\lvert x-y\rvert$	$\mathbf{O}[i] = \sum_j \lvert\mathbf{P}[j] - \mathbf{T}[i+j]\rvert$	$O[i][j] = \sum_{k=1}^{n} \lvert\mathbf{A}_i[k] - \mathbf{B}_j[k]\rvert$

for some binary function $\diamond$. Just as those pattern matching generalizations are based on *convolution*, there is a family of problems based on *matrix multiplication*, varying in flavour according to the vector product used. There, we are given two matrices A and B and the output is the matrix $O[i][j] = \sum_k A[i][k] \diamond B[k][j]$. This is equivalent to the computation of all pairwise $(+, \diamond)$-vector products for two vector families, the so called AllPairs- problems. For a certain class of score functions, pattern matching generalizations admit independently algorithms of identical complexity $\mathcal{O}(n\sqrt{m \log m})$ (c.f. [1–3, 8]). For the same score functions, the best algorithms for corresponding AllPairs- problems are of complexity $\mathcal{O}(n^{(\omega+3)/2})$ or similar (c.f. [6, 8, 11]).

Our contribution:

We show that for a wide class of $(+, \diamond)$ products, the corresponding problems are of (almost) equivalent hardness. This class includes Hamming distance or Dominance, but also any piecewise polynomial function of two variables (for appropriate definition of piecewise polynomiality, c.f. Definition 2) excluding certain degenerate forms (e.g. polynomials). Thus we should not expect the problems based on $(+, \diamond)$ products to be significantly harder to compute than e.g. ones based on Hamming distance. The reduction applies both to Pattern Matching setting and to All Pairs- setting alike. We refer to Table 1 for a summary of considered problems and to Figure 1 for a summary of the old and new reductions. It implies that Yuster's [11] improvement to the exponent of AllPairsDominanceProducts applies to all other AllPairs- problems considered here. Additionally, any tradeoffs between vectors dimension and runtime (c.f. [5, 8]), or input sparsity and runtime (c.f. [4, 9, 10]) translates between problems. Additionally, we link the complexity of AllPairsHammingDistances (and thus to other AllPairs- problems) to one of a sparse rectangular matrix multiplication (c.f. Theorem 4): an instance of APHam can be expanded to an instance of sparse matrix multiplication of rectangular matrices, and any matrix multiplication instance with those parameters can be contracted back to APHam. It is interesting to observe that applying the fastest existing sparse matrix multiplication algorithm (c.f. [12]) to the resulting instance results in the same runtime as solving APHam directly.

2 Preliminaries

For vectors $\mathbf{A}, \mathbf{B}$ and matrices $\mathcal{A}, \mathcal{B}$, we denote the $(+, \diamond)$ vector product as $\textsc{vprod}(\diamond, \mathbf{A}, \mathbf{B}) \stackrel{\text{def}}{=} \sum_i \mathbf{A}[i] \diamond \mathbf{B}[i]$, the $(+, \diamond)$ convolution as $\textsc{conv}(\diamond, \mathbf{A}, \mathbf{B}) = \mathbf{C}$ where $\mathbf{C}[k] = \sum_{i+j=k} \mathbf{A}[i] \diamond \mathbf{B}[j]$ and the $(+, \diamond)$ matrix product as $\textsc{mprod}(\diamond, \mathcal{A}, \mathcal{B}) = \mathcal{C}$ where $\mathcal{C}[i, j] = \sum_k \mathcal{A}[i, k] \diamond \mathcal{B}[k, j]$.

Thus, e.g. defining $\mathsf{Ham}(x, y) \stackrel{\text{def}}{=} \mathbf{1}[x \neq y]$, then $\textsc{vprod}(\mathsf{Ham}, \cdot, \cdot)$, $\textsc{conv}(\mathsf{Ham}, \cdot, \cdot)$ and $\textsc{mprod}(\mathsf{Ham}, \cdot, \cdot)$ correspond to Hamming Distance between vectors, HamPM and APHam.

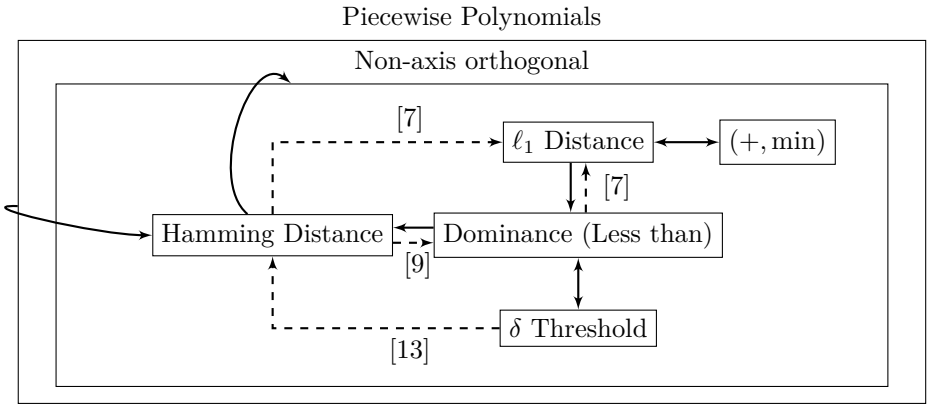


Figure 1 Existing and new reductions between problems, together with problem classes.

▶ **Definition 1.** We say that $\diamond$ reduces preserving linearity to instances of $\square_1, \ldots, \square_K$, if there are functions $f_1, \ldots, f_K$ and $g_1, \ldots, g_K$ and coefficients $\alpha_1, \ldots, \alpha_K$, such that for any x, y:[1] $x \diamond y = \sum_i \alpha_i \cdot \big(f_i(x) \mathbin{\square_i} g_i(y)\big)$.

Given Definition 1, we have for any vectors $\mathbf{A}, \mathbf{B}$ and matrices $\mathcal{A}, \mathcal{B}$: $\textsc{vprod}(\diamond, \mathbf{A}, \mathbf{B}) = \sum_i \alpha_i \cdot \textsc{vprod}(\square_i, f_i(\mathbf{A}), g_i(\mathbf{B}))$, $\textsc{conv}(\diamond, \mathbf{A}, \mathbf{B}) = \sum_i \alpha_i \cdot \textsc{conv}(\square_i, f_i(\mathbf{A}), g_i(\mathbf{B}))$ and $\textsc{mprod}(\diamond, \mathcal{A}, \mathcal{B}) = \sum_i \alpha_i \cdot \textsc{mprod}(\square_i, f_i(\mathcal{A}), g_i(\mathcal{B}))$, where $f(\mathbf{A})$ and $f(\mathcal{A})$ denotes a coordinate-wise application of f to vector $\mathbf{A}$ and matrix $\mathcal{A}$, respectively.

3 Main results

▶ Remark. We assume that all input values and coefficients are integers bounded in absolute value by $\mathrm{poly}(n)$.

▶ **Definition 2.** For integers A, B, C and polynomial $P(x, y)$ we say that the function $P(x, y) \cdot \mathbf{1}[Ax + By + C > 0]$ is *halfplane polynomial*. We call a sum of halfplane polynomial functions a *piecewise polynomial*. We say that a function is *axis-orthogonal piecewise polynomial*, if it is piecewise polynomial and for every i, $A_i = 0$ or $B_i = 0$.

Observe that $\mathsf{Ham}(x, y) = \mathbf{1}[x > y] + \mathbf{1}[x < y]$, $\max(x, y) = x \cdot \mathbf{1}[x \geq y] + y \cdot \mathbf{1}[x < y]$, $|x - y|^{2p+1} = (x - y)^{2p+1} \cdot \mathbf{1}[x > y] + (y - x)^{2p+1} \cdot \mathbf{1}[x < y]$, and $\mathsf{Thr}_\delta(x, y) \stackrel{\text{def}}{=} \mathbf{1}[|x - y| \geq \delta] = \mathbf{1}[x \leq y - \delta] + \mathbf{1}[x \geq y + \delta]$.

▶ **Theorem 3.** *Let $\diamond$ be a piecewise polynomial of constant degree and $\mathrm{poly}\log n$ number of summands.*

- *If $\diamond$ is axis orthogonal, then $\diamond$ is "easy": $(+, \diamond)$ convolution takes $\widetilde{O}(n)$ time, $(+, \diamond)$ matrix multiplication takes $\widetilde{O}(n^\omega)$ time.*
- *Otherwise, $\diamond$ is* Hamming distance complete*: under one-to-polylog reductions, $(+, \diamond)$ product is equivalent to Hamming distance, $(+, \diamond)$ convolution is equivalent to* HamPM *and $(+, \diamond)$ matrix multiplication is equivalent to* APHam.

[1] For the sake of simplicity, we are omitting in the definition the post-processing function necessary e.g. $(\,\cdot\,)^{1/p}$ for L_p norms.

▶ **Theorem 4.** *The time complexity of* APHam *on n vectors of dimension d is (under randomized Las Vegas reductions) within* $\operatorname{poly}\log n$ *from time it takes to multiply matrices $n \times (n \cdot d)$ and $(n \cdot d) \times n$, each with $(n \cdot d)$ non-zero entries.*

References

1. Amihood Amir, Ohad Lipsky, Ely Porat, and Julia Umanski. Approximate matching in the L_1 metric. In *CPM*, pages 91–103, 2005. doi:10.1007/11496656_9.
2. Mikhail J. Atallah and Timothy W. Duket. Pattern matching in the hamming distance with thresholds. *Inf. Process. Lett.*, 111(14):674–677, 2011. doi:10.1016/j.ipl.2011.04.004.
3. Peter Clifford, Raphaël Clifford, and Costas S. Iliopoulos. Faster algorithms for δ,γ-matching and related problems. In *CPM*, pages 68–78, 2005. doi:10.1007/11496656_7.
4. Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *SODA*, pages 384–391, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496813.
5. Omer Gold and Micha Sharir. Dominance product and high-dimensional closest pair under L_∞. In *ISAAC*, pages 39:1–39:12, 2017. doi:10.4230/LIPIcs.ISAAC.2017.39.
6. Piotr Indyk, Moshe Lewenstein, Ohad Lipsky, and Ely Porat. Closest pair problems in very high dimensions. In *ICALP*, pages 782–792, 2004. doi:10.1007/978-3-540-27836-8_66.
7. Ohad Lipsky and Ely Porat. L_1 pattern matching lower bound. *Inf. Process. Lett.*, 105(4):141–143, 2008. doi:10.1016/j.ipl.2007.08.011.
8. Kerui Min, Ming-Yang Kao, and Hong Zhu. The closest pair problem under the Hamming metric. In *COCOON*, pages 205–214, 2009. doi:10.1007/978-3-642-02882-3_21.
9. Virginia Vassilevska. *Efficient algorithms for path problems in weighted graphs*. PhD thesis, Carnegie Mellon University, 2008.
10. Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All pairs bottleneck paths and max-min matrix products in truly subcubic time. *Theory of Computing*, 5(1):173–189, 2009. doi:10.4086/toc.2009.v005a009.
11. Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In *SODA*, pages 950–957, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496873.
12. Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005. doi:10.1145/1077464.1077466.
13. Peng Zhang and Mikhail J. Atallah. On approximate pattern matching with thresholds. *Inf. Process. Lett.*, 123:21–26, 2017. doi:10.1016/j.ipl.2017.03.001.

Brief Announcement: Treewidth Modulator: Emergency Exit for DFVS

Daniel Lokshtanov
Department of Informatics, University of Bergen, Norway
daniello@uib.no

M. S. Ramanujan
Algorithms and Complexity Group, TU Wien, Austria
ramanujan@ac.tuwien.ac.at

Saket Saurabh
Institute of Mathematical Sciences, HBNI, India and UMI ReLax
saket@imsc.res.in

Roohani Sharma
Institute of Mathematical Sciences, HBNI, India and UMI ReLax
roohani@imsc.res.in

Meirav Zehavi
Department of Computer Science, Ben-Gurion University, Israel
meiravze@bgu.ac.il

Abstract

In the Directed Feedback Vertex Set (DFVS) problem, we are given as input a directed graph D and an integer k, and the objective is to check whether there exists a set S of at most k vertices such that $F = D - S$ is a directed acyclic graph (DAG). Determining whether DFVS admits a polynomial kernel (parameterized by the solution size) is one of the most important open problems in parameterized complexity. In this article, we give a polynomial kernel for DFVS parameterized by the solution size plus the size of any treewidth-η modulator, for any positive integer η. We also give a polynomial kernel for the problem, which we call Vertex Deletion to treewidth-η DAG, where given as input a directed graph D and a positive integer k, the objective is to decide whether there exists a set of at most k vertices, say S, such that $D - S$ is a DAG and the treewidth[1] of $D - S$ is at most η.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

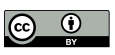
Keywords and phrases Polynomial Kernel, Directed Feedback Vertex Set, Treewidth Modulator

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.110

1 Introduction and Overview of Our Results

In the Directed Feedback Vertex Set (DFVS) problem, the input consists of a directed graph D on n vertices, and an integer k. The parameter is k, and the objective is to check whether there exists a set of at most k vertices, say S, such that $F = D - S$ is a directed acyclic graph (DAG). The question whether DFVS is fixed-parameter tractable was posed as an open problem in the first few papers on fixed-parameter tractability (FPT) [8, 9].

[1] Throughout the article, by treewidth of a directed graph we mean the treewidth of its underlying undirected graph.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 110; pp. 110:1–110:4

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This remained an open problem for over a decade until, in a breakthrough paper, DFVS was shown to be fixed-parameter tractable by Chen et al. [5] in 2008. Specifically, they gave an algorithm that runs in time $\mathcal{O}(4^k \cdot k! \cdot k^4 \cdot n^4)$. Following the resolution of the fixed-parameter tractability status of DFVS, one of the most natural follow-up questions in parameterized complexity, that has been raised several times, and has become one of the most fundamental questions, is "does DFVS admit a polynomial kernel?" A polynomial kernel is essentially a polynomial-time preprocessing algorithm that transforms the given instance of the problem into an equivalent one whose size is bounded polynomially in the specified parameter. Whenever the parameter is not specified, it is implied that the parameter is the solution size (in our case, the integer k in the input of the DFVS problem). In an attempt to develop an understanding on what makes this problem hard, and to move closer to answering this open question, several routes have been taken. These include:

1. enriching the parameterization to encompass not only solution size but also additional structural parameters,
2. restricting the input instances,
3. restricting the structure of the resulting DAG (F).

In this article, we give two results concerning DFVS that contribute to progress along all these three routes. We begin by first stating our results formally. For a directed graph D, a subset $M \subseteq V(D)$ is called a *treewidth η-modulator* if $D - M$ has treewidth at most η. For a fixed positive integer $\eta > 0$, let $\mathcal{F}_\eta$ be the family of digraphs of treewidth at most η. Formally, our first problem is the following.

DFVS/DFVS+Treewidth-η Modulator (DFVS/DFVS+Tw-η Mod) **Parameter:** $k + \ell$
Input: A digraph D, an integer k, $M \subseteq V(D)$ such that $|M| = \ell$ and $D - M \in \mathcal{F}_\eta$.
Question: Does there exist $S \subseteq V(D)$ such that $|S| \leq k$ and $D - S$ is a DAG?

Our first result is the following.

▶ **Theorem 1.** DFVS/DFVS+Tw-η Mod *admits a polynomial kernel of size* $(k \cdot \ell)^{\mathcal{O}(\eta^2)}$.

Our second problem is the following.

Vertex Deletion to treewidth-η DAG **Parameter:** k
Input: A digraph D, an integer k.
Question: Does there exist $S \subseteq V(D)$ such that $|S| \leq k$ and $D - S$ is a DAG and $D - S \in \mathcal{F}_\eta$?

The next theorem states our second result.

▶ **Theorem 2.** *For any fixed positive integer η,* Vertex Deletion to treewidth-η DAG *has polynomial kernel.*

Let us now see how both our results make progress along all the three routes described above. Along the first route, Bergougnoux et al. [4] studied DFVS parameterized by the feedback vertex set (**fvs**) number of the underlying undirected graph, and gave a polynomial kernel for this problem. Our first result gives a polynomial kernel for DFVS when the parameter is solution size (k) plus the size of any treewidth-η modulator in D (say ℓ), for any fixed positive integer η. Note that the parameter $k + \ell$ is not only upper bounded by $\mathcal{O}(\mathbf{fvs})$, where **fvs** is the feedback vertex set number of the underlying undirected graph of D, but it can be arbitrarily smaller than **fvs**. Thus, studying such a parameter brings us closer to

the problem of the existence of a polynomial kernel for DFVS. (Note that the question of the existence of a polynomial kernel for DFVS parameterized by the size of a treewidth-η modulator, for $\eta \geq 2$, alone has a negative answer because Vertex Cover parameterized by the size of any treewidth-2 modulator cannot have a polynomial kernel, unless $\mathsf{NP} \subseteq \frac{\text{co-NP}}{\text{poly}}$ [6].) Moreover, the ideas harnessed during the construction of our polynomial kernel utilizes the tool of important separators in a novel fashion. To the best of our knowledge, this is the first time that the power of important separators has been harnessed to develop a polynomial kernel. Furthermore, to derive this result, we need to embed this tool in state-of-the-art machinery such as the use of protrusion replacers where the replacement is a minor of the part of the graph that is replaced.

Along the second route (that is, studying DFVS by restricting the input instance), there have been several results for polynomial kernels for DFVS when the input graph is a tournament or some generalization of it (like a bipartite tournament etc.) [1, 3, 7, 10]. However, the existence of a polynomial kernel for DFVS is open even when the input digraph is a planar digraph. From our first result (Theorem 1), we can conclude that we have a polynomial kernel for DFVS when the treewidth of the input graph is polynomial in the solution size ($k^{\mathcal{O}(1)}$).

Along the third route, Mnich and van Leeuwen [11] studied the problem, where they considered DFVS with an additional restriction on the output DAG rather than the input instance. They inspected this question by considering k vertex deletion to the classes of out-forests, out-trees and (directed) pumpkins. They obtained polynomial kernels for all these problems. Observe that for all these classes, the treewidth of the graphs in these classes is constant (at most 2). In a follow-up paper [2], the kernel sizes given by Mnich and van Leeuwen [11] were reduced. Our second result generalizes this approach by demanding that the resulting DAG has bounded treewidth (bounded by any fixed constant η).

Our Methods. We now give a very brief overview of the methods used to prove our results. In fact, here, we only focus on our first result.

Proof Idea of Theorem 1. Our kernelization algorithm can be divided into three main phases. Recall that the input is a directed graph D, an integer k and a treewidth-η modulator M of size ℓ. In the first phase, we decompose the graph into $\mathcal{O}(k\ell^2)$ parts (which we call *zones*), each of which have constant treewidth and a "controlled" neighbourhood in the rest of the graph. In the next step, we mark $(k\ell)^{\mathcal{O}(\eta^2)}$ vertices inside each zone, which have the property that in the case of a YES-instance, there is also a solution that does not use any of the unmarked vertices in these zones. Getting such a set of marked vertices of polynomial size is the core of our algorithm. Having such a set of marked vertices at our disposal, we then design reduction rules that partition the unmarked vertices in each zone into disjoint protrusions that are then replaced by constant size graphs, such that the modulator M remains a treewidth-η modulator in the resulting graph too. Note that this is done (over a standard protrusion replacement) to ensure that our parameter does not increase in the resulting instance. If one does not care about the parameter increase in the resulting instance, then some of the steps in this algorithm can be simplified.

References

1 F N Abu-Khzam. A kernelization algorithm for d-Hitting Set. *JCSS*, 76(7):524–531, 2010.

2 Akanksha Agrawal, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Kernels for deletion to classes of acyclic digraphs. *J. Comput. Syst. Sci.*, 92:9–21, 2018. doi:10.1016/j.jcss.2017.07.008.

3 J Bang-Jensen, A Maddaloni, and S Saurabh. Algorithms and kernels for feedback set problems in generalizations of tournaments. *Algorithmica*, pages 1–24, 2015.

4 Benjamin Bergougnoux, Eduard Eiben, Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Towards a polynomial kernel for directed feedback vertex set. In *MFCS 2017*, 2017. To Appear.

5 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5), 2008. doi:10.1145/1411509.1411511.

6 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. On the hardness of losing width. *Theory Comput. Syst.*, 54(1):73–82, 2014. doi:10.1007/s00224-013-9480-1.

7 M Dom, J Guo, F Hüffner, R Niedermeier, and A Truß. Fixed-parameter tractability results for feedback set problems in tournaments. *JDA*, 8(1):76–86, 2010. doi:10.1016/j.jda.2009.08.001.

8 Rodney G. Downey and Michael R. Fellows. Fixed-parameter intractability. In *Proceedings of the Seventh Annual Structure in Complexity Theory Conference, Boston, Massachusetts, USA, June 22-25, 1992*, pages 36–49, 1992.

9 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4):873–921, 1995.

10 Tien-Nam Le, Daniel Lokshtanov, Saket Saurabh, Stéphan Thomassé, and Meirav Zehavi. Subquadratic kernels for implicit 3-hitting set and 3-set packing problems. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 331–342, 2018.

11 Matthias Mnich and Erik Jan van Leeuwen. Polynomial kernels for deletion to classes of acyclic digraphs. *Discrete Optimization*, 25:48–76, 2017. doi:10.1016/j.disopt.2017.02.002.

Brief Announcement: Erasure-Resilience Versus Tolerance to Errors

Sofya Raskhodnikova
Boston University, Boston, USA
sofya@bu.edu

Nithin Varma
Boston University, Boston, USA
nvarma@bu.edu

Abstract

We describe work in progress on providing a separation between erasure-resilient and tolerant property testing. Specifically, we are able to exhibit a property which is testable (with the number of queries independent of the length of the input) in the presence of erasures, but is not testable tolerantly.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Property testing, erasures, tolerance to errors, model separation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.111

Funding This material is based upon work supported by the National Science Foundation under Grant No. CCF-142297.

1 Description

In this brief announcement, we describe our recent investigation of the effects of adversarial corruption to inputs on the complexity of sublinear-time algorithms. Input corruption occurs either in the form of errors (when some values get changed) or in the form of erasures (when some values go missing). Understanding the relative difficulty of designing algorithms that work in the presence of different forms of corruption is a problem of fundamental importance. It is with this motivation in mind that property testing [5, 7], one of the most widely studied models of sublinear-time algorithms, was generalized to erasure-resilient testing [3] and (error) tolerant testing [6].

Erasure-resilient property testing falls between (standard) property testing and tolerant testing. Specifically, an erasure-resilient tester for a property, in the special case when no erasures occur, is a standard tester for this property. Also, a tolerant tester for a property implies the existence of an erasure-resilient tester with comparable parameters for the same property. Dixit, Raskhodnikova, Thakurta and Varma [3] separate standard and erasure-resilient testing by describing a property that is *easy* to test in the standard model and *hard* to test in the erasure-resilient model. Their separation is based on an earlier result by Fischer and Fortnow [4] that separates standard property testing from tolerant property testing in the same sense. Their main tool is PCPs of proximity (also known as assignment testers) defined by Ben-Sasson, Goldreich, Harsha, Sudan and Vadhan [1] and by Dinur and Reingold [2]. Dixit et al. [3] asked whether it is possible to obtain a separation between erasure-resilient and tolerant testing. Here, we announce such a separation. Specifically,

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 111; pp. 111:1–111:3


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

we are able to describe a property testable in the erasure-resilient model with the query complexity independent of the input size, but for which the query complexity of tolerant testing grows with the input size.

1.1 Erasure-Resilient and Tolerant Testing: Definitions

We now describe the erasure-resilient and tolerant models of testing. A *property* $\mathcal{P}$ is a set of strings. A string is *α-erased* for $\alpha \in [0,1)$ if at most an α fraction of its values are erasures (denoted by $\perp$). A *completion* of an α-erased string $x \in \{0,1,\perp\}^n$ is a string $y \in \{0,1\}^n$ that agrees with x on all the positions where x is nonerased. An *α-erasure-resilient ε-tester* [3] for a property $\mathcal{P}$ is a randomized algorithm that, given parameters $\alpha \in [0,1), \varepsilon \in (0,1)$ and oracle access to an α-erased string x, accepts with probability at least $2/3$ if x has a completion in $\mathcal{P}$ and rejects with probability at least $2/3$ if, in every completion of x, at least an ε fraction of the **nonerased** positions has to be changed to get a string in $\mathcal{P}$. The property $\mathcal{P}$ is *α-erasure-resiliently ε-testable* if there exists an α-erasure-resilient ε-tester for $\mathcal{P}$ with query complexity that depends only on the parameters α and ε (but not on the length of the input string).

A string $x \in \{0,1\}^n$ is ε'-far (α-close) from (to, respectively) a property $\mathcal{P}$, if the normalized Hamming distance of x from $\mathcal{P}$ is at least ε' (at most α, respectively). An *(α,ε')-tolerant tester* [6] for $\mathcal{P}$ is a randomized algorithm that, given parameters $\alpha \in (0,1), \varepsilon' \in (\alpha,1)$ and oracle access to a string x, accepts with probability at least $\frac{2}{3}$, if x is α-close to $\mathcal{P}$ and rejects with probability at least $\frac{2}{3}$, if x is ε'-far from $\mathcal{P}$. The property $\mathcal{P}$ is *(α,ε')-tolerantly testable* if there exists an (α,ε')-tolerant tester for $\mathcal{P}$ with query complexity that depends only on the parameters α and ε' (but not on the length of the input string).

1.2 Comparison of parameters

We remark that, while comparing the above two models, it is appropriate to compare $(\alpha, \alpha+\varepsilon(1-\alpha))$-tolerant testing of a property $\mathcal{P}$ with α-erasure-resilient ε-testing of $\mathcal{P}$ for the same values of α and ε. The parameter α in both the models is an upper bound on the fraction of corruptions (erasures, or errors) that an adversary can make to an input. An α-erasure-resilient ε-tester rejects with probability at least $\frac{2}{3}$ if, for for every way of completing an input string, one needs to change at least an ε fraction of the remaining part of the input to make it satisfy $\mathcal{P}$. Similarly, an $(\alpha, \alpha+\varepsilon(1-\alpha))$-tolerant tester rejects with probability at least $\frac{2}{3}$ if, for every way of *correcting* α fraction of the input values, one needs to change at least an ε fraction of the remaining $(1-\alpha)$ fraction of the input to make it satisfy $\mathcal{P}$.

1.3 Our Results

Our main contribution is the following theorem which states that there exists a property that is erasure-resiliently testable and is not tolerantly testable. This proves that tolerant testing is, in general, a harder problem than erasure-resilient testing.

▶ **Theorem 1** (Main Theorem). *There exists a property $\mathcal{P}$ and constants $\varepsilon, \alpha \in (0,1)$ such that*

- *$\mathcal{P}$ is α-erasure-resiliently ε-testable;*
- *$\mathcal{P}$ is not $(\alpha, \alpha+\varepsilon(1-\alpha))$-tolerantly testable.*

2 Conclusions

To summarize, we solve an open question proposed by Dixit et al. [3] and prove that tolerant testing is harder than erasure-resilient testing.

References

1 Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006.

2 Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. *SIAM J. Comput.*, 36(4):975–1024, 2006.

3 Kashyap Dixit, Sofya Raskhodnikova, Abhradeep Thakurta, and Nithin Varma. Erasure-resilient property testing. *SIAM Journal on Computing*, 47(2):295–329, 2018.

4 Eldar Fischer and Lance Fortnow. Tolerant versus intolerant testing for boolean properties. *Theory of Computing*, 2(9):173–183, 2006.

5 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

6 Michal Parnas, Dana Ron, and Ronitt Rubinfeld. Tolerant property testing and distance approximation. *J. Comput. Syst. Sci.*, 72(6):1012–1042, 2006.

7 Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

Brief Announcement: Bounded-Degree Cut is Fixed-Parameter Tractable

Mingyu Xiao[1]
School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China
myxiao@gmail.com
https://orcid.org/0000-0002-1012-2373

Hiroshi Nagamochi
Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Japan
nag@amp.i.kyoto-u.ac.jp

Abstract

In the bounded-degree cut problem, we are given a multigraph $G = (V, E)$, two disjoint vertex subsets $A, B \subseteq V$, two functions $\mathrm{u}_A, \mathrm{u}_B : V \to \{0, 1, \ldots, |E|\}$ on V, and an integer $k \geq 0$. The task is to determine whether there is a minimal (A, B)-cut (V_A, V_B) of size at most k such that the degree of each vertex $v \in V_A$ in the induced subgraph $G[V_A]$ is at most $\mathrm{u}_A(v)$ and the degree of each vertex $v \in V_B$ in the induced subgraph $G[V_B]$ is at most $\mathrm{u}_B(v)$. In this paper, we show that the bounded-degree cut problem is fixed-parameter tractable by giving a $2^{18k}|G|^{O(1)}$-time algorithm. This is the first single exponential FPT algorithm for this problem. The core of the algorithm lies two new lemmas based on important cuts, which give some upper bounds on the number of candidates for vertex subsets in one part of a minimal cut satisfying some properties. These lemmas can be used to design fixed-parameter tractable algorithms for more related problems.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability, Mathematics of computing → Graph algorithms

Keywords and phrases FPT, Important Cuts, Graph Cuts, Graph Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.112

1 Introduction

A *cut* of a graph is a partition of the vertices of the graph into two disjoint subsets. Graph cuts play an important role in combinatorial optimization and graph theory. The classical *minimum cut problem* is well known to be polynomially solvable [13]. Due to the rich application realm of this problem, many variants and extensions have been investigated. Some problems ask to partition the graph into more than two parts to disconnect some vertices such as the *k-way cut problem* (the *k-cut problem*) [14, 15], the *multiterminal cut problem* [9, 21] and the *multicut problem* [5, 19]. Some problems are still going to partition the graph into two parts, but with some additional requirements beyond the disconnectivity. One of the most extensively studied additional requirements is the constraint on the numbers of vertices or edges in each of the two parts. For examples, the *balanced cut problem* [1, 12, 16] and the *minimum bisection problem* [7, 10, 11] require the numbers of vertices in the two parts of the cut as close as possible. The (*balanced*) *judicious bipartition problem* [17] has

[1] Supported by the National Natural Science Foundation of China, under grants 61772115 and 61370071.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 112; pp. 112:1–112:6

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

conditions on the numbers of edges in the two parts. Some other well studied additional requirements include conditions on the connectivity of the two parts such as the *2-disjoint connected subgraphs problem* [8], and conditions on the degree of the two parts, such as the series of bipartition problems with degree constraints [2, 3, 4, 20, 22].

In this paper, we study the *bounded-degree cut* problem, which belongs to the latter kind of the extensions: to partition a given graph into two parts with some degree constraints on the induced subgraphs of the two parts. We mainly consider the upper bounds of the degree. An (S,T)-cut (V_1,V_2) is minimal if $E_G(V_1)$ does not contain $E_G(V_1')$ or $E_G(V_2')$ as a subset for any $S \subseteq V_1' \subsetneqq V_1$ and $T \subseteq V_2' \subsetneqq V_2$. Our problem is defined as follows.

BOUNDED-DEGREE CUT (with parameter: k)
Instance: A multigraph $G=(V,E)$, two disjoint nonempty vertex subsets $A,B \subseteq V$, two functions u_A and u_B from V to $\{0,1,\ldots,|E|\}$ and an integer $k \geq 0$.
Question: Does there exist a minimal (A,B)-cut (V_A,V_B) such that
the number of edges with one endpoint in V_A and one endpoint in V_B is at most k,
for each vertex $v \in V_A$, the degree of it in the induced graph $G[V_A]$ is at most $\mathrm{u}_A(v)$, and
for each vertex $v \in V_B$, the degree of it in the induced graph $G[V_B]$ is at most $\mathrm{u}_B(v)$?

During the last decade, cut related problems were extensively studied from the viewpoint of parameterized algorithms [12, 18, 7, 17, 15, 19, 21, 6]. In this paper, we will study BOUNDED-DEGREE CUT from the viewpoint of parameterized algorithms. Our main result is the first single-exponential FPT algorithm for BOUNDED-DEGREE CUT, which implies that BOUNDED-DEGREE CUT can be solved in polynomial time for $k = O(\log |G|)$.

▶ **Theorem 1.** BOUNDED-DEGREE CUT *can be solved in* $2^{18k} \cdot |G|^{O(1)}$ *time.*

2 The main idea

The most crucial techniques in this paper are: to use important cuts introduced by Marx [18] to obtain the following two general lemmas for bounded sets related to cuts; and then based on these two lemmas, to construct from a given instance a set of at most 2^{18k} new "easy" instances such that the original instance is feasible if and only if at least one of the "easy" instances is feasible.

▶ **Lemma 2.** *Let $A,B,C \subseteq V$ be non-empty subsets in a graph $G=(V,E)$ and k and ℓ be nonnegative integers. Then one can find in $2^{3(k+\ell)}(n+m)^{O(1)}$ time a family $\mathcal{X}$ of at most $2^{3(k+\ell)}$ subsets of C with a property that $C \cap V_1 \in \mathcal{X}$ for any minimal (A,B)-cut (V_1,V_2) with size at most k such that $|C \cap V_1| \leq \ell$.*

▶ **Lemma 3.** *Let $A,B,B' \subseteq V$ be non-empty subsets in a graph $G=(V,E)$, where $B' \subseteq B$, and k be a nonnegative integer. Then one can find in $2^{3k}(n+m)^{O(1)}$ time a family $\mathcal{Y}$ of at most 2^{3k} subsets of $N_G(B')$ with a property that $N_G(B') \cap V_1 \in \mathcal{Y}$ for any minimal (A,B)-cut (V_1,V_2) with size at most k.*

We will use $I=(G=(V,E),A,B)$ to denote an instance of the problem, where u_A, u_B and k are omitted since they remain unchanged throughout our argument. We use Z_A and Z_B to denote the sets of *A-unsatisfied vertices* and *B-unsatisfied vertices*, respectively, i.e., $Z_A \triangleq \{v \in V \mid \deg_G(v) > \mathrm{u}_A(v)\}$ and $Z_B \triangleq \{v \in V \mid \deg_G(v) > \mathrm{u}_B(v)\}$. We call I an *easy instance* if it holds that $Z_A \cup Z_B \subseteq A \cup B$, $N_G(Z_A \cap A) \subseteq A \cup B$, and $N_G(Z_B \cap B) \subseteq A \cup B$. We can see that an easy instance can be solved in polynomial time.

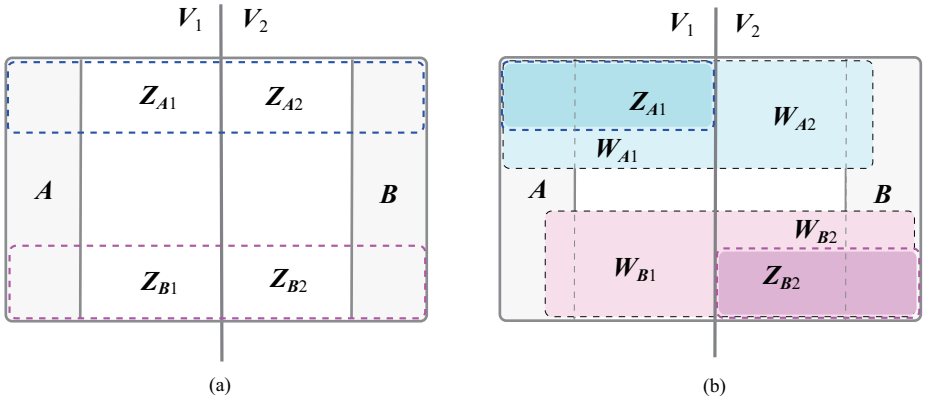


Figure 1 (a) An (A, B)-cut $\pi = (V_A, V_B)$ to I and the partitions $\{Z_{A1}, Z_{A2}\}$ of Z_A and $\{Z_{B1}, Z_{B2}\}$ of Z_B by π, where possibly $Z_A \cap Z_B \neq \emptyset$, (b) The partitions $\{W_{B1}, W_{B2}\}$ of $W_B = N_G(Z_{B2})$ and $\{W_{A1}, W_{A2}\}$ of $W_A = N_G(Z_{A1})$ by π, where possibly $W_A \cap W_B \neq \emptyset$.

For a given instance I with a feasible (A, B)-cut (V_1, V_2), we try to guess some subsets $V_1' \subseteq V_1 \setminus A$ and $V_2' \subseteq V_2 \setminus B$ so that the new instance $(G, A^* = A \cup V_1', B^* = B \cup V_2', \mathrm{u}_A, \mathrm{u}_B, k)$ remains feasible and is an easy. We will generate at most 2^{18k} easy instances.

3 Constructing Easy Instances

For a minimal (A, B)-cut $\pi = (V_1, V_2)$ (not necessarily feasible) in a given instance $I = (G = (V, E), A, B)$, we define the following notation on vertex subsets:
$Z_{Ai} \triangleq Z_A \cap V_i$ and $Z_{Bi} \triangleq Z_B \cap V_i$, $i = 1, 2$;
$W_A \triangleq N_G(Z_{A1})$ and $W_B \triangleq N_G(Z_{B2})$; $W_{Ai} \triangleq W_A \cap V_i$, and $W_{Bi} \triangleq W_B \cap V_i$, $i = 1, 2$;
$A_\pi \triangleq A \cup Z_{A1} \cup Z_{B1} \cup W_{A1} \cup W_{B1}$ and $B_\pi \triangleq B \cup Z_{A2} \cup Z_{B2} \cup W_{A2} \cup W_{B2}$.
See in Fig. 1 for an illustration on these subsets. Observe that the resulting instance (G, A_π, B_π) is an easy instance. The (A, B)-cut $\pi = (V_1, V_2)$ is feasible if and only if the corresponding instance (G, A_π, B_π) is feasible.

3.1 Partitioning Unsatisfied Vertices

For a minimal (A, B)-cut (V_1, V_2) to an instance I, let Z_{A1} and Z_{B2} be the subsets defined in the above. We observe that if the cut is feasible, then

$$|Z_{A1}|, |Z_{B2}| \leq k$$

since each vertex in $Z_{A1} \cup Z_{B2}$ has at least one incident edge included in $E_G(V_1, V_2)$ so that the degree constraint on the vertex holds.

By applying Lemma 2 to $(A, B, C = Z_A, k, \ell = k)$, we can construct in $2^{6k}(n+m)^{O(1)}$ time a family $\mathcal{X}_1$ of at most 2^{6k} subsets of Z_A such that $\mathcal{X}_1$ contains the set Z_{A1} defined to each feasible (A, B)-cut (V_1, V_2) in the instance $I = (G, A, B)$. Symmetrically it takes $2^{6k}(n+m)^{O(1)}$ time to find a family $\mathcal{X}_2$ of at most 2^{6k} subsets of Z_B such that $\mathcal{X}_2$ contains the set Z_{B2} defined to each feasible (A, B)-cut (V_1, V_2) in the instance $I = (G, A, B)$. Then the set $\mathcal{X}_{1,2}$ of all pairs (X_1, X_2) of disjoint sets $X_i \in \mathcal{X}_i$, $i = 1, 2$ contains the pair (Z_{A1}, Z_{B2}) defined to each feasible (A, B)-cut (V_1, V_2) in I. By noting that $|\mathcal{X}_{1,2}| \leq 2^{6k} 2^{6k} = 2^{12k}$, we obtain the next.

▶ **Lemma 4.** *Given an instance $I = (G, A, B)$, one can construct in $2^{12k}(n+m)^{O(1)}$ time at most 2^{12k} new instances $I' = (G, A', B')$ with $Z_A \cup Z_B \subseteq A' \cup B'$, one of which is equal to $(G, A \cup Z_{A1} \cup Z_{B1}, B \cup Z_{A2} \cup Z_{B2})$ for each feasible (A, B)-cut (V_1, V_2) to I.*

3.2 Partitioning Neighbors of Unsatisfied Vertices

For a minimal (A, B)-cut (V_1, V_2) to an instance I, let W_{A2} and W_{B1} be the subsets defined in the above. We observe that if the cut is feasible, then

$$|W_{B1}|, |W_{A2}| \leq k$$

since each of $|N_G(Z_{B2}) \cap V_1|$ and $|N_G(Z_{A1}) \cap V_2|$ is at most $|E_G(V_1, V_2)| \leq k$ to the feasible (A, B)-cut (V_1, V_2).

By applying Lemma 3 to $(A \cup Z_{A1} \cup Z_{B1}, B \cup Z_{A2} \cup Z_{B2}, B' = Z_{B2}, k)$, we can construct in $2^{3k}n^{O(1)}$ time a family $\mathcal{Y}_1$ of at most 2^{3k} subsets of $N_G(Z_{B2})$ such that $\mathcal{Y}_1$ contains the set $W_{B1} = N_G(Z_{B2}) \cap V_1$ defined to each feasible (A, B)-cut (V_1, V_2) in the instance $I = (G, A, B)$. Symmetrically it takes $2^{3k}(n+m)^{O(1)}$ time to find a family $\mathcal{Y}_2$ of at most 2^{3k} subsets of $N_G(Z_{A1})$ such that $\mathcal{Y}_2$ contains the set $W_{A2} = N_G(Z_{A1}) \cap V_2$ defined to each feasible (A, B)-cut (V_1, V_2) in I. Then the set $\mathcal{Y}_{1,2}$ of all pairs (Y_1, Y_2) of disjoint sets $Y_i \in \mathcal{Y}_i$, $i = 1, 2$ contains the pair (W_{B1}, W_{A2}) defined to each feasible (A, B)-cut (V_1, V_2) in the instance $I = (G, A, B)$. By noting that $|\mathcal{Y}_{1,2}| \leq 2^{6k}$, we obtain the next.

▶ **Lemma 5.** *Given an instance $I = (G, A, B)$ and the subsets Z_{A1} and Z_{B2} defined to a feasible (A, B)-cut (V_1, V_2) in I, one can construct in $2^{6k}(n+m)^{O(1)}$ time at most 2^{6k} new easy instances $I' = (G, A', B')$, one of which is equal to (G, A_π, B_π) defined to the feasible (A, B)-cut $\pi = (V_1, V_2)$.*

By Lemmas 4 and 5, we obtain the next, which can imply Theorem 1.

▶ **Lemma 6.** *Given an instance $I = (G, A, B)$, one can construct in $2^{18k}(n+m)^{O(1)}$ time at most 2^{18k} new easy instances $I' = (G, A', B')$, one of which is equal to (G, A_π, B_π) for each feasible (A, B)-cut $\pi = (V_1, V_2)$ to I.*

References

1. Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 222–231. ACM, 2004. doi:10.1145/1007352.1007355.
2. Jørgen Bang-Jensen and Stéphane Bessy. Degree-constrained 2-partitions of graphs. *CoRR*, abs/1801.06216, 2018. arXiv:1801.06216.
3. Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. Degree-constrained decompositions of graphs: Bounded treewidth and planarity. *Theor. Comput. Sci.*, 355(3):389–395, 2006. doi:10.1016/j.tcs.2006.01.024.
4. Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. Efficient algorithms for decomposing graphs under degree constraints. *Discrete Applied Mathematics*, 155(8):979–988, 2007. doi:10.1016/j.dam.2006.10.005.
5. Gruia Călinescu, Cristina G. Fernandes, and Bruce A. Reed. Multicuts in unweighted graphs and digraphs with bounded degree and bounded tree-width. *J. Algorithms*, 48(2):333–359, 2003. doi:10.1016/S0196-6774(03)00073-7.
6. Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009. doi:10.1007/s00453-007-9130-6.

7 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Minimum bisection is fixed parameter tractable. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 323–332. ACM, 2014. URL: http://dl.acm.org/citation.cfm?id=2591796, doi:10.1145/2591796.2591852.

8 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Solving the 2-disjoint connected subgraphs problem faster than 2 n. *Algorithmica*, 70(2):195–207, 2014. doi:10.1007/s00453-013-9796-x.

9 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994. doi:10.1137/S0097539792225297.

10 Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM J. Comput.*, 31(4):1090–1118, 2002. doi:10.1137/S0097539701387660.

11 Uriel Feige, Robert Krauthgamer, and Kobbi Nissim. Approximating the minimum bisection size (extended abstract). In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 530–536. ACM, 2000. doi:10.1145/335305.335370.

12 Uriel Feige and Mohammad Mahdian. Finding small balanced separators. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 375–384. ACM, 2006. doi:10.1145/1132516.1132573.

13 L.R. Ford and Delbert R. Fulkerson. *Flows in networks*. Princeton U. Press, Princeton, NJ, 1962.

14 Olivier Goldschmidt and Dorit S. Hochbaum. A polynomial algorithm for the k-cut problem for fixed k. *Math. Oper. Res.*, 19(1):24–37, 1994. doi:10.1287/moor.19.1.24.

15 Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 160–169. IEEE Computer Society, 2011. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6108120, doi:10.1109/FOCS.2011.53.

16 Frank Thomson Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999. doi:10.1145/331524.331526.

17 Daniel Lokshtanov, Saket Saurabh, Roohani Sharma, and Meirav Zehavi. Balanced judicious bipartition is fixed-parameter tractable. In Satya V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, volume 93 of *LIPIcs*, pages 40:40–40:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. URL: http://www.dagstuhl.de/dagpub/978-3-95977-055-2, doi:10.4230/LIPIcs.FSTTCS.2017.40.

18 Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006. doi:10.1016/j.tcs.2005.10.007.

19 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 469–478. ACM, 2011. doi:10.1145/1993636.1993699.

20 Michael Stiebitz. Decomposing graphs under degree constraints. *Journal of Graph Theory*, 23(3):321–324, 1996. doi:10.1002/(SICI)1097-0118(199611)23:3<321::AID-JGT12>3.0.CO;2-H.

21 Mingyu Xiao. Simple and improved parameterized algorithms for multiterminal cuts. *Theory Comput. Syst.*, 46(4):723–736, 2010. doi:10.1007/s00224-009-9215-5.

22 Mingyu Xiao and Hiroshi Nagamochi. Complexity and kernels for bipartition into degree-bounded induced graphs. *Theor. Comput. Sci.*, 659:72–82, 2017. doi:10.1016/j.tcs.2016.11.011.

Almost Sure Productivity

Alejandro Aguirre
IMDEA Software Institute, Madrid, Spain

Gilles Barthe
IMDEA Software Institute, Madrid, Spain

Justin Hsu
University College London, London, UK

Alexandra Silva
University College London, London, UK

Abstract

We introduce *Almost Sure Productivity (ASP)*, a probabilistic generalization of the productivity condition for coinductively defined structures. Intuitively, a probabilistic coinductive stream or tree is ASP if it produces infinitely many outputs with probability 1. Formally, we define ASP using a final coalgebra semantics of programs inspired by Kerstan and König. Then, we introduce a core language for probabilistic streams and trees, and provide two approaches to verify ASP: a syntactic sufficient criterion, and a decision procedure by reduction to model-checking LTL formulas on probabilistic pushdown automata.

2012 ACM Subject Classification Theory of computation → Denotational semantics, Theory of computation → Probabilistic computation, Theory of computation → Program reasoning

Keywords and phrases Coinduction, Probabilistic Programming, Productivity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.113

Related Version The full version of this paper can be found at `https://arxiv.org/abs/1802.06283`.

Acknowledgements We thank Benjamin Kaminski, Charles Grellois and Ugo dal Lago for helpful discussions on preliminary versions of this work, and we are grateful to the anonymous reviewers for pointing out areas where the exposition could be improved. This work was initiated at the Bellairs Research Institute in Barbados, and was partially supported by NSF grant #1637532 and ERC grant ProFoundNet (#679127).

1 Introduction

The study of probabilistic programs has a long history, especially in connection with semantics [22] and verification [23, 17, 28]. Over the last decade the field of probabilistic programming has attracted renewed attention with the emergence of practical probabilistic programming languages and novel applications in machine learning, privacy-preserving data mining, and modeling of complex systems. On the more theoretical side, many semantical and syntactic tools have been developed for verifying probabilistic properties. For instance, significant attention has been devoted to termination of probabilistic programs, focusing on the complexity of the different termination classes [19], and on practical methods for proving that a program terminates [16, 25, 2, 27]. The latter class of works generally focuses on *almost sure termination*, which guarantees that a program terminates with probability 1.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 113; pp. 113:1–113:15


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Coinductive probabilistic programming is a new computational paradigm that extends probabilistic programming to infinite objects, such as streams and infinite trees, providing a natural setting for programming and reasoning about probabilistic infinite processes such as Markov chains or Markov decision processes. Rather surprisingly, the study of coinductive probabilistic programming was initiated only recently [3], and little is known about generalizations of coinductive concepts and methods to the probabilistic setting. In this paper we consider *productivity*, which informally ensures that one can compute arbitrarily precise approximations of infinite objects in finite time. Productivity has been studied extensively for standard, non-probabilistic coinductive languages [18, 14, 1, 12, 7], but the probabilistic setting introduces new subtleties and challenges.

Contributions

Our first contribution is conceptual. We introduce *almost sure productivity* (ASP), a probabilistic counterpart to productivity. A probabilistic stream computation is almost surely productive if it produces an infinite stream of outputs with probability 1. For instance, consider the stream defined by the equation

$$\sigma = (a : \sigma) \oplus_p \sigma$$

Viewed as a program, this stream repeatedly flips a coin with bias $p \in (0, 1)$, producing the value a if the coin comes up heads and retrying if the coin comes up tails. This computation is almost surely productive since the probability it fails to produce outputs for n consecutive steps is $(1 - p)^n$, which tends to zero as n increases. In contrast, consider the stream

$$\sigma = \bar{a} \oplus_p \epsilon$$

This computation flips a single biased coin and returns an infinite stream of a's if the coin comes up heads, and the empty stream ϵ if the coin comes up tails. This process is *not* almost surely productive since its probability of outputting an infinite stream is only p, which is strictly less than 1.

We define almost sure productivity for any system that can be equipped with a final coalgebra semantics in the style of Kerstan and König [20] (Section 3). We instantiate our semantics on a core probabilistic language for computing over streams and trees (Section 4). Then, we consider two methods for proving almost sure productivity.

1. We begin with a syntactic method that assigns a real-valued measure to each expression e (Section 5). Intuitively, the measure represents the expected difference between the number of outputs produced and consumed per evaluation step of the expression. For instance, the computation that repeatedly flips a fair coin and outputs a value if the coin is heads has measure $\frac{1}{2}$ – with probability $1/2$ it produces an output, with probability 0 it produces no outputs. More complex terms in our language can also consume outputs internally, leading to possibly negative values for the productivity measure.
   We show that every expression whose measure is strictly positive is almost surely productive; the proof of soundness of the method uses concentration results from martingale theory. While simple to carry out, our syntactic method is incomplete – it does not yield any information for expressions with non-positive measure.
2. To give a more sophisticated analysis, we reduce the problem of deciding ASP to probabilistic model-checking (Section 6). We translate our programs to probabilistic pushdown automata and show that almost sure productivity is characterized by a logical formula in LTL. This fragment is known to be decidable [6], giving a sound and complete procedure for deciding ASP.

We consider more advanced generalizations and extensions in Section 7, survey related work in Section 8, and conclude in Section 9.

2 Mathematical Preliminaries

This section reviews basic notation and definitions from measure theory and category theory. Given a set A we will denote by $A_\perp$ the coproduct of A with a one-element set containing a distinguished element $\perp$, i.e., $A_\perp = A + \{\perp\}$.

Coalgebra, Monads, Kleisli categories

We assume that the reader is familiar with the notions of objects, morphisms, functors and natural transformations (see, for instance, [5]). Given an endofunctor $F : \mathcal{C} \to \mathcal{C}$, a *coalgebra* of F is a pair (X, f) of an object $X \in \mathcal{C}$ and a morphism $f : X \to F(x)$ in $\mathcal{C}$. A *monad* is a triple (T, η, μ) of an endofunctor $T : \mathcal{C} \to \mathcal{C}$ and two natural transformations $\eta : 1_\mathcal{C} \to T$ (the *unit*) and $\mu : T^2 \to T$ (the *multiplication*) such that $\mu \circ \mu_T = \mu \circ T\mu$ and $\mu \circ \eta_T = 1_T = \mu \circ T\eta$. Given a category $\mathcal{C}$ and a monad (T, η, μ), the *Kleisli category* $\mathcal{K}\ell(T)$ of T has as objects the objects of $\mathcal{C}$ and as morphisms $X \to Y$ the morphisms $X \to T(Y)$ in $\mathcal{C}$.

Streams, Trees, Final Coalgebra

We will denote by O^ω the set of infinite streams of elements of O (alternatively characterized as functions $\mathbb{N} \to O$). We have functions $\mathsf{head} \colon O^\omega \to O$ and $\mathsf{tail} \colon O^\omega \to O^\omega$ that enable observation of the elements of the stream. In fact, they provide O^ω with a one-step structure that is canonical: given any set S and any two functions $h \colon S \to O$ and $t \colon S \to S$ (i.e., a coalgebra $(S, \langle h, t\rangle)$ of the functor $F(X) = O \times X$) there exists a *unique* stream function associating semantics to elements of S:

$$\begin{array}{ccc} S & \overset{[\![-]\!]}{\dashrightarrow} & O^\omega \\ {\scriptstyle \langle h,t\rangle}\downarrow & & \downarrow{\scriptstyle \langle \mathsf{head},\mathsf{tail}\rangle} \\ O \times S & \overset{id\times[\![-]\!]}{\dashrightarrow} & O \times O^\omega \end{array}$$

Formally, this uniqueness property is known as *finality*: O^ω is the *final coalgebra* of the functor $F(X) = O \times X$ and the above diagram gives rise to a coinductive definition principle. A similar principle can be obtained for infinite binary trees and other algebraic datatypes. The above diagrams are in the category of sets and functions, but infinite streams and trees have a very rich algebraic structure and they are also the carrier of final coalgebras in other categories. For the purpose of this paper, we will be particularly interested in a category where the maps are probabilistic – the Kleisli category of the distribution (or *Giry*) monad.

Probability Distributions, σ-algebras, Measurable Spaces

To model probabilistic behavior, we need some basic concepts from measure theory (see, e.g., [30]). Given an arbitrary set X we call a set Σ of subsets of X a *σ-algebra* if it contains the empty set and is closed under complement and countable union. A *measurable space* is a pair (X, Σ). A *probability measure* or distribution μ over a measurable space is a function $\mu : \Sigma \to [0, 1]$ assigning probabilities $\mu(A) \in [0, 1]$ to the *measurable sets* $A \in \Sigma$ such that $\mu(X) = 1$ and $\mu(\bigcup_{i\in I} A_i) = \sum_{i\in I} \mu(A_i)$ whenever $\{A_i\}_{i\in I}$ is a countable collection of disjoint measurable sets. The collection $\mathcal{D}(X)$ of probability distributions over a measurable space

X forms the so-called Giry monad. The monad unit $\eta\colon X \to \mathcal{D}(X)$ maps $a \in X$ to the point mass (or Dirac measure) δ_a, i.e., the measure assigning 1 to any set containing a and 0 to any set not containing a. The monad multiplication $m\colon \mathcal{D}\mathcal{D}(X) \to \mathcal{D}(X)$ is given by integration:

$$m(P)(S) = \int ev_S dP, \text{ where } ev_S(\mu) = \mu(S).$$

Given measurable spaces (X, Σ_X) and (Y, Σ_Y), a *Markov kernel* is a function $P : X \times \Sigma_Y \to [0,1]$ (equivalently, $X \to \Sigma_Y \to [0,1]$) that maps each source state $x \in X$ to a distribution over target states $P(x,-) : \Sigma_Y \to [0,1]$.

Markov kernels form the arrows in the Kleisli category $\mathcal{K}\ell(\mathcal{D})$ of the $\mathcal{D}$ monad; we denote such arrows by $X \xrightarrow{P} Y$. Composition in the Kleisli category is given by integration:

$$X \xrightarrow{P} Y \xrightarrow{Q} Z \qquad (P \circ Q)(x, A) = \int_{y \in Y} P(x, dy) \cdot Q(y, A)$$

Associativity of composition is essentially Fubini's theorem.

3 Defining Almost Sure Productivity

We will consider programs that denote probability distributions over coinductive types, such as infinite streams or trees. In this section, we focus on the definitions for programs producing streams and binary trees for simplicity, but our results should extend to arbitrary polynomial functors (see Section 7).

First, we introduce the semantics of programs. Rather than fix a concrete programming language at this point, we let $\mathbb{T}$ denote an abstract state space (e.g., the terms of a programming language or the space of program memories). The state evolves over an infinite sequence of discrete time steps. At each step, we will probabilistic observe either a concrete output ($a \in A$) or nothing ($\bot$), along with a resulting state. Intuitively, $p \in \mathbb{T}$ is ASP if its probability of producing unboundedly many outputs is 1. Formally, we give states in $\mathbb{T}$ a denotational semantics $[\![-]\!]\colon \mathbb{T} \to \mathcal{D}((A_\bot)^\omega)$ defined coinductively, starting from a given one-step semantics function that maps each term to an output in $A_\bot$ and the resulting term. Since the step function is probabilistic, we work in the Kleisli category for the distribution monad; this introduces some complications when computing the final coalgebras in this category. We take the work on probabilistic streams by Kerstan and König [20] as our starting point, and then generalize to probabilistic trees.

▶ **Theorem 1** (Finality for streams [20]). *Given a set $\mathbb{T}$ of programs endowed with a probabilistic step function $\mathsf{st}\colon \mathbb{T} \to \mathcal{D}(A_\bot \times \mathbb{T})$, there is a* unique *semantics function $[\![-]\!]$ assigning to each program a probability distribution of output streams such that the following diagram commutes in the Kleisli category $\mathcal{K}\ell(\mathcal{D})$:*

$$\begin{array}{ccc} \mathbb{T} & \xrightarrow{[\![-]\!]} & (A_\bot)^\omega \\ {\scriptstyle \mathsf{st}}\downarrow & & \downarrow{\scriptstyle <\mathsf{head},\mathsf{tail}>} \\ A_\bot \times \mathbb{T} & \xrightarrow{id \times [\![-]\!]} & A_\bot \times (A_\bot)^\omega \end{array}$$

▶ **Definition 2** (ASP for streams). A stream program $p \in \mathbb{T}$ is *almost surely productive* (ASP) if

$$\Pr_{\sigma \sim [\![p]\!]}[\sigma \text{ has infinitely many concrete output elements } a \in A] = 1.$$

For this to be a sensible definition, the event "σ has infinitely many concrete output elements $a \in A$" must be a measurable set in some σ-algebra on $(A_\perp)^\omega$. Following Kerstan and König, we take the σ-algebra generated by *cones*, sets of the form $uA^\omega = \{v \in (A_\perp)^\omega \mid u \text{ prefix of } v, u \in (A_\perp)^*\}$. Our definition evidently depends on the definition of $[\![-]\!] \colon \mathbb{T} \to \mathcal{D}(A_\perp)^\omega$; our coinductively defined semantics will be useful later for showing soundness when verifying ASP, but our definition of ASP is sensible for any semantics $[\![-]\!]$.

▶ **Example 3.** Let us consider the following program defining a stream σ recursively, in which each recursion step is determined by a coin flip with bias p:

$$\sigma = (a : \sigma) \oplus_p \mathsf{tail}(\sigma)$$

In the next section we will formally introduce this programming language, but intuitively the program repeatedly flips a coin. If the coin flip results in heads the program produces an element a. Otherwise the program tries to compute the tail of the recursive call; the first element produced by the recursive call is dropped (consumed), while subsequent elements produced (if any) are emitted as output.

To analyze the productivity behavior of this probabilistic program, we can reason intuitively. Each time the second branch is chosen, the program must choose the first branch strictly more than once in order to produce one output (since, e.g., $\mathsf{tail}(a : \sigma) = \sigma$). Accordingly, the productivity behavior of this program depends on the value of p. When p is less than $1/2$, the program chooses the first branch less often than the second branch and the program is not ASP. On the other hand, when $p > 1/2$ the program will tend to produce more elements a than are consumed by the destructors, and the above program is ASP. In the sequel, we will show two methods to formally prove this fact.

It will be convenient to represent the functor as $F(X) = A_\perp \times X$ as $A \times X + X$. In the rest of this paper we will often use the latter representation and refer to the final coalgebra as *observation streams* $\mathsf{OS} = (A_\perp)^\omega$ with structure $\mathsf{OS} \xleftarrow[\cong]{\langle \mathsf{out},\mathsf{unf}\rangle} A \times \mathsf{OS} + \mathsf{OS}$ given by $\mathsf{out}(a, \sigma) = a : \sigma$ and $\mathsf{unf}(\sigma) = \perp : \sigma$.

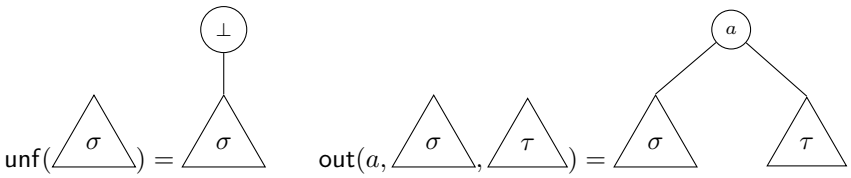
Streams are not the only coinductively defined data; infinite binary trees are another classical example. To generate trees, we can imagine that a program produces an output value – labeling the root node – and two child programs, which then generate the left and right child of a tree of outputs. Much like we saw for streams, probabilistic programs generating these trees may sometimes step to a single new program without producing outputs. Accordingly we will work with the functor $F(X) = A \times X \times X + X$, where the left summand can be thought of as the result of an output step, while the right summand gives the result of a non-output step.

▶ **Theorem 4** (Finality for trees). *Given a set of programs $\mathbb{T}$ endowed with a probabilistic step function $\mathsf{st} \colon \mathbb{T} \to \mathcal{D}(A \times \mathbb{T} \times \mathbb{T} + \mathbb{T})$, there is a* unique *semantics function $[\![-]\!]$ assigning to each program a probability distribution of output trees such that the following diagram commutes in the Kleisli category $\mathcal{K}\ell(\mathcal{D})$.*

$$\begin{array}{ccc} \mathbb{T} & \xrightarrow{\quad [\![-]\!] \quad} & \mathsf{Trees}(A_\perp) \\ {\scriptstyle \mathsf{st}}\downarrow & & \downarrow{\scriptstyle \langle \mathsf{out},\mathsf{unf}\rangle^{-1}} \\ A \times \mathbb{T} \times \mathbb{T} + \mathbb{T} & \xrightarrow{id \times [\![-]\!] \times [\![-]\!] + [\![-]\!]} & A \times \mathsf{Trees}(A_\perp) \times \mathsf{Trees}(A_\perp) + \mathsf{Trees}(A_\perp) \end{array}$$

$\mathsf{Trees}(A_\perp)$ are infinite trees where the nodes are either elements of A or $\perp$. An a-node has two children whereas a $\perp$-node only has one child. Formally, we can construct these trees

with the two maps out and unf:



Defining ASP for trees is a bit more subtle than for streams. Due to measurability issues, we can only refer to the probability of infinitely many outputs along one path at a time in the tree. A bit more formally, let $w \in \{L, R\}^\omega$ be an infinite word on alphabet $\{L, R\}$. Given any tree $t \in \mathsf{Trees}(A_\perp)$, w induces a single path t_w in the tree: from the root, the path follows the left/right child of a-nodes as indicated by w, and the single child of $\perp$-nodes.

▶ **Definition 5** (ASP for trees)**.** A tree program $p \in \mathbb{T}$ is *almost surely productive* (ASP) if

$$\forall w \in \{L, R\}^\omega.\ \Pr_{t \sim [\![p]\!]}[t_w \text{ has infinitely many concrete output nodes } a \in A] = 1.$$

We have omitted the σ-algebra structure on $\mathsf{Trees}(A_\perp)$ for lack of space, but it is quite similar to the one for streams: it is generated by the cones $u\mathsf{Trees}(A_\perp) = \{t \in \mathsf{Trees}(A_\perp) \mid t \text{ is an extension of the finite tree } u\}$.

▶ **Example 6.** Consider the probabilistic tree defined by the following equation:

$$\tau = \mathsf{mk}(a, \tau, \tau) \oplus_p \mathsf{left}(\tau)$$

The $\mathsf{mk}(a, t_1, t_2)$ constructor produces a tree with the root labeled by a and children t_1 and t_2, while the $\mathsf{left}(t)$ destructor consumes the output at the root of t and steps to the left child of t. While this example is more difficult to work out informally, it has similar ASP behavior as the previous example we saw for streams: when $p > 1/2$ this program is ASP, since it has strictly higher probability of constructing a node (and producing an output) than destructing a node (and consuming an output).

4 A Calculus for Probabilistic Streams and Trees

Now that we have introduced almost sure productivity, we consider how to verify this property. We work with two variants of a simple calculus for probabilistic coinductive programming, for producing streams and trees respectively. We suppose that outputs are drawn from some finite alphabet A. The language for streams considers terms of the following form:

$$e \in \mathbb{T} ::= \sigma \mid e \oplus_p e \mid a : e\ (a \in A) \mid \mathsf{tail}(e)$$

The distinguished variable σ represents a recursive occurrence of the stream so that streams can be defined via equations $\sigma = e$. The operation $e_1 \oplus_p e_2$ selects e_1 with probability p and e_2 with probability $1 - p$. The constructor $a : e$ builds a stream with head a and tail e. The destructor $\mathsf{tail}(e)$ computes the tail of a stream, discarding the head.

The language for trees is similar, with terms of the following form:

$$e \in \mathbb{T} ::= \tau \mid e \oplus_p e \mid \mathsf{mk}(a, e, e)\ (a \in A) \mid \mathsf{left}(e) \mid \mathsf{right}(e)$$

The variable τ represents a recursive occurrence of the tree, so that trees are defined as $\tau = e$. The constructor $\mathsf{mk}(a, e_1, e_2)$ builds a tree with root labeled a and children e_1 and e_2. The destructors $\mathsf{left}(e)$ and $\mathsf{right}(e)$ extract the left and right children of e, respectively.

We interpret these terms coalgebraically by first giving a step function from $\mathsf{st}_e : \mathbb{T} \to \mathcal{D}(F(\mathbb{T}))$ for an appropriate functor, and then taking the semantics as the map to the final coalgebra. For streams, we take the functor $F(X) = A \times X + X$: a term steps to a distribution over either an output in A and a resulting term, or just a resulting term (with no output). To describe how the recursive occurrence σ steps, we parametrize the step function st_e by the top level stream term e; this term remains fixed throughout the evaluation. This choice restricts recursion to be global in nature, i.e., our language does not support mutual or nested recursion. Supporting more advanced recursion is also possible, but we stick with the simpler setting here; we return to this point in Section 7.

The step relation is defined by case analysis on the syntax of terms. Probabilistic choice terms reduce by scaling the result of stepping e and the result of stepping e' by p and $1-p$ respectively, and then combining the distributions:

$$\mathsf{st}_e(e_1 \oplus_p e_2) \triangleq p \cdot \mathsf{st}_e(e_1) + (1-p) \cdot \mathsf{st}_e(e_2)$$

The next cases push destructors into terms:

$$\mathsf{st}_e(\mathsf{tail}^k(a : e)) \triangleq \mathsf{st}_e(\mathsf{tail}^{k-1}(e))$$
$$\mathsf{st}_e(\mathsf{tail}^k(e_1 \oplus_p e_2)) \triangleq \mathsf{st}_e(\mathsf{tail}^k(e_1) \oplus_p \mathsf{tail}^k(e_2))$$

Here and below, we write tail^k as a shorthand for $k > 0$ applications of tail.

The remaining cases return point distributions. If we have reached a constructor then we produce a single output. Otherwise, we replace σ by the top level stream term, unfolding a recursive occurrence.

$$\mathsf{st}_e(a : e') \triangleq \delta(inl(a, e'))$$
$$\mathsf{st}_e(e') \triangleq \delta(inr(e'[e/\sigma])) \quad \text{otherwise}$$

Note that a single evaluation step of a stream may lead to multiple constructors at top level of the term, but only one output can be recorded each step – the remaining constructors are preserved in the term and will give rise to outputs in subsequent steps.

The semantics is similar for trees. We take the functor $F(X) = (A \times X \times X) + X$: a term reduces to a distribution over either an output in A and two child terms, or a resulting term and no output. The main changes to the step relation are for constructors and destructors. The constructor $\mathsf{mk}(a, e_1, e_2)$ reduces to $\delta(inl(a, e_1, e_2))$, representing an output a this step. Destructors are handle like tail for streams, where $\mathsf{left}(\mathsf{mk}(a, e_1, e_2))$ reduces to e_1 and $\mathsf{right}(\mathsf{mk}(a, e_1, e_2))$ reduces to e_2, and $\mathsf{tail}^k(-)$ is generalized to any finite combination of $\mathsf{left}(-)$ and $\mathsf{right}(-)$.

Concretely, let $C[e]$ be any (possibly empty) combination of left and right applied to e. We have the following step rules:

$$\mathsf{st}_e(C[\mathsf{left}(\mathsf{mk}(a, e_l, e_r))]) \triangleq \mathsf{st}_e(C[e_l])$$
$$\mathsf{st}_e(C[\mathsf{right}(\mathsf{mk}(a, e_l, e_r))]) \triangleq \mathsf{st}_e(C[e_r])$$
$$\mathsf{st}_e(C[e_1 \oplus_p e_2]) \triangleq p \cdot \mathsf{st}_e(C[e_1]) + (1-p) \cdot \mathsf{st}_e(C[e_2])$$
$$\mathsf{st}_e(\mathsf{mk}(a, e_l, e_r)) \triangleq \delta(inl(a, e_l, e_r))$$
$$\mathsf{st}_e(C[\tau]) \triangleq \delta(inr(C[e]))$$

5 Syntactic Conditions for ASP

With the language and semantics in hand, we now turn to proving ASP. While it is theoretically possible to reason directly on the semantics using our definitions from Section 3, in practice it

is much easier to reason about the language. In this section we present a syntactic sufficient condition for ASP. Intuitively, the idea is to approximate the expected number of outputs every step; if this measure is strictly positive, then the program is ASP.

5.1 A Syntactic Measure

We define a syntactic measure $\#(-) : \mathbb{T} \to \mathbb{R}$ by induction on stream terms:

$$\begin{aligned} \#(\sigma) &\triangleq 0 \\ \#(e_1 \oplus_p e_2) &\triangleq p \cdot \#(e_1) + (1-p) \cdot \#(e_2) \\ \#(a : e) &\triangleq \#(e) + 1 \\ \#(\mathsf{tail}(e)) &\triangleq \#(e) - 1 \end{aligned}$$

The measure # describes the expected difference between the number of outputs produced (by constructors) and the number of outputs consumed (by destructors) in each unfolding of the term. We can define a similar measure for tree terms:

$$\begin{aligned} \#(\tau) &\triangleq 0 \\ \#(e_1 \oplus_p e_2) &\triangleq p \cdot \#(e_1) + (1-p) \cdot \#(e_2) \\ \#(\mathsf{mk}(a, e_1, e_2)) &\triangleq \min(\#(e_1), \#(e_2)) + 1 \\ \#(\mathsf{left}(e)) = \#(\mathsf{right}(e)) &\triangleq \#(e) - 1 \end{aligned}$$

We can now state conditions for ASP for streams and trees.

▶ **Theorem 7.** *Let e be a stream term with $\gamma = \#(e)$. If $\gamma > 0$, e is ASP.*

▶ **Theorem 8.** *Let e be a tree term with $\gamma = \#(e)$. If $\gamma > 0$, e is ASP.*

The main idea behind the proof for streams is that by construction of the step relation, each step either produces an output or unfolds a fixed point (if there is no output). In unfolding steps, the expected measure of the term plus the number of outputs increases by γ. By defining an appropriate martingale and applying the Azuma-Hoeffding inequality, the sum of the measure and the number of outputs must increase linearly as the term steps when $\gamma > 0$. Since the measure is bounded above – when the measure is large the stream outputs instead of unfolding – the number of outputs must increase linearly and the stream is ASP.

The proof for trees is similar, showing that on any path through the observation tree there are infinitely many output steps with probability 1. We present detailed proofs in the full version of this paper.

5.2 Examples

We consider a few examples of our analysis. The alphabet A does not affect the ASP property; without loss of generality, we can let the alphabet A be the singleton $\{\star\}$.

▶ **Example 9.** Consider the stream definition $\sigma = (\star : \sigma) \oplus_p \mathsf{tail}(\sigma)$. The # measure of the stream term is $p \cdot 1 + (1-p) \cdot (-1) = 2p - 1$. By Theorem 7, the stream is ASP when $p > 1/2$.

The measure does not give useful information when # is not positive.

▶ **Example 10.** Consider the stream definition $\sigma = (\star : \sigma) \oplus_{1/2} \mathsf{tail}(\sigma)$; the # measure of the term is 0. The number of outputs can be modeled by a simple random walk on a line, where

the maximum position is the number of outputs produced by the stream. Since a simple random walk has probability 1 of reaching every $n \in \mathbb{N}$ [26], the stream term is ASP.

In contrast, $\#(\sigma)$ is 0 but the stream definition $\sigma = \sigma$ is clearly non-productive.

We can give similar examples for tree terms.

▶ **Example 11.** Consider the tree definitions $\tau = e_i$, where

- $e_1 \triangleq \mathsf{left}(\tau) \oplus_{1/4} \mathsf{mk}(\star, \tau, \tau)$
- $e_2 \triangleq \mathsf{left}(\tau) \oplus_{1/4} \mathsf{mk}(\star, \tau, \mathsf{left}(\tau))$.

We apply Theorem 8 to deduce ASP. We have $\#(e_1) = (1/4) \cdot (-1) + (3/4) \cdot (+1) = 1/2$, so the first term is ASP. For the second term, $\#(e_2) = (1/4) \cdot (-1) + (3/4) \cdot 0 = -1/4$, so our analysis does not give any information.

6 Probabilistic Model-Checking for ASP

The syntactic analysis for ASP is simple, but it is not complete – no information is given if the measure is not positive. In this section we give a more sophisticated, complete analysis by first modeling the operational semantics of a term by a *Probabilistic Pushdown Automaton* (pPDA), then deciding ASP by reduction to model-checking.

6.1 Probabilistic Pushdown Automata and LTL

A pPDA is a tuple $\mathcal{A} = (S, \Gamma, \mathcal{T})$ where S is a finite set of states and Γ is a finite *stack alphabet*. The *transition function* $\mathcal{T} : S \times (\Gamma \cup \{\bot\}) \times S \times \Gamma^* \to [0, 1]$ looks at the top symbol on the stack (which might be empty, denoted $\bot$), consumes it, and pushes a (possibly empty, denoted ε) string of symbols onto the stack, before transitioning to the next state. A *configuration* of $\mathcal{A}$ is an element of $\mathcal{C} = S \times \Gamma^*$, and represents the state of the pPDA and the contents of its stack (with the top on the left) at some point of its execution. Given a configuration, the transition function $\mathcal{T}$ specifies a distribution over configurations in the next step. Given an initial state s and an initial stack $\gamma \in \Gamma^*$, $\mathcal{T}$ induces a distribution $\mathrm{Paths}(s, \gamma)$ over the infinite sequence of configurations starting in (s, γ).

Linear Temporal Logic (LTL) [29] is a linear-time temporal logic that describes runs of a transition system, which in a pPDA correspond to infinite sequences of configurations. Propositions in LTL are defined by the syntax

$$\phi, \psi ::= Q \mid \neg\phi \mid \mathcal{X}\phi \mid \phi \,\mathcal{U}\, \psi \mid \Diamond\phi \mid \Box\phi$$

where ϕ, ψ are *path formulas*, which describe a particular path, and Q is a set of atomic propositions. The validity of an LTL formula on a run π of pPDA $\mathcal{A}$ is defined as follows:

$$\pi \models \Phi \Leftrightarrow \pi[0] \in Q \qquad \pi \models \phi \,\mathcal{U}\, \psi \Leftrightarrow \exists i. \pi_i \models \psi \wedge \forall j < i. \pi_j \models \phi$$
$$\pi \models \neg\phi \Leftrightarrow \pi \not\models \phi \qquad \pi \models \Diamond\phi \Leftrightarrow \exists i. \pi_i \models \phi$$
$$\pi \models \mathcal{X}\phi \Leftrightarrow \pi_1 \models \phi \qquad \pi \models \Box\phi \Leftrightarrow \forall i. \pi_i \models \phi$$

where atomic propositions q are interpreted as $[\![q]\!] \subseteq \mathcal{C}$. As expected, path formulas are interpreted in traces of configurations $\pi \in \mathcal{C}^\omega$; $\pi[i]$ is the ith element in the path π, and π_i is the suffix of π from $\pi[i]$.

Given a pPDA $\mathcal{A}$, a starting configuration $(s, \gamma) \in \mathcal{C}$ and a LTL formula ϕ, the *qualitative model-checking problem* is to decide whether runs starting from (s, γ) satisfy ϕ almost surely, i.e., whether $\Pr_{\pi \in \mathrm{Paths}(s,\gamma)}[\pi \models \phi] = 1$. The following is known.

▶ **Theorem 12** (Brázdil, et al. [6]). *The quantitative model-checking problem for pPDAs against LTL specifications is decidable.*

Almost sure productivity states that an event – namely, producing an output – occurs infinitely often with probability 1. Such properties can be expressed in LTL.

▶ **Lemma 13.** *Let $(s,\gamma) \in \mathcal{C}$ be an initial configuration and $\mathcal{B} \subseteq \mathcal{C}$ be a set of configurations. Then $\Pr_{\pi \in \mathrm{Paths}(s,\gamma)}[\pi$ visits $\mathcal{B}$ infinitely often$] = 1$ iff $\Pr_{\pi \in \mathrm{Paths}(s,\gamma)}[\pi \models \Box\Diamond\mathcal{B}] = 1$.*

We will encode language terms as pPDAs and cast almost sure productivity as an LTL property stating that configurations representing output steps are reached infinitely often with probability 1. Theorem 12 then gives a decision procedure for ASP. In general, this algorithm[1] is in **PSPACE**.

6.2 Modeling streams with pPDAs

The idea behind our encoding from terms to pPDAs is simple to describe. The states of the pPDA will represent subterms of the original term, and transitions will model steps. In the original step relation, the only way a subterm can step to a non-subterm is by accumulating destructors. We use a single-letter stack alphabet to track the number of destructors so that a term like $\mathsf{tail}^k(e)$ can be modeled by the state corresponding to e and k counters on the stack. More formally, given a stream term e we define a pPDA $\mathcal{A}_e = (\mathcal{S}_e, \{tl\}, \mathcal{T}_e)$, where $\mathcal{S}_e$ is the set of syntactic subterms of e and $\mathcal{T}_e$ is the following transition function:

$$\begin{aligned}
\mathcal{T}_e((\sigma,a),(e,a)) &= 1 & \mathcal{T}_e((a':e',\bot),(e',\varepsilon)) &= 1\\
\mathcal{T}_e((e_1 \oplus_p e_2,a),(e_1,a)) &= p & \mathcal{T}_e((a':e',tl),(e',\varepsilon)) &= 1\\
\mathcal{T}_e((e_1 \oplus_p e_2,a),(e_2,a)) &= 1-p & \mathcal{T}_e((\mathsf{tail}(e'),a),(e',tl\cdot a) &= 1
\end{aligned}$$

Above, $\cdot$ concatenates strings and we implicitly treat a as alphabet symbol or a singleton string. All non-specified transitions have zero probability. We define the set of *outputting configurations* as $\mathcal{O} \triangleq \{s \in \mathcal{C} \mid \exists a', e'.\, s = (a' : e', \bot)\}$, that is, configurations where the current term is a constructor and there are no pending destructors. Our main result states that this set is visited infinitely often with probability 1 if and only if e is ASP. In fact, we prove something stronger:

▶ **Theorem 14.** *Let e be a stream term and let $\mathcal{A}_e$ be the corresponding pPDA. Then,*

$$\Pr_{t\sim[\![e]\!]}[t \text{ has infinitely many output nodes}] = \Pr_{\pi\sim\mathrm{Paths}(e,\varepsilon)}[\pi \models \Box\Diamond\,\mathcal{O}].$$

In particular, e is ASP if and only if for almost all runs π starting in (e,ε), $\pi \models \Box\Diamond\mathcal{O}$.

By Theorem 12, ASP is decidable for stream terms. In fact, it is also possible to decide whether a stream term is almost surely *not* productive, i.e., the probability of producing infinitely many outputs is zero.

[1] Technically, this algorithm requires first encoding the LTL formula into a Deterministic Rabin Automaton (DRA). Even though this encoding can in general blow up the problem size exponentially, this is not the case for the simple conditions we consider.

6.3 Extending to trees

Now, we extend our approach to trees. The main difficulty can be seen in the constructors. For streams, we can encode the term $a : e$ by proceeding to the tail e. For trees, however, how can we encode $\mathsf{mk}(a, e_1, e_2)$? The pPDA cannot step to both e_1 and e_2. Since the failure of ASP may occur down either path, we cannot directly translate the ASP property on trees to LTL – ASP is a property of *all* paths down the tree. Instead, on constructors our pPDA encoding will choose a path at random to simulate. As we will show, if the probability of choosing a path that outputs infinitely often is 1, then every path will output infinitely often. Notice that in general, properties that happen with probability 1 do not necessarily happen for every path, but the structure of our problem allows us to make this generalization.

More formally, the stack alphabet will now be $\{rt, lt\}$, and on constructors we transition to each child with probability $1/2$:

$$\begin{aligned}
\mathcal{T}_e((\tau, a), (e, a)) &= 1 \\
\mathcal{T}_e((e_1 \oplus_p e_2, a), (e_1, a)) &= p \\
\mathcal{T}_e((e_1 \oplus_p e_2, a), (e_2, a)) &= 1 - p \\
\mathcal{T}_e((\mathsf{mk}(a', e_l, e_r), \bot), (e_l, \varepsilon)) &= 1/2 \\
\mathcal{T}_e((\mathsf{mk}(a', e_l, e_r), \bot), (e_r, \varepsilon)) &= 1/2
\end{aligned}$$

$$\begin{aligned}
\mathcal{T}_e((\mathsf{mk}(a', e_l, e_r), lt), (e_l, \varepsilon)) &= 1 \\
\mathcal{T}_e((\mathsf{mk}(a', e_l, e_r), rt), (e_r, \varepsilon)) &= 1 \\
\mathcal{T}_e((\mathsf{left}(e'), a), (e', lt \cdot a)) &= 1 \\
\mathcal{T}_e((\mathsf{right}(e'), a), (e', rt \cdot a)) &= 1
\end{aligned}$$

We define $\mathcal{O} \triangleq \{s \in \mathcal{C} \mid \exists a', e_l, e_r.\, s = (\mathsf{mk}(a', e_l, e_r), \varepsilon)\}$ to be the set of outputting configurations, and we can characterize ASP with the following theorem.

▶ **Theorem 15.** *Let e be a tree term and $\mathcal{A}_e$ be the corresponding probabilistic PDA. Then $\Pr_{\pi \sim \mathrm{Paths}(e,\bot)}[\pi \models \Box\Diamond\, \mathcal{O}] = 1$ if and only if for every $w \in \{L, R\}^\omega$,*

$$\Pr_{t \sim [\![e]\!]}[t \text{ has infinitely many output nodes along } w] = 1.$$

Thus we can decide ASP by deciding a LTL formula.

7 Possible Generalizations and Extensions

Our definition of ASP and our verification approaches suggest several natural directions.

Handling Richer Languages. The most concrete direction is to consider richer languages for coinductive probabilistic programming. Starting from our core language, one might consider allowing more operations on coinductive terms, mutually recursive definitions, or conditional tests of some kind. It should also be possible to develop languages for more complex coinductive types associated with general polynomial functors (see, e.g., Kozen [24]). Note that adding more operations, e.g. pointwise + of streams would increase the expressivity of the language but raise additional challenges from the perspective of the semantics – we would have to add extra structure to the base category and re-check that the finality proof.

Developing new languages for coinductive probabilistic programming – perhaps an imperative language or a higher-order language – would also be interesting. From the semantics side, our development in Section 3 should support any language equipped with a small-step semantics producing output values, allowing ASP to be defined for many kinds of languages. The verification side appears more challenging. Natural extensions, like a pointwise addition operation, already seem to pose challenges for the analyses. We know of no general method to reasoning about ASP. This stands in contrast to almost sure termination, which can be established by where flexible criteria like decreasing probabilistic variants [17]. Considering counterparts of these methods for ASP is an interesting avenue of research.

Exploring Other Definitions. Our definition of ASP is natural, but other definitions are possible. For trees (and possibly more complex coinductive structures), we could instead require that there *exists* a path producing infinitely many outputs, rather than requiring that *all* paths produce infinitely many outputs. This weaker notion of ASP can be defined in our semantics, but it is currently unclear how to verify this kind of ASP.

Our notion of ASP also describes just the probability of generating infinitely many outputs, and does not impose any requirement on the generation rate. Quantitative strengthenings of ASP – say, requiring bounds on the expected number of steps between outputs – could give more useful information.

Understanding Dependence on Step Relation. Our coalgebraic semantics supporting our verification methods are based on a small-step semantics for programs. A natural question is whether this dependence is necessary, or if one could verify ASP with a less step-dependent semantics. Again drawing an analogy, it appears that fixing a reduction strategy is important in order to give a well-defined notion of almost sure termination for probabilistic higher-order languages (see, e.g., [25]). The situation for almost sure productivity is less clear.

8 Related Work

Our work is inspired by two previously independent lines of research: probabilistic termination and productivity of coalgebraic definitions.

Probabilistic Termination. There are a broad range of techniques for proving termination of probabilistic programs. Many of the most powerful criteria use advanced tools from probability theory [27], especially martingale theory [8, 16, 9, 10, 11]. Other works adopt more pragmatic approaches, generally with the goal of achieving automation. Arons, Pnueli and Zuck [4] reduce almost sure termination of a program P to termination of a non-deterministic program Q, using a planner that must be produced by the verifier. Esparza, Gaiser and Kiefer [15] give a CEGAR-like approach for building patterns (which play a role similar to planners) and prove that their approach is complete for a natural class of programs.

Productivity of Corecursive Definitions. There has been a significant amount of work on verifying productivity of corecursive definitions without probabilistic choice. Endrullis and collaborators [14] give a procedure for deciding productivity of an expressive class of stream definitions. In a companion work [13], they study the strength of data oblivious criteria, i.e., criteria that do not depend on values. More recently, Komendantskaya and collaborators [21] introduce observational productivity and give a semi-decision procedure for logic programs.

9 Conclusion

We introduce almost sure productivity, a counterpart to almost sure termination for probabilistic coinductive programs. In addition, we propose two methods for proving ASP for a core language for streams and infinite trees. Our results demonstrate that verification of ASP is feasible and can even be decidable for simple languages.

References

1. Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns: Programming infinite structures by observations. In Roberto Giacobazzi and Radhia Cousot, editors, *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Rome, Italy*, pages 27–38, 2013. doi:10.1145/2429069.2429075.
2. Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *Proceedings of the ACM on Programming Languages*, 2(POPL):34:1–34:32, 2018. doi:10.1145/3158122.
3. Alejandro Aguirre, Gilles Barthe, Lars Birkedal, Ales Bizjak, Marco Gaboardi, and Deepak Garg. Relational reasoning for Markov chains in a probabilistic guarded lambda calculus. In *European Symposium on Programming (ESOP), Thessaloniki, Greece*, Lecture Notes in Computer Science. Springer-Verlag, 2018.
4. Tamarah Arons, Amir Pnueli, and Lenore D. Zuck. Parameterized verification by probabilistic abstraction. In Andrew D. Gordon, editor, *International Conference on Foundations of Software Science and Computation Structures (FoSSaCS), Warsaw, Poland*, volume 2620 of *Lecture Notes in Computer Science*, pages 87–102. Springer-Verlag, 2003. doi:10.1007/3-540-36576-1_6.
5. S. Awodey. *Category Theory*. Oxford Logic Guides. Ebsco Publishing, 2006. URL: https://books.google.es/books?id=IK_sIDI2TCwC.
6. Tomáš Brázdil, Javier Esparza, Stefan Kiefer, and Antonín Kučera. Analyzing probabilistic pushdown automata. *Formal Methods in System Design*, 43(2):124–163, Oct 2013. doi:10.1007/s10703-012-0166-0.
7. Venanzio Capretta and Jonathan Fowler. The continuity of monadic stream functions. In *IEEE Symposium on Logic in Computer Science (LICS), Reykjavik, Iceland*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005119.
8. Aleksandar Chakarov and Sriram Sankaranarayanan. Probabilistic program analysis with martingales. In *International Conference on Computer Aided Verification (CAV), Saint Petersburg, Russia*, pages 511–526, 2013. URL: https://www.cs.colorado.edu/~srirams/papers/cav2013-martingales.pdf.
9. Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. Termination analysis of probabilistic programs through Positivstellensatz's. In *International Conference on Computer Aided Verification (CAV), Toronto, Ontario*, volume 9779 of *Lecture Notes in Computer Science*, pages 3–22. Springer-Verlag, 2016. doi:10.1007/978-3-319-41528-4_1.
10. Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Saint Petersburg, Florida*, pages 327–342, 2016. doi:10.1145/2837614.2837639.
11. Krishnendu Chatterjee, Petr Novotný, and Đorđe Žikelić. Stochastic invariants for probabilistic termination. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Paris, France*, pages 145–160, 2017. doi:10.1145/3009837.3009873.
12. Ranald Clouston, Ales Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. The guarded lambda-calculus: Programming and reasoning with guarded recursion for coinductive types. *Logical Methods in Computer Science*, 12(3), 2016. doi:10.2168/LMCS-12(3:7)2016.
13. Jörg Endrullis, Clemens Grabmayer, and Dimitri Hendriks. Data-oblivious stream productivity. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR), Doha, Qatar*, volume 5330 of *Lecture Notes in Computer Science*, pages 79–96. Springer-Verlag, 2008. doi:10.1007/978-3-540-89439-1_6.

14 Jörg Endrullis, Clemens Grabmayer, Dimitri Hendriks, Ariya Isihara, and Jan Willem Klop. Productivity of stream definitions. *Theor. Comput. Sci.*, 411(4-5):765–782, 2010. doi:10.1016/j.tcs.2009.10.014.

15 Javier Esparza, Andreas Gaiser, and Stefan Kiefer. Proving termination of probabilistic programs using patterns. In P. Madhusudan and Sanjit A. Seshia, editors, *International Conference on Computer Aided Verification (CAV), Berkeley, California*, volume 7358 of *Lecture Notes in Computer Science*, pages 123–138. Springer-Verlag, 2012. doi:10.1007/978-3-642-31424-7_14.

16 Luis María Ferrer Fioriti and Holger Hermanns. Probabilistic termination: Soundness, completeness, and compositionality. In Sriram K. Rajamani and David Walker, editors, *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), Mumbai, India*, pages 489–501, 2015. doi:10.1145/2676726.2677001.

17 Sergiu Hart, Micha Sharir, and Amir Pnueli. Termination of probabilistic concurrent program. *ACM Trans. Program. Lang. Syst.*, 5(3):356–380, 1983. doi:10.1145/2166.357214.

18 John Hughes, Lars Pareto, and Amr Sabry. Proving the correctness of reactive systems using sized types. In Hans-Juergen Boehm and Guy L. Steele Jr., editors, *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), St. Petersburg Beach, Florida*, pages 410–423, 1996. doi:10.1145/237721.240882.

19 Benjamin Lucien Kaminski and Joost-Pieter Katoen. On the hardness of almost-sure termination. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Symposium on Mathematical Foundations of Computer Science (MFCS), Milan, Italy*, volume 9234 of *Lecture Notes in Computer Science*, pages 307–318. Springer-Verlag, 2015. doi:10.1007/978-3-662-48057-1_24.

20 Henning Kerstan and Barbara König. Coalgebraic trace semantics for continuous probabilistic transition systems. *Logical Methods in Computer Science*, 9(4), 2013. doi:10.2168/LMCS-9(4:16)2013.

21 Ekaterina Komendantskaya, Patricia Johann, and Martin Schmidt. A productivity checker for logic programming. In Manuel V. Hermenegildo and Pedro López-García, editors, *International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR), Edinburgh, Scotland*, volume 10184 of *Lecture Notes in Computer Science*, pages 168–186. Springer-Verlag, 2016. doi:10.1007/978-3-319-63139-4_10.

22 Dexter Kozen. Semantics of probabilistic programs. *J. Comput. Syst. Sci.*, 22(3):328–350, 1981. doi:10.1016/0022-0000(81)90036-2.

23 Dexter Kozen. A probabilistic PDL. In *ACM SIGACT Symposium on Theory of Computing (STOC), Boston, Massachusetts*, pages 291–297, 1983. doi:10.1145/800061.808758.

24 Dexter Kozen. Realization of coinductive types. *Electronic Notes in Theoretical Computer Science*, 276:237–246, 2011. doi:10.1016/j.entcs.2011.09.024.

25 Ugo Dal Lago and Charles Grellois. Probabilistic termination by monadic affine sized typing. In Hongseok Yang, editor, *European Symposium on Programming (ESOP), Uppsala, Sweden*, volume 10201 of *Lecture Notes in Computer Science*, pages 393–419. Springer-Verlag, 2017. doi:10.1007/978-3-662-54434-1_15.

26 David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2009. URL: https://pages.uoregon.edu/dlevin/MARKOV/markovmixing.pdf.

27 Annabelle McIver, Carroll Morgan, Benjamin Lucien Kaminski, and Joost-Pieter Katoen. A new proof rule for almost-sure termination. *Proceedings of the ACM on Programming Languages*, 2(POPL):33:1–33:28, 2018. doi:10.1145/3158121.

28 Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *toplas*, 18(3):325–353, 1996. doi:10.1145/229542.229547.

29 Amir Pnueli. The temporal logic of programs. In *IEEE Symposium on Foundations of Computer Science (FOCS), Providence, Rhode Island*, pages 46–57, 1977. doi:10.1109/SFCS.1977.32.

30 Walter Rudin. *Real and Complex Analysis*. McGraw-Hill, New York, third edition, 1987.

O-Minimal Invariants for Linear Loops

Shaull Almagor
Department of Computer Science, Oxford University, UK
shaull.almagor@cs.ox.ac.uk

Dmitry Chistikov
Centre for Discrete Mathematics and its Applications (DIMAP) &
Department of Computer Science, University of Warwick, UK
d.chistikov@warwick.ac.uk

Joël Ouaknine[1]
Max Planck Institute for Software Systems, Germany &
Department of Computer Science, Oxford University, UK
joel@mpi-sws.org

James Worrell[2]
Department of Computer Science, Oxford University, UK
jbw@cs.ox.ac.uk

Abstract

The termination analysis of linear loops plays a key rôle in several areas of computer science, including program verification and abstract interpretation. Such deceptively simple questions also relate to a number of deep open problems, such as the decidability of the Skolem and Positivity Problems for linear recurrence sequences, or equivalently reachability questions for discrete-time linear dynamical systems. In this paper, we introduce the class of *o-minimal invariants*, which is broader than any previously considered, and study the decidability of the existence and algorithmic synthesis of such invariants as certificates of non-termination for linear loops equipped with a large class of halting conditions. We establish two main decidability results, one of them conditional on Schanuel's conjecture in transcendental number theory.

2012 ACM Subject Classification Computing methodologies → Algebraic algorithms, Theory of computation → Logic and verification

Keywords and phrases Invariants, linear loops, linear dynamical systems, non-termination, o-minimality

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.114

Related Version The full version of this paper can be found at `https://arxiv.org/abs/1802.09263`.

1 Introduction

This paper is concerned with the existence and algorithmic synthesis of suitable *invariants* for linear loops, or equivalently for discrete-time linear dynamical systems. Invariants are one of the most fundamental and useful notions in the quantitative sciences, and within computer science play a central rôle in areas such as program analysis and verification, abstract

[1] [Joël Ouaknine was supported by ERC grant AVS-ISS (648701)]
[2] [James Worrell was supported by EPSRC Fellowship EP/N008197/1.]

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 114; pp. 114:1–114:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

interpretation, static analysis, and theorem proving. To this day, automated invariant synthesis remains a topic of active research; see, e.g., [17], and particularly Sec. 8 therein.

In program analysis, invariants are often invaluable tools enabling one to establish various properties of interest. Our focus here is on simple linear loops, of the following form:

$$P: \quad x \leftarrow s; \;\; \texttt{while } x \notin F \texttt{ do } x \leftarrow Ax\,, \tag{1}$$

where x is a d-dimensional column vector of variables, s is a d-dimensional vector of integer, rational, or real numbers, $A \in \mathbb{Q}^{d\times d}$ is a square rational matrix of dimension d, and $F \subseteq \mathbb{R}^d$ represents the halting condition.

Much research has been devoted to the termination analysis of such loops (and variants thereof); see, e.g., [3, 2, 25]. For $S \subseteq \mathbb{R}^d$, we say that P *terminates* on S if it terminates for all initial vectors $s \in S$. One of the earliest and most famous results in this line of work is due to Kannan and Lipton, who showed polynomial-time decidability of termination in the case where S and F are both singleton vectors with rational entries [15, 16]. This work was subsequently extended to instances in which F is a low-dimensional vector space [6, 8] or a low-dimensional polyhedron [7]. Still starting from a fixed initial vector, the case in which the halting set F is a hyperplane is equivalent to the famous Skolem Problem for linear recurrence sequences, whose decidability has been open for many decades [29, §3.9], although once again positive results are known in low dimensions [20, 32]. The case in which F is a half-space corresponds to the Positivity Problem for linear recurrence sequences, likewise famously open in general but for which some partial results also exist [23, 22].

Cases in which the starting set S is infinite have also been extensively studied, usually in conjunction with a halting set F consisting of a half-space. For example, decidability of termination for $S = \mathbb{R}^d$ and $S = \mathbb{Q}^d$ are known [31, 4]; see also [21]. In the vast majority of cases, however, termination is a hard problem (and often undecidable [34]), which has led researchers to turn to semi-algorithms and heuristics. One of the most popular and successful approaches to establishing termination is the use of ranking functions, on which there is a substantial body of work; see, e.g., [2], which includes a broad survey on the subject.

Observe, for a loop P such as that given in (1), that failure to terminate on a set S corresponds to the existence of some vector $s \in S$ from which P loops forever. It is important to note, however, that the absence of a suitable ranking function does not necessarily entail non-termination, owing to the non-completeness of the method. Yet surprisingly, as pointed out in [14], there has been significantly less research in methods seeking to establish *non-termination* than in methods aimed at proving termination. Most existing efforts for the former have focused on the synthesis of appropriate invariants; see, e.g., [11, 9, 28, 26, 10, 27, 13].

In order to make this notion more precise, let us associate with our loop P a *discrete-time linear dynamical system* (A, s). The *orbit* of this dynamical system is the set $\mathcal{O} = \{A^n s \mid n \geq 0\}$. It is clear that P fails to terminate from s iff $\mathcal{O}$ is disjoint from F. A possible method to establish the latter is therefore to exhibit a set $\mathcal{I} \subseteq \mathbb{R}^d$ such that:

1. $\mathcal{I}$ contains the initial vector s, i.e., $s \in \mathcal{I}$;
2. $\mathcal{I}$ is invariant under A, i.e., $A\mathcal{I} \subseteq \mathcal{I}$; and
3. $\mathcal{I}$ is disjoint from F, i.e., $\mathcal{I} \cap F = \emptyset$.

Indeed, the first two conditions ensure that $\mathcal{I}$ contains the entire orbit $\mathcal{O}$, from which the desired claim follows thanks to the third condition.

In instances of non-termination, one notes that the orbit $\mathcal{O}$ itself is always an invariant meeting the above conditions. However, since in general one does not know how to algorithmically check Condition (3), such an invariant is of little use. One therefore usually first fixes a suitable class of candidate sets for which the above conditions can be mechanically verified,

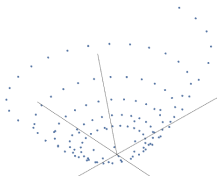


Figure 1 The orbit $\mathcal{O}$ of (A, s)

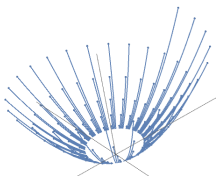


Figure 2 Trajectory rays of $\mathcal{O}$.

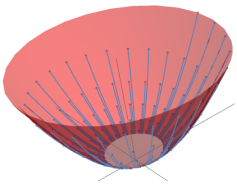


Figure 3 Trajectory cone for $\mathcal{O}$.

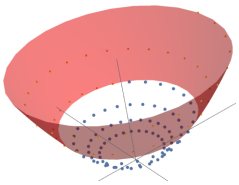


Figure 4 Invariant for $\mathcal{O}$.

and within that class, one seeks to determine if an invariant can be found. Examples of such classes include polyhedra [11], algebraic sets [27], and semi-algebraic sets [13].

Main contributions. We focus on loops of the form given in (1) above. We introduce the class of *o-minimal invariants*, which, to the best of our knowledge, is significantly broader than any of the classes previously considered. We also consider two large classes of halting sets, namely semi-algebraic sets, as well as sets definable in the first-order theory of the reals with exponentiation, denoted $\mathfrak{R}_{\exp}$. Given $s \in \mathbb{Q}^d$, $A \in \mathbb{Q}^{d\times d}$, and $F \subseteq \mathbb{R}^d$, our main results are the following: if F is a semi-algebraic set, it is decidable whether there exists an o-minimal invariant $\mathcal{I}$ containing s and disjoint from F, and moreover in positive instances such an invariant can be defined explicitly in $\mathfrak{R}_{\exp}$; for the more general case in which F is $\mathfrak{R}_{\exp}$-definable, the same holds assuming Schanuel's conjecture.

We illustrate below some of the key ideas from our approach. Consider a linear dynamical system (A, s) with $A \in \mathbb{Q}^{3\times 3}$ whose orbit $\mathcal{O}$ is depicted in Figure 1. In our example, $\mathcal{O}$ spirals outward at some rate ρ_1 in the x, y-plane, and increases along the z-axis at some rate ρ_2. Intuitively, ρ_1 and ρ_2 are the moduli of the eigenvalues of A.

We now consider a 'normalised' version of A, with both moduli set to 1. We then connect every point on the normalised orbit with a *trajectory ray* to its corresponding point on $\mathcal{O}$, while respecting the rates ρ_1 and ρ_2 (see Figure 2). One can observe that the normalised orbit is dense in the unit circle. We prove that *any* o-minimal invariant for (A, s) must in fact eventually contain every trajectory ray for every point on the unit circle; we depict the union of these rays, referred to as the *trajectory cone*, in Figure 3. Finally, we show that any o-minimal invariant must in fact contain some truncation of the trajectory cone from below, starting from some height. That is, there is a uniform bound from which all the rays must belong to the invariant. Moreover, we can now synthesise an $\mathfrak{R}_{\exp}$-definable o-minimal invariant by simply adjoining a finite number of orbit points to the truncated trajectory cone, as depicted in Figure 4.

It is worth emphasising that, whilst in general there cannot exist a smallest o-minimal invariant, the family of truncated cones that we define plays the rôle of a 'minimal class', in the sense that *any* o-minimal invariant must necessarily contain some truncated cone. We make all of these notions precise in the main body of the paper.

The work that is closest to ours in the literature is [13], which considers the same kind of loops as we do here, but restricted to the case in which the halting set F is always a rational singleton. The authors then exhibit a procedure for deciding the existence of semi-algebraic invariants. The present paper has a considerably broader scope, in that we deal with much wider classes both of invariants and halting sets. From a technical standpoint, the present paper correspondingly makes heavy use of model-theoretic and number-theoretic tools that are entirely absent from [13]. It is interesting to note, however, that the question of the existence of semi-algebraic (rather than o-minimal) invariants in the present setting appears to be a challenging open problem.

2 Preliminaries

The *first-order theory of the reals*, denoted $\mathfrak{R}_0$, is the collection of true sentences in the first-order logic of the structure $\langle \mathbb{R}, 0, 1, +, \cdot, < \rangle$. Sentences in $\mathfrak{R}_0$ are quantified Boolean combinations of atomic propositions of the form $P(x_1, \ldots, x_n) > 0$ where P is a polynomial with integer coefficients, and $x_1, \ldots, x_n$ are variables. Tarski famously showed that this theory admits quantifier elimination [30] and is therefore decidable. In addition to $\mathfrak{R}_0$, we also consider the *first-order theory of the reals with exponentiation*, denoted $\mathfrak{R}_{\exp}$, which augments $\mathfrak{R}_0$ with the exponentiation function $x \mapsto e^x$.

A set $S \subseteq \mathbb{R}^d$ is *definable* in a theory $\mathfrak{R}$ if there exists a formula $\varphi(x_1, \ldots, x_d)$ in $\mathfrak{R}$ with free variables $x_1, \ldots, x_d$ such that $S = \{(c_1, \ldots, c_d) \in \mathbb{R}^d \mid \varphi(c_1, \ldots, c_d) \text{ is true}\}$. A function $f \colon B \to \mathbb{R}^m$ with $B \subseteq \mathbb{R}^n$ is *definable* in $\mathfrak{R}$ if its graph $\Gamma(f) = \{(x, f(x)) \mid x \in B\} \subseteq \mathbb{R}^{n+m}$ is an $\mathfrak{R}$-definable set. For $\mathfrak{R} = \mathfrak{R}_0$, the first-order theory of the reals, $\mathfrak{R}_0$-definable sets (resp. functions) are known as *semi-algebraic* sets (resp. functions).

▶ Remark. Our usage of the terms "definable" and "semi-algebraic" corresponds to "definable without parameters" and "semi-algebraic without parameters" in model theory.

A theory $\mathfrak{R}$ is said to be *o-minimal* if every $\mathfrak{R}$-definable subset of the reals $S \subseteq \mathbb{R}$ is a finite union of points and (possibly unbounded) intervals.

▶ **Definition 1.** A set $S \subseteq \mathbb{R}^d$ is *o-minimal* if it is definable in some o-minimal theory that extends $\mathfrak{R}_{\exp}$.

Tarski's result on quantifier elimination [30] also implies that $\mathfrak{R}_0$ is o-minimal. The o-minimality of $\mathfrak{R}_{\exp}$, on the other hand, is due to Wilkie [33]. O-minimal theories enjoy many useful properties, some of which we list below, referring the reader to [12] for precise definitions and proofs. In what follows, $\mathfrak{R}$ is a fixed o-minimal theory.

1. For an $\mathfrak{R}$-definable set $S \subseteq \mathbb{R}^d$, its topological closure $\overline{S}$ is also $\mathfrak{R}$-definable.
2. For an $\mathfrak{R}$-definable function $f \colon S \to \mathbb{R}$, the number $\inf\{f(x) \mid x \in S\}$ is $\mathfrak{R}$-definable (as a singleton set).
3. O-minimal theories admit *cell decomposition*: every $\mathfrak{R}$-definable set $S \subseteq \mathbb{R}^d$ can be written as a finite union of connected components called *cells*. Moreover, each cell is $\mathfrak{R}$-definable and homeomorphic to $(0,1)^m$ for some $m \in \{0, 1, \ldots, d\}$. The *dimension* of S is defined as the maximal such m occurring in the cell decomposition of S.
4. For an $\mathfrak{R}$-definable function $f \colon S \to \mathbb{R}^m$, the dimension of its graph $\Gamma(f)$ is the same as the dimension of S.

As mentioned above, $\mathfrak{R}_0$ is decidable thanks to its effective quantifier elimination procedure. Equivalently, given a semi-algberaic set, we can effectively compute its cell decomposition. Unfortunately, few more expressive theories are known to be decidable. The theory $\mathfrak{R}_{\exp}$ is decidable provided that Schanuel's conjecture, an assertion in transcendental number theory, holds [18]. Our decidability result in Theorem 12 is subject to Schanuel's conjecture; somewhat surprisingly, however, we exhibit in Theorem 13 an unconditional decidability result.

▶ Remark. While all our $\mathfrak{R}$-definable sets live in $\mathbb{R}^d$, it is often convenient or necessary to consider sets in $\mathbb{C}^d$. To this end, by identifying $\mathbb{C}$ with $\mathbb{R}^2$, we define a set $S \subseteq \mathbb{C}^d$ to be $\mathfrak{R}$-definable if the set $\{(x, y) \in \mathbb{R}^d \times \mathbb{R}^d \mid x + iy \in S\}$ in $\mathbb{R}^{2d}$ is $\mathfrak{R}$-definable.

A *discrete-time linear dynamical system* (LDS) consists of a pair (A, s), where $A \in \mathbb{Q}^{d \times d}$ and $s \in \mathbb{Q}^d$. Its *orbit* $\mathcal{O}$ is the set $\{A^n s \mid n \in \mathbb{N}\}$. An *invariant* for (A, s) is a set $\mathcal{I} \subseteq \mathbb{R}^d$ that contains s and is stable under applications of A, i.e., $A\mathcal{I} \subseteq \mathcal{I}$. Given a set $F \subseteq \mathbb{R}^d$, we say that the invariant $\mathcal{I}$ *avoids* F if the two sets are disjoint.

3 From the Orbit to Trajectory Cones and Rays

Let (A, s) be an LDS with $A \in \mathbb{Q}^{d\times d}$ and $s \in \mathbb{Q}^d$. We consider the orbit $\mathcal{O} = \{A^n s \mid n \in \mathbb{N}\}$. Write A in Jordan form as $A = PJP^{-1}$ where P is an invertible matrix, and J is a block diagonal matrix of the form $J = \mathrm{diag}(B_1, \ldots, B_k)$, where for every $1 \le i \le k$, $B_i \in \mathbb{C}^{d_i \times d_i}$ is a Jordan block corresponding to an eigenvalue $\rho_i\lambda_i$:

$$B_i = \begin{pmatrix} \rho_i\lambda_i & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \rho_i\lambda_i \end{pmatrix}.$$

Here $\rho_1, \ldots, \rho_k \in \mathbb{R}_{\ge 0}$, the numbers $\lambda_1, \ldots, \lambda_k$ are algebraic and have modulus 1, and $\sum_{i=1}^{k} d_i = d$. To reflect the block structure of J, we often range over $\{1, \ldots, d\}$ via a pair (i, j), with $1 \le i \le k$ and $1 \le j \le d_i$, which denotes the index corresponding to row j in block i; we refer to this notation as *block-row indexing*.

▶ Remark. Henceforth, we assume that for all $1 \le i \le k$ we have that $\rho_i > 0$ (i.e., that the matrices A and J are invertible). Indeed, if $\rho_i = 0$, then B_i is a nilpotent block and therefore, for the purpose of invariant synthesis, we can ignore finitely many points of the orbit under A until B_i^n is the 0 block. We can then restrict our attention to the image of A^n, by identifying it with $\mathbb{R}^{d-d_i}$.

Observe that now, for every set $F \subseteq \mathbb{R}^d$, we have that $A^n s \in F$ iff $J^n s' \in P^{-1}F$ where $s' = P^{-1}s$.

For every $n > d$, $J^n = \mathrm{diag}(B_1^n, \ldots, B_k^n)$ with

$$B_i^n = \begin{pmatrix} (\rho_i\lambda_i)^n & \frac{n}{\rho_i\lambda_i}(\rho_i\lambda_i)^n & \cdots & \frac{\binom{n}{d_i-1}}{(\rho_i\lambda_i)^{d_i-1}}(\rho_i\lambda_i)^n \\ & \ddots & & \vdots \\ & & & (\rho_i\lambda_i)^n \end{pmatrix}.$$

Every coordinate of $J^n s'$ is of the form $(\rho_i\lambda_i)^n Q_{i,j}(n) = \rho_i^n \lambda_i^n Q_{i,j}(n)$ for some $1 \le i \le k$ and $1 \le j \le d_i$, where $Q_{i,j}$ is a polynomial (possibly with complex coefficients) that depends on J and s'.

Let $R = \mathrm{diag}(\rho_1, \ldots, \rho_k)$ and $L = \mathrm{diag}(\lambda_1, \ldots, \lambda_k)$. We define $\mathbb{T}$ to be the subgroup of the torus in $\mathbb{C}^k$ generated by the multiplicative relations of the normalised eigenvalues $\lambda_1, \ldots, \lambda_k$. That is, consider the subgroup $G = \{v = (v_1, \ldots, v_k) \in \mathbb{Z}^k \mid \lambda_1^{v_1} \cdots \lambda_k^{v_k} = 1\}$ of $\mathbb{Z}^k$, and let

$$\mathbb{T} = \{(\alpha_1, \ldots, \alpha_k) \in \mathbb{C}^k \mid |\alpha_i| = 1 \text{ for all } i, \text{ and for every } v \in G,\ \alpha_1^{v_1} \cdots \alpha_k^{v_k} = 1\}.$$

A result by Masser [19] allows to compute in polynomial time a basis for G, and hence a representation of $\mathbb{T}$. Using Kronecker's theorem on inhomogeneous simultaneous Diophantine approximation [5] it is shown in [24] that $\{L^n \mid n \in \mathbb{N}\}$ is a dense subset of $\{\mathrm{diag}(\alpha_1, \ldots, \alpha_k) \mid (\alpha_1, \ldots, \alpha_k) \in \mathbb{T}\}$.

Thus, for every $n \in \mathbb{N}$, we have

$$J^n s' \in \left\{ \left(\rho_1^n p_1 Q_{1,1}(n), \ldots, \rho_k^n p_k Q_{k,d_k}(n)\right)^{\intercal} \mid (p_1, \ldots, p_k) \in \mathbb{T} \right\}.$$

We now define a continuous over-approximation of the expressions ρ_i^n. To this end, if there exists some modulus ρ_i larger than 1 (in which case, without loss of generality, assume that $\rho_k > 1$), then for every $1 \le i \le k$ let $b_i = \log_{\rho_k} \rho_i$, and observe that $\rho_i^n = (\rho_k^n)^{b_i}$. We

then replace the expression ρ_k^n with a continuous variable t, so that ρ_i^n becomes t^{b_i}, and n is replaced by $\log_{\rho_k} t$. If all moduli are at most 1 and some are strictly smaller than 1 (in which case, without loss of generality, $\rho_k < 1$), then replace the expression ρ_k^n with $1/t$. Note that in both cases, t grows unboundedly large as n tends to infinity. In the full version of this paper we handle the special (and simpler) case in which all eigenvalues have modulus exactly 1. Henceforth, we assume that $\rho_k > 1$. If $\rho_k < 1$, the proofs are carried out *mutatis mutandis*.

This over-approximation leads to the following definition, which is central to our approach.

▶ **Definition 2.** For every $t_0 \geq 1$, we define the *trajectory cone*[3] for the orbit $\mathcal{O}$ as

$$\mathcal{C}_{t_0} = \left\{ \left(t^{b_1} p_1 Q_{1,1}(\log_{\rho_k} t), \ldots, t^{b_k} p_k Q_{k,d_k}(\log_{\rho_k} t)\right)^\intercal \;\middle|\; (p_1, \ldots, p_d) \in \mathbb{T},\ t \geq t_0 \right\}.$$

In particular, we have that $J^n s' \in \mathcal{C}_1$.

In order to analyse invariants, we require a finer-grained notion than the entire trajectory cone. To this end, we introduce the following.

▶ **Definition 3.** For every $p = (p_1, \ldots, p_k) \in \mathbb{T}$ and every $t_0 \geq 1$, we define the *(trajectory) ray*[4] $\mathsf{r}(p, t_0) = \left\{ \left(t^{b_1} p_1 Q_{1,1}(\log_{\rho_k} t), \ldots, t^{b_k} p_k Q_{k,d_k}(\log_{\rho_k} t)\right)^\intercal \mid t \geq t_0 \right\}$.

Observe that we have $\mathcal{C}_{t_0} = \bigcup_{p \in \mathbb{T}} \mathsf{r}(p, t_0)$.

▶ **Example 4.** Consider the matrix $A = \mathrm{diag}(5, 2)$ and the initial point $s = (1, 1)^\intercal$. We then have $\mathbb{T} = \{(1, 1)\}$ and $\mathcal{C}_{t_0} = \left\{ \left(t^{\log_2 5}, t\right)^\intercal \mid t \geq t_0 \right\}$. Observe that this is not an $\mathfrak{R}_0$-definable set, as the exponent $\log_2 5$ is not rational. This shows that even for diagonalizable matrices (where $\mathcal{C}_{t_0}$ has a simple form, devoid of the polynomials $Q_{i,j}$), $\mathfrak{R}_0$ might not be enough to recover definability of the orbit (in the sense of Theorem 5 below).

4 Constructing Invariants from Trajectory Cones

We now proceed to show that the trajectory cones defined in Section 3 can be used to characterise o-minimal invariants. More precisely, we show that for an LDS (A, s) with $A = PJP^{-1}$, the image under P of every trajectory cone $\mathcal{C}_{t_0}$, augmented with finitely many points from $\mathcal{O}$, is an invariant. Moreover, we show that such invariants are $\mathfrak{R}_{\exp}$-definable, and hence o-minimal. Complementing this, we show in Section 5 that *every* o-minimal invariant must contain some trajectory cone.

In what follows, let $A = PJP^{-1}$, s, as well as the real numbers $b_1, \ldots, b_d$ be defined as in Section 3.

▶ **Theorem 5.** *For every $t_0 \geq 1$, the set $P \cdot \mathcal{C}_{t_0} \cup \left\{A^n s \mid n < \log_{\rho_k} t_0\right\}$ is an $\mathfrak{R}_{\exp}$-definable invariant for the LDS (A, s).*

The intuition behind Theorem 5 is as follows. Clearly, the orbit $\mathcal{O}$ itself is always an invariant for (A, s). However, it is generally not definable in any o-minimal theory (in particular, since it has infinitely many connected components). In order to recover definability

[3] These sets are, of course, not really cones. Nevertheless, if for all i we have $b_i = 1$ and the polynomials $Q_{i,j}$ are constant, then the set $\mathcal{C}_{t_0}$ is a conical surface formed by the union of rays going from the origin through all points of $\mathbb{T}$. The initial segments of the rays, of length determined by the parameter t_0, are removed.

[4] Likewise, this set is not, strictly speaking, a straight half-line.

in $\mathfrak{R}_{\exp}$ while maintaining stability under A, the invariants constructed in Theorem 5 over-approximate the orbit by the image of the trajectory cone $\mathcal{C}_{t_0}$ under the linear transformation P. Finally, a finite set of points from $\mathcal{O}$ is added to this image of the trajectory cone, to fill in the missing points in case t_0 is too large.

The proof of Theorem 5 has several parts. First, recall that the trajectory cone itself, $\mathcal{C}_{t_0}$, is an over-approximation of the set $\left\{J^n P^{-1} s \mid n \in \mathbb{N}\right\}$. As such, clearly $\mathcal{C}_{t_0} \subseteq \mathbb{C}^d$. In comparison, the orbit can be written as $\mathcal{O} = \left\{P J^n P^{-1} s \mid n \in \mathbb{N}\right\} \subseteq \mathbb{R}^d$. We prove in the full version of the paper the following lemma, from which it follows that the entire set $P \cdot \mathcal{C}_{t_0}$ is also a subset of $\mathbb{R}^d$.

▶ **Lemma 6.** *For every $p \in \mathbb{T}$ and $t_0 \geq 1$, we have $P \cdot \mathsf{r}(p, t_0) \subseteq \mathbb{R}^d$.*

Let us simply remark here that by analysing the structure of the matrices involved in defining $P \cdot \mathsf{r}(p, t_0)$, and using the facts that the columns of P are generalised eigenvectors of A, and that conjugate pairs of eigenvalues correspond to conjugate pairs of generalised eigenvectors, it is not hard to see that the above product does indeed yield only real values. However, a formal proof of this involves fairly tedious calculations. In the full version of the paper, we proceed instead via an analytic argument.

In the second part of the proof of Theorem 5, we show that $P \cdot \mathcal{C}_{t_0}$ is stable under A. The key ingredient is the following lemma, which characterises the action of J on rays and is proved in Section 4.1.

▶ **Lemma 7.** *For every $p = (p_1, \ldots, p_k) \in \mathbb{T}$ and $t_0 \geq 1$, we have $J \cdot \mathsf{r}(p, t_0) = \mathsf{r}(Lp, \rho_k t_0)$.*

The next lemma then lifts Lemma 7 to the entire trajectory cone.

▶ **Lemma 8.** *For every $t_0 \geq 1$, we have $J \cdot \mathcal{C}_{t_0} \subseteq \mathcal{C}_{t_0}$.*

Proof. Recall that $\mathcal{C}_{t_0} = \bigcup_{p \in \mathbb{T}} \mathsf{r}(p, t_0)$. By Lemma 7 we have that $J \cdot \mathcal{C}_{t_0} = \bigcup_{p \in \mathbb{T}} \mathsf{r}(Lp, \rho_k t_0)$. Since $\rho_k > 1$, it follows that $\rho_k t_0 \geq t_0$. In addition, $p \in \mathbb{T}$ iff $Lp \in \mathbb{T}$. Hence we have that $\mathsf{r}(Lp, \rho_k t_0) \subseteq \mathsf{r}(Lp, t_0)$, from which we conclude that $J \cdot \mathcal{C}_{t_0} \subseteq \bigcup_{p \in \mathbb{T}} \mathsf{r}(Lp, t_0) = \bigcup_{p \in \mathbb{T}} \mathsf{r}(p, t_0) = \mathcal{C}_{t_0}$. ◀

The proof of Theorem 5 combines all these ingredients together and is given in subsection 4.2.

4.1 Proof of Lemma 7

Let $y = \begin{pmatrix} t^{b_1} p_1 Q_{1,1}(\log_{\rho_k} t) \\ \vdots \\ t^{b_k} p_k Q_{k,d_k}(\log_{\rho_k} t) \end{pmatrix} \in \mathsf{r}(p, t_0)$. We claim that $Jy = \begin{pmatrix} (\rho_k t)^{b_1} \lambda_1 p_1 Q_{1,1}(\log_{\rho_k}(\rho_k t)) \\ \vdots \\ (\rho_k t)^{b_k} \lambda_k p_k Q_{k,d_k}(\log_{\rho_k}(\rho_k t)) \end{pmatrix}$.
Note that since $Lp = (\lambda_1 p_1, \ldots, \lambda_k p_k)$, the above suffices to conclude the proof.

Consider a coordinate $m = (i, j)$ of Jy in block-row index, with $j < d_i$. The case of $j = d_i$ is similar and simpler. To simplify notation, we write λ, ρ, and d instead of λ_i, ρ_i, and d_i, respectively. Then we have

$$(Jy)_m = \lambda \rho t^{b_i} p_i Q_{i,j}(\log_{\rho_k} t) + t^{b_i} p_i Q_{i,j+1}(\log_{\rho_k} t)\,.$$

Recall that[5]

$$Q_{i,j}(\log_{\rho_k} t) = \sum_{c=0}^{d-j} \frac{\binom{\log_{\rho_k} t}{c}}{(\rho\lambda)^c} s'_{i,j+c}\,,$$

[5] Here, for $w \in \mathbb{R}$ and $m \in \mathbb{N}$, one defines $\binom{w}{m} = \frac{1}{m!} \prod_{i=0}^{m-1} (w - i)$, which maintains consistency with the original definition of $Q_{i,j}$ in Section 3.

with $(i, j+c)$ in block-row index. We can then write

$$(Jy)_m = \lambda\rho t^{b_i} p_i \sum_{c=0}^{d-j} \frac{\binom{\log_{\rho_k} t}{c}}{(\rho\lambda)^c} s'_{i,j+c} + t^{b_i} p_i \sum_{c=0}^{d-j-1} \frac{\binom{\log_{\rho_k} t}{c}}{(\rho\lambda)^c} s'_{i,j+c+1}\,. \tag{2}$$

We now compare this to coordinate m of our claim, namely

$$(\rho_k t)^{b_i} \lambda p_i Q_{i,j}(\log_{\rho_k}(\rho_k t)) = (\rho_k t)^{b_i} \lambda p_i \sum_{c=0}^{d-j} \frac{\binom{\log_{\rho_k}(\rho_k t)}{c}}{(\rho\lambda)^c} s'_{i,j+c}\,. \tag{3}$$

We compare the right-hand sides of Equations (2) and (3) by comparing the coefficients of $s'_{i,q}$ for $q \in \{j, \dots, d\}$ (these being the only ones that appear in the expressions). For $q = j$ we see that in (2) the number $s'_{i,j}$ occurs in the first summand only, and its coefficient is thus $\lambda\rho t^{b_i} p_i$, while in (3) it is $(\rho_k t)^{b_i} \lambda p_i = \rho_k^{b_i} t^{b_i} \lambda p_i = \rho t^{b_i} \lambda p_i$, since $b_i = \log_{\rho_k} \rho$. Thus, the coefficients are equal.

For $q > j$, write $q = j + c$ with $c \geq 1$; the coefficient at $s'_{i,j+c}$ in (2) is then

$$\lambda\rho t^{b_i} p_i \frac{\binom{\log_{\rho_k} t}{c}}{(\rho\lambda)^c} + t^{b_i} p_i \frac{\binom{\log_{\rho_k} t}{c-1}}{(\rho\lambda)^{c-1}} = \frac{t^{b_i}\rho\lambda p_i}{(\rho\lambda)^c} \left(\binom{\log_{\rho_k} t}{c} + \binom{\log_{\rho_k} t}{c-1} \right) = \frac{t^{b_i}\lambda\rho p_i}{\lambda^c} \binom{\log_{\rho_k} t + 1}{c}$$

where the last equality follows from a continuous version of Pascal's identity. Finally, by noticing that $\log_{\rho_k} t + 1 = \log_{\rho_k}(\rho_k t)$, it is easy to see that this is the same coefficient as in (3).

4.2 Proof of Theorem 5

Let $t_0 \geq 1$. By applying Lemma 6 to every $p \in \mathbb{T}$, we conclude that $P \cdot \mathcal{C}_{t_0} \subseteq \mathbb{R}^d$. It is then easy to see that $P \cdot \mathcal{C}_{t_0}$ is definable in $\mathfrak{R}_{\exp}$ (note that the only reason the set $\mathcal{C}_{t_0}$ might fail to be $\mathfrak{R}_{\exp}$-definable is that the underlying domain should be $\mathbb{R}$ and not $\mathbb{C}$).

Next, by Lemma 8 we have that $J \cdot \mathcal{C}_{t_0} \subseteq \mathcal{C}_{t_0}$. Applying P from the left, we get $PJ \cdot \mathcal{C}_{t_0} \subseteq P \cdot \mathcal{C}_{t_0}$. Thus, we have $AP \cdot \mathcal{C}_{t_0} = PJP^{-1}P \cdot \mathcal{C}_{t_0} = PJ \cdot \mathcal{C}_{t_0} \subseteq P \cdot \mathcal{C}_{t_0}$.

Finally, observe that $\{A^n s \mid n \geq \log_{\rho_k} t_0\} \subseteq P \cdot \mathcal{C}_{t_0}$. Since any finite subset of $\mathcal{O}$ can be described in $\mathfrak{R}_0$, we conclude that the set $\{A^n s \mid n < \log_{\rho_k} t_0\} \cup P \cdot \mathcal{C}_{t_0}$ is an $\mathfrak{R}_{\exp}$-definable invariant for (A, s).

5 O-Minimal Invariants Must Contain Trajectory Cones

In this section we consider invariants definable in o-minimal extensions of $\mathfrak{R}_{\exp}$. Fix such an extension $\mathfrak{R}$ for the remainder of this section.

▶ **Theorem 9.** *Consider an $\mathfrak{R}$-definable invariant $\mathcal{I}$ for the LDS (A, s). Then there exists $t_0 \geq 1$ such that $P \cdot \mathcal{C}_{t_0} \subseteq \mathcal{I}$.*

To prove Theorem 9, we begin by making following claims of increasing strength:

▶ **Claim 1.** *For every $p \in \mathbb{T}$ there exists $t_0 \geq 1$ such that $P \cdot \mathsf{r}(p, t_0) \subseteq \mathcal{I}$ or $P \cdot \mathsf{r}(p, t_0) \cap \mathcal{I} = \emptyset$.*

▶ **Claim 2.** *For every $p \in \mathbb{T}$ there exists $t_0 \geq 1$ such that $P \cdot \mathsf{r}(p, t_0) \subseteq \mathcal{I}$.*

▶ **Claim 3.** *There exists $t_0 \geq 1$ such that for every $p \in \mathbb{T}$ we have $P \cdot \mathsf{r}(p, t_0) \subseteq \mathcal{I}$.*

Proof of Claim 1. Fix $p \in \mathbb{T}$. Observe that by Lemma 6, $P \cdot \mathsf{r}(p,1)$ is $\mathfrak{R}$-definable. Further note that $P \cdot \mathsf{r}(p,1)$ is of dimension 1 (as it is homeomorphic to $[1,\infty)$). Thus, the dimension of $P \cdot \mathsf{r}(p,1) \cap \mathcal{I}$ is at most 1, so its cell decomposition contains finitely many connected components of dimensions 0 or 1. In particular, either one component is unbounded, in which case there exists a t_0 such that $P \cdot \mathsf{r}(p,t_0) \subseteq \mathcal{I}$, or all the components are bounded, in which case there exists a t_0 such that $P \cdot \mathsf{r}(p,t_0) \cap \mathcal{I} = \emptyset$. ◀

Before proceeding to Claim 2, we prove an auxiliary lemma, which is an adaptation of a similar result from [13]. For a set X, we write $\overline{X}$ to refer to the topological closure of X. We use the usual topology on $\mathbb{R}^n$, $\mathbb{C}^n$, and the (usual) subspace topology on their subsets.

▶ **Lemma 10.** *Let $S, F \subseteq \mathbb{T}$ be $\mathfrak{R}$-definable[6] sets such that $\overline{S} = \overline{F} = \mathbb{T}$. Then $F \cap S \neq \emptyset$.*

Proof. We start by stating two properties of the dimension of a definable set in an o-minimal theory $\mathfrak{R}$. First, for any $\mathfrak{R}$-definable set $X \subseteq \mathbb{R}^n$ we have $\dim(X) = \dim(\overline{X})$ [12, Chapter 4, Theorem 1.8]. Secondly, if $X \subseteq Y$ are $\mathfrak{R}$-definable subsets of $\mathbb{R}^n$ that have the same dimension, then X has non-empty interior in Y [12, Chapter 4, Corollary 1.9]. In the situation at hand, since $\dim(F) = \dim(\overline{F})$, it follows that F has non-empty interior with respect to the subspace topology on $\overline{F} = \overline{S}$. But then S is dense in $\overline{S}$ while F has non-empty interior in $\overline{S}$, and thus $S \cap F \neq \emptyset$. ◀

Proof of Claim 2. We strengthen Claim 1. Assume by way of contradiction that there exist $p \in \mathbb{T}$ and $t_0 \in \mathbb{R}$ such that $P \cdot \mathsf{r}(p,t_0) \cap \mathcal{I} = \emptyset$, and consider $J^{-1} \cdot \mathsf{r}(p,t_0)$. Let $q \in \mathbb{T}$ be $L^{-1}p = (\frac{p_1}{\lambda_1}, \ldots, \frac{p_k}{\lambda_k})$ and let $t_1 = \frac{t_0}{\rho_k}$. Then $p = Lq$ and $t_0 = \rho_k t_1$ and, by Lemma 7, $J\mathsf{r}(q,t_1) = \mathsf{r}(Lq, \rho_k t_1) = \mathsf{r}(p,t_0)$. Since J is invertible, we conclude that $J^{-1}\mathsf{r}(p,t_0) = \mathsf{r}(q,t_1)$.

We now claim that $P \cdot \mathsf{r}(q,t_1) \cap \mathcal{I} = \emptyset$. Recall that $P \cdot \mathsf{r}(p,t_0) \cap \mathcal{I} = \emptyset$. Applying $A^{-1} = PJ^{-1}P^{-1}$, we have by the above that $P \cdot \mathsf{r}(q,t_1) \cap A^{-1}\mathcal{I} = \emptyset$. Since $A\mathcal{I} \subseteq \mathcal{I}$, then $\mathcal{I} \subseteq A^{-1}\mathcal{I}$, so we have $P \cdot \mathsf{r}(q,t_1) \cap \mathcal{I} \subseteq P \cdot \mathsf{r}(q,t_1) \cap A^{-1}\mathcal{I} = \emptyset$.

Recall that, following the discussion in section 3, we have $\rho_k > 1$. This implies $t_1 \leq t_0$ and $\mathsf{r}(q,t_0) \subseteq \mathsf{r}(q,t_1)$, so in particular $P \cdot \mathsf{r}(q,t_0) \cap \mathcal{I} = \emptyset$. Thus, assuming $P \cdot \mathsf{r}(p,t_0) \cap \mathcal{I} = \emptyset$, we have just proved that $P \cdot \mathsf{r}(L^{-1}p,t_0) \cap \mathcal{I} = \emptyset$; repeating this argument, we get that for every $n \in \mathbb{N}$, the point $s = L^{-n}p$ satisfies $P \cdot \mathsf{r}(s,t_0) \cap \mathcal{I} = \emptyset$.

Let $S = \{L^{-n}p \mid n \in \mathbb{N}\}$. Then S is dense in $\mathbb{T}$, since the group of multiplicative relations defined by the eigenvalues of L^{-1} is the same as the one defined by those of L. Define $S' = \{s \in \mathbb{T} \mid P \cdot \mathsf{r}(s,t_0) \cap \mathcal{I} = \emptyset\}$. Then S' is $\mathfrak{R}$-definable, and we have $S \subseteq S' \subseteq \mathbb{T}$. Moreover, $\overline{S} = \mathbb{T}$, so $\overline{S'} = \mathbb{T}$.

We now prove that, in fact, $S' = \mathbb{T}$. Assuming (again by way of contradiction) that there exists $q \in \mathbb{T} \setminus S'$, then by the definition of S' we have $P \cdot \mathsf{r}(q,t_0) \cap \mathcal{I} \neq \emptyset$. It follows that for every $n \in \mathbb{N}$, the point $q' = L^n q$ also satisfies $P \cdot \mathsf{r}(q',t_0) \cap \mathcal{I} \neq \emptyset$. Define $F = \{L^n q \mid n \in \mathbb{N}\}$, then F is dense in $\mathbb{T}$. But then the set $F' = \{q \in \mathbb{T} \mid P \cdot \mathsf{r}(q,t_0) \cap \mathcal{I} \neq \emptyset\}$ satisfies $F \subseteq F' \subseteq \mathbb{T}$ and $\overline{F'} = \mathbb{T}$. Now the sets S' and F' are both definable in $\mathfrak{R}$, and the topological closure of each of them is $\mathbb{T}$. It follows from Lemma 10 that $F' \cap S' \neq \emptyset$, which is clearly a contradiction. Therefore, there is no $q \in \mathbb{T} \setminus S'$; that is, $S' = \mathbb{T}$.

From this, however, it follows that $P \cdot \mathcal{C}_{t_0} \cap \mathcal{I} = \emptyset$, which is again a contradiction, since $P \cdot \mathcal{C}_{t_0} \cap \mathcal{O} \neq \emptyset$ and $\mathcal{O} \subseteq \mathcal{I}$, so we are done. ◀

Proof of Claim 3. Consider the function $f\colon \mathbb{T} \to \mathbb{R}$ defined by $f(p) = \inf\{t \in \mathbb{R} \mid P \cdot \mathsf{r}(p,t) \subseteq \mathcal{I}\}$. By Claim 2 this function is well-defined. Since $P \cdot \mathsf{r}(p,t)$ is $\mathfrak{R}$-definable, then so is f.

[6] Recall that, in order to reason about $\mathbb{T} \subseteq \mathbb{C}^k$ in $\mathfrak{R}$, we identify $\mathbb{C}$ with $\mathbb{R}^2$.

Moreover, its graph $\Gamma(f)$ has finitely many connected components, and the same dimension as $\mathbb{T}$. Thus, there exists an open set $K \subseteq \mathbb{T}$ (in the induced topology on $\mathbb{T}$) such that f is continuous on K. Furthermore, K is homeomorphic to $(0,1)^m$ for some $0 \leq m \leq k$, and thus we can find sets $K'' \subseteq K' \subseteq K$ such that K'' is open, and K' is closed.[7] Since f is continuous on K, it attains a maximum on K'. Consider the set $\{L^n \cdot K'' \mid n \in \mathbb{N}\}$. By the density of $\{L^n \mid n \in \mathbb{N}\}$ in $\mathbb{T}$, this is an open cover of $\mathbb{T}$, and hence there is a finite subcover $\{L^{n_1}K'', \ldots, L^{n_m}K''\}$. Since $K'' \subseteq K'$, it follows that $\{L^{n_1}K', \ldots, L^{n_m}K'\}$ is a finite closed cover of $\mathbb{T}$.

We now show that, for all $p \in \mathbb{T}$, we have $f(Lp) \leq \rho_k \cdot f(p)$. Indeed, consider any $p \in \mathbb{T}$ and $t > 0$ such that $P \cdot \mathsf{r}(p,t) \subseteq \mathcal{I}$. Applying $A = PJP^{-1}$, we get $PJ \cdot \mathsf{r}(p,t) \subseteq A\mathcal{I} \subseteq \mathcal{I}$. By Lemma 7, $J \cdot \mathsf{r}(p,t) = \mathsf{r}(Lp, \rho_k t)$, so we can conclude that $P \cdot \mathsf{r}(Lp, \rho_k t) \subseteq \mathcal{I}$. This means that $P \cdot \mathsf{r}(p,t) \subseteq \mathcal{I}$ implies $P \cdot \mathsf{r}(Lp, \rho_k t) \subseteq \mathcal{I}$; therefore, $f(Lp) \leq \rho_k \cdot f(p)$.

Now denote $s_0 = \max_{p \in K'} f(p)$. Then for every $1 \leq i \leq m$ we have $\max_{p \in L^{n_i}K'} f(p) \leq \rho_k^{n_i} s_0$; so $f(p)$ is indeed bounded on $\mathbb{T}$. ◀

Finally, we conclude from Claim 3 that there exists $t_0 \geq 1$ such that $P \cdot \mathcal{C}_{t_0} \subseteq \mathcal{I}$. This completes the proof of Theorem 9.

6 Deciding the Existence of O-Minimal Invariants

We now turn to the algorithmic aspects of invariants and present our two main results, Theorems 12 and 13.

Let $\mathfrak{R}$ be either $\mathfrak{R}_0$ or $\mathfrak{R}_{\exp}$. We consider the following problem: given an LDS (A, s), with $A \in \mathbb{Q}^{d \times d}$ and $s \in \mathbb{Q}^d$, and given an $\mathfrak{R}$-definable halting set $F \subseteq \mathbb{R}^d$, we wish to decide whether there exists an o-minimal invariant $\mathcal{I}$ for (A, s) that avoids F, and to compute such an invariant if it exists. We term this question the *O-Minimal Invariant Synthesis Problem for $\mathfrak{R}$-Definable Halting Sets*.

The following is a consequence of Theorems 5 and 9.

▶ **Lemma 11.** *Let (A, s) and $\mathfrak{R}$ be as above, and let F be $\mathfrak{R}$-definable. Then there exists an o-minimal invariant $\mathcal{I}$ for (A, s) that avoids F iff there is some $t_0 \geq 1$ such that $P \cdot \mathcal{C}_{t_0} \cap F = \emptyset$ and such that $A^n s \notin F$ for every $0 \leq n \leq \log_{\rho_k} t_0$.*

Proof. By Theorem 9, if an o-minimal invariant $\mathcal{I}$ for (A, s) exists, then there exists $t_0 \geq 1$ such that $P \cdot \mathcal{C}_{t_0} \subseteq \mathcal{I}$. Moreover, $\mathcal{I} \cap F = \emptyset$ implies $\mathcal{O} \cap F = \emptyset$, so that $A^n s \notin F$ for every $n \in \mathbb{N}$, and in particular for $0 \leq n \leq \log_{\rho_k} t_0$.

Conversely, let there be $t_0 \geq 1$ such that $P \cdot \mathcal{C}_{t_0} \cap F = \emptyset$ and, for every $0 \leq n \leq \log_{\rho_k} t_0$, it holds that $A^n s \notin F$. Let $t_0' \in \mathbb{Q}$ be such that $t_0' \geq t_0$. By Theorem 5, the set $\mathcal{I} = P \cdot \mathcal{C}_{t_0'} \cup \left\{A^n s \mid 0 \leq n \leq \log_{\rho_k} t_0'\right\}$ is an $\mathfrak{R}_{\exp}$-definable invariant that avoids F. ◀

Observe that the formula $\exists t_0 \geq 1 : P \cdot \mathcal{C}_{t_0} \cap F = \emptyset$ is a sentence in $\mathfrak{R}_{\exp}$, and by Lemma 11, deciding the existence of an invariant amounts to determining the truth value of this sentence.

Decidability for $\mathfrak{R}_{\exp}$-definable halting sets assuming Schanuel's conjecture. Applying Theorem 5, we note that an invariant for (A, s) that avoids F—if one exists—can always be defined in $\mathfrak{R}_{\exp}$.

[7] In case $m = 0$, the proof actually follows immediately from Claim 2, since $\mathbb{T}$ is finite.

▶ **Theorem 12.** *The O-Minimal Invariant Synthesis Problem for $\mathfrak{R}_{\exp}$-Definable Halting Sets is decidable, assuming Schanuel's conjecture. Moreover, in positive instances, we can explicitly define such an invariant in $\mathfrak{R}_{\exp}$.*

Proof. Assume Schanuel's conjecture. Then by [18], the theory $\mathfrak{R}_{\exp}$ is decidable. Thus we can decide whether there exists $t_0 \geq 1$ such that $P \cdot \mathcal{C}_{t_0} \cap F = \emptyset$. If the sentence is false, then by Lemma 11 there is no invariant, and we are done. If the sentence is true, however, it still remains to check whether $A^n s \notin F$ for every $0 \leq n \leq \log_{\rho_k} t_0$. While we can decide whether $A^n s \in F$ for a fixed n, observe that we do not have an *a priori* bound on t_0. Hence we proceed as follows: For every $n \in 1, 2, \ldots$, check both whether $A^n s \in F$ and, for $t_0 = \rho_k^n$, whether $PC_{t_0} \cap F = \emptyset$. In case $A^n s \in F$, then clearly there is no invariant, since $\mathcal{O} \cap F \neq \emptyset$, and we are done. On the other hand, if $PC_{t_0} \cap F = \emptyset$, then return the $\mathfrak{R}_{\exp}$-definable invariant as per Lemma 11.

We claim that the above procedure always halts. Indeed, we know that there exists t_0 for which $P \cdot \mathcal{C}_{t_0} \cap F = \emptyset$. Thus, either for some $n < \log_{\rho_k} t_0$, it holds that $A^n s \in F$, in which case there is no invariant and we halt when we reach n, or we proceed until we reach $n \geq \log_{\rho_k} t_0$, in which case we halt and return the invariant. ◀

▶ Remark. It is interesting to note that, should Schanuel's conjecture turn out to be false, the above procedure could still never return a 'wrong' invariant. The worst that could happen is that decidability of $\mathfrak{R}_{\exp}$ fails in that the putative algorithm of [18] simply never halts, so no verdict is ever returned.

Unconditional decidability for semi-algebraic halting sets.

▶ **Theorem 13.** *The O-Minimal Invariant Synthesis Problem for Semi-Algebraic Halting Sets is decidable. Moreover, in positive instances, we can explicitly define such an invariant in $\mathfrak{R}_{\exp}$.*

By Lemma 11, in order to prove Theorem 13, it is enough to decide the truth value of the $\mathfrak{R}_{\exp}$-sentence $\exists t_0 \geq 1 : P \cdot \mathcal{C}_{t_0} \cap F = \emptyset$. Indeed, as $A^n s \in \mathbb{Q}^d$, one can always check unconditionally whether for a given n the vector $A^n s$ belongs to the semi-algebraic set F. The algorithm is then otherwise the same as that presented in the proof of Theorem 12. The proof of Theorem 13 therefore boils down to the following lemma.

▶ **Lemma 14.** *For F a semi-algebraic set, it is decidable whether there exists $t_0 \geq 1$ such that $P \cdot \mathcal{C}_{t_0} \cap F = \emptyset$.*

Our key tool is the following celebrated result from transcendental number theory:

▶ **Theorem 15** (Baker's theorem [1])**.** *Let $\alpha_1, \ldots, \alpha_m \in \mathbb{C}$ be algebraic numbers different from 0 and let $b_1, \ldots, b_m \in \mathbb{Z}$ be integers. Write $\Lambda = b_1 \log \alpha_1 + \ldots + b_m \log \alpha_m$. There exists a number C effectively computable from $b_1, \ldots, b_m, \alpha_1, \ldots, \alpha_m$ such that if $\Lambda \neq 0$ then $|\Lambda| > H^{-C}$, where H is the maximum height of $\alpha_1, \ldots, \alpha_m$.*

As in Section 3, we assume that $\rho_k > 1$ (with the cases of $\rho_k < 1$ and $\rho_k = 1$ being analogous and easier, respectively). Recall that the subgroup $\mathbb{T}$ of the torus defined by the multiplicative relations of the eigenvalues of A is a semi-algebraic set. Write $\vec{\tau}(t) = (t^{b_1} Q_{1,1}(\log_{\rho_k} t), \ldots, t^{b_k} Q_{k,d_k}(\log_{\rho_k} t))$, and consider the set

$$U = \left\{ (z_1, \ldots, z_d)^\intercal \in \mathbb{C}^d \mid \forall (p_1, \ldots, p_d) \in \mathbb{T},\ P\,(z_1 p_1, \ldots, z_d p_d)^\intercal \in \mathbb{R}^d \setminus F \right\}.$$

It is enough to decide whether there exists $t_0 \geq 1$ such that for all $t \geq t_0$, $\vec{\tau}(t) \in U$.

Observe that U is a semi-algebraic set (see Remark on Page 4). In the full version of the paper, we reduce our analysis to the case where U is of the form $\bigwedge_{l=1}^{m} R_l(u_1, \ldots, u_d, v_1, \ldots, v_d) \sim_l 0$. Here, for every $1 \leq l \leq m$, $\sim_l \in \{>, =\}$ and R_l is a polynomial with integer coefficients in variables $u_1, \ldots, u_d, v_1, \ldots, v_d$; for each i, the variables u_i and v_i represent $\operatorname{Re} z_i$ and $\operatorname{Im} z_i$, the real and imaginary parts of z_i, respectively.

Thus, we now need to decide whether we can find $t_0 \geq 1$ such that for every $t \geq t_0$ it holds that $R_l(\vec{\tau}(t)) \sim_l 0$ for every $1 \leq l \leq m$. Fix $1 \leq l \leq m$. Recall that we consider every vector in $\mathbb{C}^d$ as a vector in $\mathbb{R}^{2d}$; thus, the polynomial R_l has the form

$$\sum_i a_i \cdot u_1^{n'_{i,1}} \cdot \ldots \cdot u_d^{n'_{i,d}} \cdot v_1^{n''_{i,1}} \cdot \ldots \cdot v_d^{n''_{i,d}},$$

with $a_i \in \mathbb{Z}$ and $n'_{i,s}, n''_{i,s} \in \mathbb{Z}_{\geq 0}$. Therefore, $R_l(\vec{\tau}(t))$ is the sum of terms of the form

$$a_i \cdot t^{(n'_{i,1}+n''_{i,1})\cdot b_1 + \ldots + (n'_{i,d}+n''_{i,d})\cdot b_k} \cdot (\operatorname{Re} Q_{1,1}(\log_{\rho_k} t))^{n'_{i,1}} \cdot \ldots \cdot (\operatorname{Re} Q_{k,d_k}(\log_{\rho_k} t))^{n'_{i,k}} \cdot (\operatorname{Im} Q_{1,1}(\log_{\rho_k} t))^{n''_{i,1}} \cdot \ldots \cdot (\operatorname{Im} Q_{k,d_k}(\log_{\rho_k} t))^{n''_{i,k}}$$

where $Q_{i,j}(\cdot)$, as above, are polynomials from the definition of trajectory cones. Note that all $Q_{i,j}$ are only evaluated at real points, and hence it is easy for us to refer to $\operatorname{Re} Q_{i,j}$ and $\operatorname{Im} Q_{i,j}$; these are polynomials in one real variable with real algebraic coefficients. We rewrite $R_l(\vec{\tau}(t))$ in the form

$$\sum_i t^{n_{i,1}\cdot b_1 + \ldots + n_{i,k}\cdot b_k} \cdot f_i(\log_{\rho_k} t)$$

where each $f_i(\cdot)$ is also a polynomial with real algebraic coefficients, and $b_1, \ldots, b_k$ are distinct logarithms of the moduli of the eigenvalues of A. We can compute all these polynomials f_i, eliminating from the sum all terms that have $f_i \equiv 0$.

Observe that $R_l(\vec{\tau}(t))$ is a function of a single variable $t > 0$. In order to reason about the sign of this expression as $t \to \infty$, we need to determine its leading term. To that end, we first need to decide for every $i \neq j$ whether the two new exponents $n_{i,1}b_1 + \ldots + n_{i,k}b_k$ and $n_{j,1}b_1 + \ldots + n_{j,k}b_k$ are equal and, if not, which is larger. (If the exponents are equal, we aggregate the polynomials f_i and f_j accordingly.) By rearranging the terms, it's enough to decide whether $n_1b_1 + \ldots + n_kb_k > 0$ for some $n_1, \ldots, n_k \in \mathbb{Z}$. Recall that $b_j = \log_{\rho_k} \rho_j = \log \rho_j / \log \rho_k$ where log denotes the natural logarithm. By Baker's theorem, there exists an effectively computable $\epsilon > 0$ such that either $n_1b_1 + \ldots + n_kb_k = 0$, or $|n_1b_1 + \ldots + n_kb_k| > \epsilon$.

We now proceed by computing an approximation Δ of $n_1b_1 + \ldots + n_kb_k$ with additive error at most $\frac{\epsilon}{3}$. This is easily done, as we are dealing with computable quantities. We then have that $\Delta \in [-\frac{\epsilon}{3}, \frac{\epsilon}{3}]$ iff $n_1b_1 + \ldots + n_kb_k = 0$, and otherwise we have $\operatorname{sign}(\Delta) = \operatorname{sign}(n_1b_1 + \ldots + n_kb_k)$. Thus we can sort the exponents $n_{i,1} \cdot b_1 + \ldots + n_{i,k} \cdot b_k$ in descending order and, using the same procedure, compare each of them to 0.

Now consider the term that has the largest exponent, m; suppose this term is $t^m \cdot f_i(\log_{\rho_k} t)$. Then the sign of $R_l(\vec{\tau}(t))$ as $t \to \infty$ is determined by the sign of the leading term of the polynomial $f_i(\cdot)$; only if the sum is empty can the sign of $R_l(\vec{\tau}(t))$ be 0 for all sufficiently large t.

The argument above shows that we can compute the leading terms of the expressions $R_l(\vec{\tau}(t))$ and decide whether the conjunction $\bigwedge_{l=1}^{m} R_l \sim_l 0$ holds for all $t \geq t_0$ starting from some t_0. This completes the proof.

References

1 Alan Baker and Gisbert Wüstholz. Logarithmic forms and group varieties. *J. reine angew. Math*, 442(19-62):3, 1993.

2 Amir M. Ben-Amram and Samir Genaim. Ranking functions for linear-constraint loops. *J. ACM*, 61(4):26:1–26:55, 2014.

3 Amir M. Ben-Amram, Samir Genaim, and Abu Naser Masud. On the termination of integer loops. *ACM Trans. Program. Lang. Syst.*, 34(4):16:1–16:24, 2012.

4 Mark Braverman. Termination of integer linear programs. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, pages 372–385, 2006.

5 John W.S. Cassels. *An Introduction to Diophantine Approximation*. Cambridge University Press, 1965.

6 Ventsislav Chonev, Joël Ouaknine, and James Worrell. The Orbit Problem in higher dimensions. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 941–950, 2013.

7 Ventsislav Chonev, Joël Ouaknine, and James Worrell. The polyhedron-hitting problem. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 940–956, 2015.

8 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the complexity of the orbit problem. *J. ACM*, 63(3):23:1–23:18, 2016.

9 Michael Colón, Sriram Sankaranarayanan, and Henny Sipma. Linear invariant generation using non-linear constraint solving. In *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, pages 420–432, 2003.

10 Patrick Cousot. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In *Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005, Proceedings*, pages 1–24, 2005.

11 Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978*, pages 84–96, 1978.

12 L. P. D. van den Dries. *Tame Topology and O-minimal Structures*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1998.

13 Nathanaël Fijalkow, Pierre Ohlmann, Joël Ouaknine, Amaury Pouly, and James Worrell. Semialgebraic invariant synthesis for the kannan-lipton orbit problem. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 29:1–29:13, 2017.

14 Ashutosh Gupta, Thomas A. Henzinger, Rupak Majumdar, Andrey Rybalchenko, and Ru-Gang Xu. Proving non-termination. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 147–158, 2008.

15 Ravindran Kannan and Richard J. Lipton. The orbit problem is decidable. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, pages 252–261, 1980.

16 Ravindran Kannan and Richard J. Lipton. Polynomial-time algorithm for the orbit problem. *J. ACM*, 33(4):808–821, 1986.

17 Zachary Kincaid, John Cyphert, Jason Breck, and Thomas W. Reps. Non-linear reasoning for invariant synthesis. *PACMPL*, 2(POPL):54:1–54:33, 2018.

18 Angus Macintyre and Alex J. Wilkie. On the decidability of the real exponential field. In Piergiorgio Odifreddi, editor, *Kreiseliana. About and Around Georg Kreisel*, pages 441–467. A K Peters, 1996.

19 David W Masser. Linear relations on algebraic groups. *New Advances in Transcendence Theory*, pages 248–262, 1988.

20 M. Mignotte, T. Shorey, and R. Tijdeman. The distance between terms of an algebraic recurrence sequence. *J. für die reine und angewandte Math.*, 349, 1984.

21 Joël Ouaknine, João Sousa Pinto, and James Worrell. On termination of integer linear loops. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 957–969, 2015.

22 Joël Ouaknine and James Worrell. On the positivity problem for simple linear recurrence sequences,. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 318–329, 2014.

23 Joël Ouaknine and James Worrell. Positivity problems for low-order linear recurrence sequences. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 366–379, 2014.

24 Joël Ouaknine and James Worrell. Ultimate positivity is decidable for simple linear recurrence sequences. In *International Colloquium on Automata, Languages, and Programming*, pages 330–341. Springer, 2014.

25 Joël Ouaknine and James Worrell. On linear recurrence sequences and loop termination. *SIGLOG News*, 2(2):4–13, 2015.

26 Enric Rodríguez-Carbonell and Deepak Kapur. An abstract interpretation approach for automatic generation of polynomial invariants. In *Static Analysis, 11th International Symposium, SAS 2004, Verona, Italy, August 26-28, 2004, Proceedings*, pages 280–295, 2004.

27 Enric Rodríguez-Carbonell and Deepak Kapur. Generating all polynomial invariants in simple loops. *J. Symb. Comput.*, 42(4):443–476, 2007.

28 Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. Non-linear loop invariant generation using gröbner bases. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, pages 318–329, 2004.

29 T. Tao. *Structure and randomness: pages from year one of a mathematical blog.* American Mathematical Society, 2008.

30 Alfred Tarski. A decision method for elementary algebra and geometry. *Bulletin of the American Mathematical Society*, 59, 1951.

31 Ashish Tiwari. Termination of linear programs. In *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*, pages 70–82, 2004.

32 N. K. Vereshchagin. The problem of appearance of a zero in a linear recurrence sequence (in Russian). *Mat. Zametki*, 38(2), 1985.

33 A. J. Wilkie. Model completeness results for expansions of the ordered field of real numbers by restricted pfaffian functions and the exponential function. *Journal of the American Mathematical Society*, 9(4):1051–1094, 1996.

34 Bican Xia and Zhihai Zhang. Termination of linear programs with nonlinear constraints. *J. Symb. Comput.*, 45(11):1234–1249, 2010.

Topological Sorting with Regular Constraints

Antoine Amarilli
LTCI, Télécom ParisTech, Université Paris-Saclay

Charles Paperman
Université de Lille

Abstract

We introduce the *constrained topological sorting problem* (CTS): given a regular language K and a directed acyclic graph G with labeled vertices, determine if G has a topological sort that forms a word in K. This natural problem applies to several settings, e.g., scheduling with costs or verifying concurrent programs. We consider the problem CTS[K] where the target language K is fixed, and study its complexity depending on K. We show that CTS[K] is tractable when K falls in several language families, e.g., *unions of monomials*, which can be used for pattern matching. However, we show that CTS[K] is NP-hard for $K = (ab)^*$ and introduce a *shuffle reduction* technique to show hardness for more languages. We also study the special case of the *constrained shuffle problem* (CSh), where the input graph is a disjoint union of strings, and show that CSh[K] is additionally tractable when K is a group language or a union of district group monomials. We conjecture that a dichotomy should hold on the complexity of CTS[K] or CSh[K] depending on K, and substantiate this by proving a coarser dichotomy under a different problem phrasing which ensures that tractable languages are closed under common operators.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Topological sorting, shuffle problem, regular language

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.115

Related Version A full version of the paper is available at [5], `https://arxiv.org/abs/1707.04310`.

Acknowledgements We thank Michaël Cadilhac and Pierre McKenzie for their fruitful insights.

1 Introduction

Many scheduling or ordering problems amount to computing a *topological sort* of a directed acyclic graph (DAG), i.e., a totally ordered sequence of the vertices that is compatible with the edge relation: when we enumerate a vertex, all its predecessors must have been enumerated first. However, in some settings, we need a topological sort satisfying additional constraints that cannot be expressed as edges. We formalize this problem as follows: the vertices of the DAG are labeled with some symbols from a finite alphabet A, and we want to find a topological sort that falls into a specific regular language. We call this the *constrained topological sort problem*, or CTS. For instance, if we fix the language $K = ab^*c$, and consider the example DAGs of Figure 1, then G_1 and G_2 have a topological sort that falls in K.

CTS relates to many applications. For instance, many *scheduling* applications use a dependency graph [1] of tasks, and it is often useful to express other constraints, e.g., some tasks must be performed by specific workers and we should not assign more than p successive tasks to the same worker. We can express this as a CTS-problem: label each task by the worker which can perform it, and consider the target regular language K containing all words where the same symbol is not repeated more than p times. In *concurrency* applications,

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 115; pp. 115:1–115:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

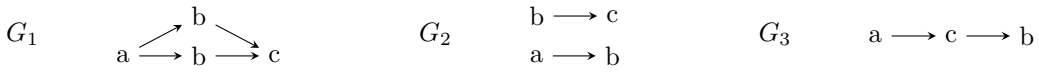


Figure 1 Example labeled DAGs on the alphabet $A = \{a, b, c\}$

we may consider a program with multiple threads, and want to verify that there is no linearization of its instructions that exhibits some unsafe behavior, e.g., executing a read before a write. To search for such a linearization, we can label each instruction with its type, and consider CTS with a target language describing the behavior that we wish to detect. CTS can also be used in uncertain data management tasks, to reason about the possible answers of aggregate queries on uncertain ordered data [4]. It can also be equivalently phrased in the language of partial order theory: seeing the labeled DAG as a *labeled partial order* $<$, we ask if some *linear extension* achieves a word in K.

We thus believe that the CTS-problem is useful, and natural, but we are not aware of previous work studying it, except for a special case called the *shuffle problem*. This problem deals with the *interleaving* of strings, as studied, e.g., in concurrent programming languages [18, 21], computational biology [17], and formal languages [10, 8, 26]. Specifically, we are given a tuple of strings, and we must decide if they have some interleaving that falls in the target language K. This problem was known to be NP-complete [19, 32, 16] when the target language K is given as input (in addition to the tuple of strings), even when K consists of just one target string. To rephrase this shuffle problem in our context, we call *constrained shuffle problem* (CSh) the special case of CTS where we require input DAGs to be a union of directed path graphs (corresponding to the strings).

Our goal in this paper is to study the complexity of CTS and CSh. We assume that the target regular language K is fixed, and call CTS[K] and CSh[K] the corresponding problems, whose complexity is only a function of the input DAG (labeled on the alphabet A of K). Our central question is: *for which regular languages K are the problems* CTS[K] *or* CSh[K] *tractable?* More precisely, for each of these problems, we conjecture a dichotomy on K: the problem is either in NL or it is NP-complete. However, the tractability boundary is challenging to chart out, and we have not been able to prove these conjectures in full generality. In this paper, we present the results that we have obtained towards this end.

Paper structure. We formally define the CTS and CSh problems in Section 2 and state the conjecture. We then show the following results:

- In Section 3, we present our hardness results. We recall the results of [32] on the shuffle problem, and present a general *shuffle reduction* technique to show hardness for more languages. We use it in particular to show that CSh[$(ab)^*$], hence CTS[$(ab)^*$], are NP-hard, and extend this to several other languages.
- In Section 4, we present tractability results. We show that CTS[K], hence CSh[K], is in non-deterministic logspace (NL) when K is a union of *monomial languages*, i.e., of languages of the form $A_1^* a_1 \cdots A_{n-1}^* a_{n-1} A_n^*$, with the a_i being letters and the A_i being subalphabets. Such languages can be used for applications such as pattern matching, e.g., with the language $A^* u A^*$ for a fixed pattern $u \in A^*$. We also show tractability for other languages that are not of this form, e.g. $(ab)^* + A^* aa A^*$ and variants thereof, using different techniques such as Dilworth's theorem [9].
- In Section 5, we use our hardness and tractability results to show a coarser dichotomy result. Specifically, we give an alternative phrasing of the CTS and CSh problems using

semiautomata and DAGs with *multi-letter* labels: this amounts to closing the tractable languages under intersection, inverse morphism, complement, and quotients. In this phrasing, when the semiautomaton is counter-free, we can show that the problems are either in NL or NP-complete. This dichotomy is effective, i.e., the criterion on the semiautomaton is decidable, and it turns out to be the same for CTS and CSh.

- In Section 6, we focus on the constrained shuffle problem, and lift the counter-free assumption of the previous section. We show that CSh$[K]$ is tractable when K is a *group language* or more generally a union of *district group monomials*. This tractability result is the main technical contribution of the paper, with a rather involved proof. It implies, e.g., that the following problem is in NL for any fixed finite group H: given $g \in H$ and words $w_1, \ldots, w_n$ of elements of H, decide whether there is an interleaving of the w_i which evaluates to g according to the group operation.

2 Problem Statement and Main Results

We give some preliminaries and define the two problems that we study. We fix a finite alphabet A, and call A^* the set of all finite words on A. For $w \in A^*$, we write $|w|$ for the *length* of w, and write $|w|_a$ for the number of occurrences of $a \in A$ in w. We denote the empty word by ϵ. A *labeled DAG* on the alphabet A, or A-*DAG*, is a triple $G = (V, E, \lambda)$ where (V, E) is a directed acyclic graph with vertex set $V = \{1, \ldots, n\}$ and edge set $E \subseteq V \times V$, and where $\lambda : V \to A$ is a function giving a label in A to each vertex in V. For $u \neq v$ in V, we say that u is an *ancestor* of v if there is a directed path from u to v in G, we say that u is a *descendant* of v if v is an ancestor of u, and otherwise we call u and v *incomparable*. A *topological sort* of G is a bijective function σ from $\{1, \ldots, n\}$ to V such that, for all $(u, v) \in E$, we have $\sigma^{-1}(u) < \sigma^{-1}(v)$. The word *achieved* by σ is $\lambda(\sigma) := \lambda(\sigma(1)) \cdots \lambda(\sigma(n)) \in A^*$.

The *constrained topological sort problem* CTS$[K]$ for a fixed language $K \subseteq A^*$ (described, e.g., by a regular expression) is defined as follows: given an A-DAG G, determine if there is a topological sort σ of G such that $\lambda(\sigma) \in K$ (in which case we say that σ *achieves* K).

We now define the *constrained shuffle problem* (CSh). Given two words $u, v \in A^*$, the *shuffle* [32] of u and v, written $u \sqcup\!\sqcup v$, is the set of words that can be obtained by interleaving them. Formally, a word $w \in A^*$ is in $u \sqcup\!\sqcup v$ iff there is a partition $P \sqcup Q$ of $\{1, \ldots, |w|\}$ such that $w_P = u$ and $w_Q = v$, where w_P denotes the sub-word of w where we keep the letters at positions in P, and likewise for w_Q. The *shuffle* $\sqcup\!\sqcup(U)$ of a tuple of words U is defined by induction as follows: we set $\sqcup\!\sqcup() := \{\epsilon\}$, set $\sqcup\!\sqcup(u) := \{u\}$, and set $\sqcup\!\sqcup(u_1, \ldots, u_n, u_{n+1}) := \bigcup_{v \in \sqcup\!\sqcup(u_1, \ldots, u_n)} v \sqcup\!\sqcup u_{n+1}$. The *constrained shuffle problem* CSh$[K]$ for a fixed language $K \subseteq A^*$ is defined as follows: given a tuple of words U, determine if $K \cap \sqcup\!\sqcup(U)$ is nonempty. Of course, CSh$[K]$ is a special case of CTS$[K]$: we can code any tuple of words U as an A-DAG G_U by coding each $u \in U$ as a directed path graph $v_1 \to \cdots \to v_{|u|}$ with $\lambda(v_i) = u_i$ for all $1 \leq i \leq |u|$. Thus, we will equivalently see inputs to CSh as tuples of words (called *strings* in this context) or as A-DAGs that are unions of directed path graphs.

▶ **Example 2.1.** The problem CTS$[(ab)^*]$ on an input $\{a, b\}$-DAG G asks if G has a topological sort starting with an a, ending with a b, and alternating between elements of each label. The problem CSh$[(aa + b)^*]$ on a tuple U of strings on $\{a, b\}$ asks if there is an interleaving $w \in \sqcup\!\sqcup(U)$ such that all a^*-factors in w are of even length (e.g., *bbaabaaaa*, but not *baaabb*).

In this work, we study the complexity of the problems CTS$[K]$ and CSh$[K]$ depending on the language K. Clearly we can always solve these problems by guessing a topological sort (or an interleaving), and verifying that it achieves a word in K. Hence, the complexity

is always in NP^K, that is, in non-deterministic PTIME with an oracle for the *word problem* of K, which we can call to test if an input word in is K:

▶ **Proposition 2.2.** *For any language K, the problems* $\mathrm{CTS}[K]$ *and* $\mathrm{CSh}[K]$ *are in* NP^K.

In particular, the problems are in NP when the language K is regular, because the word problem for regular languages is in PTIME. We will study regular languages in this work. We believe that regular languages can be classified depending on the complexity of these problems, and make the following *dichotomy conjecture*:

▶ **Conjecture 2.3.** *For every regular language K, the problem* $\mathrm{CTS}[K]$ *is either in NL or NP-complete. Likewise, the problem* $\mathrm{CSh}[K]$ *is either in NL or NP-complete.*

Towards this conjecture, we determine in this paper the complexity of CTS and CSh for several languages and classes. We first show in the next section that these problems are hard for some languages such as $(ab)^*$, and we then show tractability results in Section 4, and a coarser dichotomy result in Section 5 under an alternative phrasing of our problems.

3 Hardness Results

Our hardness results are based on the *shuffle problem* of formal language theory which asks, given a word $w \in A^*$ and a tuple U of words of A^*, whether $w \in \sqcup\!\sqcup(U)$. This problem is known to be NP-hard already on the alphabet $\{a, b\}$ (see [32]). The shuffle problem is different from CSh, because the target word of the shuffle problem is given as input, whereas the target regular language of CSh is fixed. However, the hardness of the shuffle problem directly implies the hardness of CSh, hence of CTS, for a well-chosen target language:

▶ **Proposition 3.1.** *Let $K_0 := (a_1a_2 + b_1b_2)^*$. The problem* $\mathrm{CSh}[K_0]$ *is NP-hard.*

Proof sketch. We can reduce a shuffle instance (w, U) to the instance $I := w_1 \cup U_2$ for $\mathrm{CSh}[K_0]$, where w_1 is w but adding the subscript 1 to all labels, and U_2 is defined analogously. A topological sort of I achieving K_0 must then alternate between w_1 and U_2, and enumerate letters with the same label (up to the subscript), witnessing that $w \in \sqcup\!\sqcup(U)$. ◀

In this section, we will refine this approach to show hardness for more languages. We first recall another initial hardness result from [32]. We then introduce a general *shuffle reduction* technique to show the hardness of languages by reducing from other hard languages. Last, we show that CTS and CSh are hard for the language $(ab)^*$ and for other languages.

Initial hard family. To bootstrap the hardness results of [32] on the shuffle problem (on input words) to our CSh-problem (on fixed languages), we generalize the definition of CSh to a *regular language family* $\mathcal{K}$, i.e., a (generally infinite) family of regular languages, each of which is described as a regular expression. The CSh-problem for $\mathcal{K}$, written $\mathrm{CSh}[\mathcal{K}]$, asks, given a regular expression $K \in \mathcal{K}$ and a set of strings U, whether $K \cap \sqcup\!\sqcup(U)$ is nonempty. In other words, we no longer fix one single target language but a family $\mathcal{K}$ of target languages, and the input chooses one target language from the family $\mathcal{K}$. The following is then shown in [32] by reducing from UNARY-3-PARTITION [13]:

▶ **Lemma 3.2. ([32], Lemma 3.2)** *Let $\mathcal{K} := \{(a^ib^i)^* \mid i \in \mathbb{N}\}$. Then* $\mathrm{CSh}[\mathcal{K}]$ *is NP-hard.*

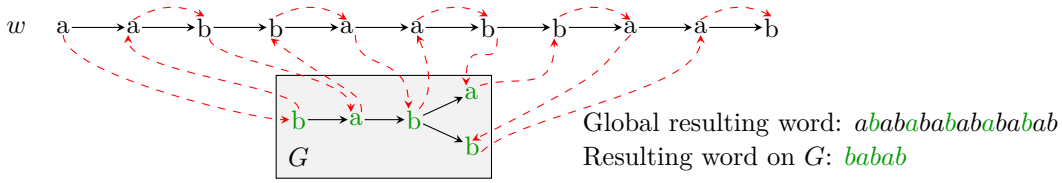


Figure 2 Example of a shuffle reduction from $K := (ba)^*b$ to $K' := (ab)^*$

Shuffle reduction. Our goal in this section is to show the hardness of CTS and CSh for more languages, but we do not wish to prove hardness for every language from scratch. Instead, we will introduce a general tool called the *shuffle reduction* that allows us to leverage the hardness of a language K to show that another language K' is also hard. Specifically, if a language K shuffle-reduces to a language K', this will imply that there is a PTIME reduction from CTS[K] to CTS[K'], and from CSh[K] to CSh[K'].

The intuition for the shuffle reduction is as follows: to reduce from K to K', given an input A-DAG G, we build an A-DAG G' formed of G plus an additional directed path labeled by a word w. Thus, any topological sort σ' of G' must be the interleaving of w and of a topological sort σ of G. Now, if we require that σ' achieves K', the presence of w can impose specific conditions on σ. Intuitively, if w is sufficiently long and "far away" from all words of K', then σ' must "repair" w to a word of K' by inserting symbols from G, so the insertions performed by σ may need to be in a specific order, i.e., σ may be forced to achieve a word of K. This means that solving CTS[K'] on G' allows us to solve CTS[K] on G. This intuition is illustrated on Figure 2: to achieve a word of $K' := (ab)^*$ on the DAG G', a topological sort must enumerate elements from G to insert them at the appropriate positions in w, achieving a word of $K := (ba)^*b$. We call *filter sequence* a family of words like w that allow us to reduce any CTS[K]-instance to CTS[K']. Formally:

▶ **Definition 3.3 (Filter sequence).** Let K and K' be languages on an alphabet A. A *filter sequence* for K and K' is an infinite sequence (f_n) of words of A^* having the following property: for every $n \in \mathbb{N}$, for every word $v \in A^*$ such that $|v| = n$, we have $v \in K$ iff $(v \sqcup\!\sqcup f_n) \cap K' \neq \emptyset$.

In Figure 2, we can choose $f_5 := w$ when defining a filter sequence for $(ba)^*b$ and $(ab)^*$: indeed, if we interleave w with any DAG G of 5 vertices, then a topological sort σ of G achieves K iff some interleaving σ' of σ with w achieves K'. We can now define our reduction:

▶ **Definition 3.4 (Shuffle reduction).** We say that a language K *shuffle-reduces* to a language K' if there is a filter sequence (f_n) for K and K' such that the function $i \mapsto f_i$ is computable in PTIME (where i is given in unary).

We say that a regular language family $\mathcal{K}$ *shuffle-reduces* to K' if each K does, and if we can compute in PTIME the function $(K, i) \mapsto f_i^K$, which maps a regular expression K of $\mathcal{K}$ and an integer i in unary to the i-th word in a filter sequence (f_n^K) for K and K'.

▶ **Theorem 3.5.** *For any regular language family $\mathcal{K}$ and language K', if $\mathcal{K}$ shuffle-reduces to K' then we can reduce in PTIME from* CTS[$\mathcal{K}$] *to* CTS[K']*, and from* CSh[$\mathcal{K}$] *to* CSh[K'].

Hardness for $(ab)^*$. We now use the shuffle reduction and the language family of Lemma 3.2 to show the hardness of $(ab)^*$. This will be instrumental for our coarser dichotomy in Section 5:

▶ **Theorem 3.6.** *The problem* CSh[$(ab)^*$] *(hence* CTS[$(ab)^*$]*) is NP-hard.*

Proof sketch. We shuffle-reduce from the language family $\mathcal{K}$ of Lemma 3.2: for the language $K_B = (a^B b^B)^*$ of $\mathcal{K}$, we define the filter sequence for words of length $2Bn$ by $f^B_{2Bn} := (b^B a^B ab)^n$. This ensures that, when interleaving f^B_{2Bn} with a word v of length $2Bn$ to achieve a word of $(ab)^*$, we must use v to insert in f^B_{2Bn} the letters written in bold: $((\mathbf{a}b)^B(a\mathbf{b})^B ab)^n$. This can be done iff $v = (a^B b^B)^n$, i.e., iff $v \in K_B$. We conclude by Theorem 3.5. ◀

Other hard languages. From the hardness of $(ab)^*$, we can use the shuffle reduction to show hardness for many other languages. For instance, we can show hardness for any language u^*, where $u \in A^*$ is a word with two different letters:

▶ **Proposition 3.7.** *Let $u \in A^*$ such that $|u|_a > 0$ and $|u|_b > 0$ for $a \neq b$ in A. Then* CSh$[u^*]$ *(hence* CTS$[u^*]$*) is NP-hard.*

Proof sketch. We shuffle-reduce from $(ab)^*$ with the filter sequence $f_{2n} := (uu_{-a}uu_{-b}u)^n$, where u_{-a} (resp. u_{-b}) is u but removing one occurrence of a (resp. of b). If a word v with $|v| = 2n$ has an interleaving w with f_{2n} that falls in u^*, then in w we must intuitively insert one a from v in each u_{-a} and one b from v in each u_{-b}, so that $v = (ab)^n$. To formalize this, we first rotate u to ensure that its first and last letters are different. We then observe that, as w is in u^*, any factor w' of length $|u|$ of w must be such that $|w'|_a = |u|_a$ and $|w'|_b = |u|_b$. We then consider factors of w of length $|u|$ centered on the u_{-a} and u_{-b} in f_{2n}: we argue that in w we must have inserted at least one a in or around each u_{-a}, and at least one b in or around each u_{-b}, otherwise these factors do not have enough a's and enough b's. ◀

We can also use the shuffle reduction to show hardness for other languages, e.g., $(aa+bb)^*$:

▶ **Proposition 3.8.** *Let $L := (aa + bb)^*$. The problem* CSh$[L]$ *(hence* CTS$[L]$*) is NP-hard.*

Proof sketch. We do again a shuffle reduction from $(ab)^*$, with the filter sequence $f_{2n} = (ab)^n$. If a word v with $|v| = 2n$ is such that $v \sqcup\!\sqcup f_{2n}$ intersects $(aa + bb)^*$ nontrivially, it must intuitively insert a's and b's in f_{2n} alternatively, so it must be $(ab)^n$. Note that a similar proof would also show hardness for the language $(a^i + b^j)^*$ for any choice of $i, j \geq 2$. ◀

We show a last result that does not use the shuffle reduction but an easy consideration on the number of letter occurrences. This result will be useful in Section 5:

▶ **Proposition 3.9.** *The problem* CSh$[(ab + b)^*]$ *(hence* CTS$[(ab + b)^*]$*) is NP-hard.*

Proof. We describe an easy PTIME reduction from CSh$[(ab)^*]$ to CSh$[(ab + b)^*]$. Given an instance I, check if the number of a-labeled and b-labeled vertices is the same, and fail if it is not. Otherwise, then I achieves a word of $(ab + b)^*$ iff it achieves one of $(ab)^*$, because we must enumerate one a-labeled vertex with each b-labeled vertex. ◀

We believe that the shuffle reduction applies to many other languages, though we do not know how to characterize them. In particular, we believe that the following could be shown with the shuffle reduction, generalizing all the above hardness results except Proposition 3.8:

▶ **Conjecture 3.10.** *Let F be a finite language such that, for some letter $a \in A$, the language F contains no power of a but contains a word which contains a. Then* CSh$[F^*]$ *is NP-hard.*

4 Tractability Results

Having shown hardness for several languages, we now present our tractability results. We will also rely on some of these results to show our coarser dichotomy result in the next section.

Closure under union. The first observation on tractable languages is that they are closed under union, as follows (recalling the definition of CTS and CSh for language *families*):

▶ **Lemma 4.1.** *For any finite family of languages $\mathcal{K}$, there is a logspace reduction from* CTS$[\bigcup \mathcal{K}]$ *to* CTS$[\mathcal{K}]$*, and likewise from* CSh$[\bigcup \mathcal{K}]$ *to* CSh$[\mathcal{K}]$*.*

Proof. To solve a problem for the language $\bigcup \mathcal{K}$ on an input instance I, simply enumerate the languages $K' \in \mathcal{K}$, and solve the problem on I for each K'. Clearly I is a positive instance of the problem for $\bigcup \mathcal{K}$ iff I is a positive instance of the problem for one of the K'. ◀

▶ **Corollary 4.2.** *For any finite family of languages $\mathcal{K}$, if* CTS$[K']$ *is in NL for each $K' \in \mathcal{K}$, then so is* CTS$[\bigcup \mathcal{K}]$*. The same is true of the CSh-problem.*

Clearly, tractability is also preserved under the *reverse operator*, i.e., reversing the order of words in a language; however tractable languages are *not* closed under many usual operators, as we will show in Section 5. Still, closure under union will often be useful in the sequel.

Monomials. We will now show that CTS is tractable for an important family of languages (and unions of such languages): the *monomial* languages. Having fixed the alphabet A, a *monomial* is a language of the form $A_1^* a_1 A_2^* a_2 \cdots a_n A_{n+1}^*$ with $a_i \in A$ and $A_i \subseteq A$ for all i. In particular, we may have $A_i = \emptyset$ so that $A_i^* = \epsilon$: hence, for every word $u \in A^*$, the language $A^* u A^*$ is a monomial language, which intuitively tests whether a word contains the pattern u. Several decidable algebraic and logical characterizations of these languages are known; in particular, unions of monomials are exactly the languages that are definable in the first-order logic fragment $\Sigma_2[<]$ of formulas with quantifier prefix $\exists^* \forall^*$, and it is decidable to check if a regular language is in this class [24, 22]. We show:

▶ **Theorem 4.3.** *For any monomial language K, the problem* CTS$[K]$ *is in NL.*

Proof sketch. Let K be $A_1^* a_1 A_2^* a_2 \cdots A_n^* a_n A_{n+1}^*$. We can first guess in NL the vertices $v_1, \ldots, v_n$ to which the $a_1, \ldots, a_n$ are mapped, so all that remains is to check, for each such guess, whether we can match the remaining vertices to the A_i. We proceed by induction on n. The base case of $n = 0$ (i.e., $K = A_1^*$) is trivial. For the induction step, using the fact that NL = co-NL (see [15, 28]), we check that the descendants of the last element v_n are all in A_{n+1}^*, and then we compute the set S of vertices that *must* be enumerated before v_n: they are the ancestors of the v_i, and the ancestors of any vertex labeled by a letter in $A \setminus A_{n+1}$. We then use the induction hypothesis to check in NL whether S has a topological sort that achieves a word in $A_1^* a_1 \ldots A_{n-1}^* a_{n-1} A_n^*$. ◀

Tractability based on width. While unions of monomials are a natural class, it turns out that they do not cover all tractable languages. In particular, we can show:

▶ **Proposition 4.4.** *Let $A := \{a, b\}$ and $K := (ab)^* + A^* aa A^*$. The problem* CTS$[K]$ *(hence* CSh$[K]$*) is in NL.*

This result is not covered by Theorem 4.3, because we can show that K cannot be expressed as a union of monomials (see the long version [5]); and the proof technique is different.

Proof. Let G be an input A-DAG. We first check in NL if G contains two incomparable vertices $v_1 \neq v_2$ such that $\lambda(v_1) = \lambda(v_2) = a$. If yes, we conclude that G is a positive instance, as we can clearly achieve K by enumerating v_1 and v_2 contiguously.

If there are no two such vertices, we check in NL if there are two comparable a-labeled vertices $v_1 \neq v_2$ that can be enumerated contiguously, i.e., there is an edge $v_1 \to v_2$ but no vertex w that is *between* v_1 and v_2, i.e., is a descendant of v_1 and an ancestor of v_2. If there are two such vertices v_1 and v_2, we conclude again that G is a positive instance.

Otherwise, our first test implies that G induces a total order on the a-labeled vertices, and our second test implies that any two consecutive a-labeled vertices in this order must have at least one b-labeled vertex between them. This ensures that no topological sort achieves A^*aaA^*, so it suffices to test whether one can achieve $(ab)^*$. Clearly this is the case iff all consecutive pairs of a-labeled vertices have exactly one b-labeled vertex between them, and there is exactly one additional b-labeled vertex that can be enumerated after the last a-labeled vertex. We can test this in NL, which concludes the proof. ◀

Intuitively, the language of Proposition 4.4 is tractable because it is easy to solve unless the input instance has a very restricted structure, namely, all a's are comparable. We do not know whether this result generalizes to $(ab)^* + A^*a^iA^*$ for $i > 2$. However, following the intuition of this proof, we can show the tractability of a similar kind of regular languages:

▶ **Proposition 4.5.** *Let $A := \{a, b\}$, let K' be a regular language, let $i \in \mathbb{N}$, and let $K := K' + A^*(a^i + b^i)A^*$. The problem* CTS$[K]$ *(hence* CSh$[K]$*) is in NL.*

As in Proposition 4.4, CTS is trivial for the languages in this proposition unless the input A-DAG G has a restricted shape. Here, the requirement is on the *width* of G, i.e., the maximal cardinality of a subset of pairwise incomparable vertices (called an *antichain*), so we can show Proposition 4.5 by distinguishing two cases depending on the width of G:

Proof sketch. We test in NL whether the input A-DAG G contains an antichain C of size $2i$: if it does, then at least i vertices in C must have the same label, and we can enumerate them in succession to achieve $A^*a^iA^*$ or $A^*b^iA^*$, so G is a positive instance. Otherwise, G has width $< 2i$, and Dilworth's theorem [9] implies that its elements can be partitioned into chains, so that CTS can be solved in NL following a dynamic algorithm on them. ◀

Other tractable case. We close the section with another example of a regular language which is tractable for the CSh-problem for what appears to be a unrelated reason.

▶ **Proposition 4.6.** *Let $A := \{a, b\}$ and $K := (aa + b)^*$. The problem* CSh$[K]$ *is in NL.*

This is in contrast to $(aa + bb)^*$, for which we showed intractability (Proposition 3.8). We do not know the complexity of the CTS-problem for $(aa + b)^*$, or the complexity for either problem of languages of the form $(a^i + b)^*$ for $i > 2$.

Proof sketch. We show that the existence of a suitable topological sort can be rephrased to an NL-testable equivalent condition, namely, there is no string in the input instance whose number of odd "blocks" of a-labeled elements dominates the total number of a-labeled elements available in the other strings. If the condition fails, then we easily establish that no suitable topological sort can be constructed: indeed, eliminating each odd block of a's in the dominating string requires one a from the other strings. If the condition holds, we can simplify the input strings and show that a greedy algorithm can find a topological sort by picking pairs of a's in the two current heaviest strings. ◀

5 A Coarser Dichotomy Theorem

In the two previous sections, we have established some intractability and tractability results about the constrained topological sort and constrained shuffle problems for various languages. Remember that our end goal would be to characterize the tractable and intractable languages, and show a dichotomy (Conjecture 2.3). This is difficult, and one reason is that the class of tractable languages is not "well-behaved": while it is closed under the union operator (Corollary 4.2), it is is *not* closed under intersection, complement, and other common operations. This makes it difficult to study tractable languages using algebraic language theory [23].

▶ **Proposition 5.1.** *We have the following counterexamples to closure:*

- ***Quotient.*** *There exists a word $u \in A^*$ and a regular language K such that* $\mathrm{CSh}[K]$ *is in NL but* $\mathrm{CSh}\left[u^{-1}K\right]$ *is NP-hard.*
- ***Intersection.*** *There exists two regular languages K_1 and K_2 such that* $\mathrm{CTS}[K_1]$ *and* $\mathrm{CTS}[K_2]$ *are both in PTIME but* $\mathrm{CSh}[K_1 \cap K_2]$ *is NP-hard*
- ***Complement.*** *There exists a regular language K such that* $\mathrm{CTS}[K]$ *is in NL, but* $\mathrm{CSh}[A^* \setminus K]$ *is NP-hard.*
- ***Inverse of morphism.*** *There exists a regular language K and morphism φ such that* $\mathrm{CTS}[K]$ *is in NL but* $\mathrm{CSh}\left[\varphi^{-1}(K)\right]$ *is NP-hard.*

The three last results of this proposition also apply to the constrained topological sort problem, but the first one does not, and in fact CTS-tractable languages *are* closed under quotients. This observation implies that there are regular languages K such that $\mathrm{CSh}[K]$ is tractable but $\mathrm{CTS}[K]$ is NP-hard; one concrete example is $K := b^*A^* + aaA^* + (ab)^*$ (see long version [5]). We sketch the proof of Proposition 5.1:

Proof sketch. For each operation, we use $(ab)^*$ as our NP-hard language (by Theorem 3.6).

For quotient, we take $K := bA^* + aaA^* + (ab)^*$, and $u := ab$. We have $u^{-1}K = (ab)^*$, but $\mathrm{CSh}[K]$ is in NL because any shuffle instance with more than one string satisfies K.

For intersection, we take $K_1 := (ab)^*(\epsilon + bA^*)$ and $K_2 := (ab)^*(\epsilon + aaA^*)$. We have $K_1 \cap K_2 = (ab)^*$, but $\mathrm{CSh}[K_1]$ and $\mathrm{CSh}[K_2]$ are in PTIME using an ad-hoc greedy algorithm.

For complement, we take $K := bA^* \cup A^*a \cup A^*aaA^* \cup A^*bbA^*$. As K is a union of monomials, we know by Theorem 4.3 that $\mathrm{CTS}[K]$ is in NL, but we have $A^* \setminus K = (ab)^*$.

For inverse of morphism, we take $A := \{a, b\}$ and $K := (ab)^* + A^*(a^3 + b^3)A^*$. We know that $\mathrm{CTS}[K]$ is in PTIME by Proposition 4.5. Now, defining $\varphi : A^* \to A^*$ by $\varphi(a) := aba$ and $\varphi(b) := bab$, we have $\varphi^{-1}(K) = (ab)^*$ because no word in the image of φ has three identical consecutive symbols. ◀

Proposition 5.1 suggests that tractable languages would be easier to study algebraically if we ensured that they were closed under all these operations, i.e., if they formed a variety [23]. In this section, we enforce this by moving to an alternative phrasing of the CTS and CSh problems. This allows us to leverage algebraic techniques and show a dichotomy theorem in this alternative phrasing, under an additional *counter-free* assumption. We first present the alternative phrasing, and then present the additional assumption and our dichotomy result.

Alternative phrasing. The first change in our alternative phrasing is that the input DAG G will now be an A^*-*DAG*, i.e., a DAG labeled with words of A^* rather than letters of A. As before, a topological sort σ of G *achieves* a word $\lambda(\sigma) \in A^*$ obtained by concatenating the λ-images of the vertices of G in the order of σ: but vertex labels are now "atomic" words

whose letters cannot be interleaved with anything else. The *multi-letter* CTS and CSh problems are the variants defined with A^*-DAGs; intuitively, this ensures that tractable languages are closed under inverse morphisms.

The second change is that we will not fix one single target language, but a *semiautomaton* [14], i.e., an automaton where initial and final states are not specified. Formally, a semiautomaton is a tuple (Q, A, δ) where Q is the set of states, A is the alphabet, and $\delta : Q \times A \to Q$ is the transition function; we extend δ to words as usual by setting $\delta(q, \epsilon) := q$ and $\delta(q, u_1 \cdots u_{n+1}) := \delta(\delta(q, u_1), u_2 \cdots u_{n+1})$. We will fix the target semiautomaton, and the initial and final states will be given in the input instance (in addition to the DAG). This enforces closure under quotients (by choosing the initial and final states) and complement (by toggling the final states). Further, to impose closure under intersection, the input instance will specify a *set* of pairs of initial-final states, with a logical AND over them. The question is to determine whether the input DAG achieves a word accepted by *all* the corresponding automata; and this enforces closure under intersection.

We can now summarize the formal definition of our problem variants. The *multi-letter* CTS-*problem for a fixed semiautomaton* $S = (Q, A, \delta)$ takes as input an A^*-DAG and a set $\{(i_1, F_1), \ldots, (i_k, F_k)\}$ of initial-final state pairs, where $i_j \in Q$ and $F_j \subseteq Q$ for all $1 \leq j \leq k$. The input is accepted if there is a topological sort σ of G such that, for all $1 \leq j \leq k$, the word $\lambda(\sigma)$ is accepted by the automaton (Q, A, δ, i_j, F_j), i.e., $\delta(i_j, \lambda(\sigma)) \in F_j$. The *multi-letter* CSh-*problem for a fixed semiautomaton* is defined in the same way, imposing that the input A^*-DAG is a union of directed path graphs.

Dichotomy result Our dichotomy will apply to the multi-letter CTS and CSh problem for semiautomata. However, we will need to make an additional assumption, namely, that the semiautomaton is *counter-free*. This assumption means that our dichotomy will only apply to a well-known subset of regular languages, namely, the *star-free languages*, that are better understood algebraically; it excludes in particular the tricky case of *group languages* that we will study separately in Section 6. Formally, a semiautomaton is *counter-free* if, for every state q and word $u \in A^*$, if $\delta(q, u^n) = q$ for some $n > 1$, then we have $\delta(q, u) = q$. Under the counter-free assumption, we can prove the following dichotomy, using our hardness and tractability results in Sections 3 and 4:

▶ **Theorem 5.2.** *Let S be a counter-free semiautomaton. Then the multi-letter CSh-problem and CTS-problem for S are either both in NL, or both NP-complete. The dichotomy is effective: given S, it is PSPACE-complete to decide which case applies.*

We conclude the section by introducing some technical tools used for this result and for Section 6, and by giving a proof sketch. The criterion of the dichotomy on S is phrased in terms of the *transition monoid* of S, which we now define (see, e.g., [23] for details). Remember that a monoid is a set that has an associative binary operation and a neutral element. The *transition monoid* $T(S)$ of a semiautomaton $S = (Q, A, \delta)$ is the set of functions $f : Q \to Q$ that are "achieved" by S in the following sense: there is a word $u \in A^*$ such that $\delta(q, u) = f(q)$ for all $q \in Q$. In particular, the neutral element is the identity function, which is achieved by taking $u := \epsilon$; and the binary operation on $T(S)$ is function composition, which is associative. Note that the transition monoid is finite and can be computed from S.

We assumed that S is counter-free, and this is equivalent [20] to saying that $T(S)$ is in the class $\mathbf{A}$ of *aperiodic* finite monoids (formally defined by the equation $x^{\omega+1} = x^\omega$ where ω is the idempotent power [23] of the monoid). Within $\mathbf{A}$, our dichotomy criterion on $T(S)$ is based on a certain subclass of $\mathbf{A}$, called $\mathbf{DA}$ (see [30]): S is tractable iff $T(S)$ is in $\mathbf{DA}$, and it is PSPACE-complete [31] to test whether this holds (using the formal definition of $\mathbf{DA}$ by the equation $(xy)^\omega x (xy)^\omega = (xy)^\omega$). We can now sketch the proof of Theorem 5.2:

Proof sketch. We first show that if $T(S)$ is in **DA** then the multi-letter CTS and CSh problems for S are in NL. For this, we rely on one characterization of **DA** (from [30]): if $T(S)$ is in **DA** then the regular languages recognized by S (for any set of initial-final states) are unions of *unambiguous monomials*, in particular they are unions of monomials, so we have tractability by Corollary 4.2 and Theorem 4.3.

For the converse direction, we use a second characterization of **DA** (from [29]): if $T(S)$ is *not* in **DA** then there is a choice of initial-final state pairs for which S computes a language K whose inverse image by some morphism is either $(ab)^*$ or $(ab+b)^*$. We know that these languages are intractable (Theorem 3.6 and Proposition 3.9) so we conclude by showing a PTIME reduction from one of these two languages: this is possible in our alternative problem phrasing, in particular using the multi-letter labels to invert the morphism. ◀

6 Lifting the Counter-Free Assumption for CSh

Our dichotomy theorem in the previous section (Theorem 5.2) was shown for an alternative phrasing of our problems (with semiautomata and multi-letter inputs), and made the additional assumption that the input semiautomaton is counter-free. In this section, we study how to lift the counter-free assumption. In exchange for this, we restrict our study to the constrained shuffle problem (CSh) rather than CTS.

To extend Theorem 5.2 for the CSh-problem, we will again classify the semiautomata S based on their transition monoid $T(S)$. However, instead of **DA**, we will use the two classes **DO** and **DS** introduced in [27] (formally **DO** is defined by the equation $(xy)^\omega(yx)^\omega(xy)^\omega = (xy)^\omega$ and **DS** by the equation $((xy)^\omega(yx)^\omega(xy)^\omega)^\omega = (xy)^\omega$ for ω the idempotent power). Both **DO** and **DS** are supersets of **DA**, specifically we have $\mathbf{DA} \subseteq \mathbf{DO} \subseteq \mathbf{DS}$, and we can test in PSPACE in S whether $T(S)$ is in each of these classes [31]. Our main result is then:

▶ **Theorem 6.1.** *Let S be a semiautomaton. If $T(S)$ is in **DO**, then the multi-letter CSh-problem for S is in NL. If $T(S)$ is not in **DS**, then it is NP-complete.*

This result generalizes Theorem 5.2 for the CSh-problem, because both **DO** and **DS** collapse to **DA** for aperiodic monoids (see [27] and [2, Chapter 8]); formally, $\mathbf{DO} \cap \mathbf{A} = \mathbf{DS} \cap \mathbf{A} = \mathbf{DA}$. However, **DO** covers more languages than **DA**: the main technical challenge to prove Theorem 6.1 is to show that CSh is tractable for these languages. One important example are the *group languages* over A: these are the regular languages recognized, for some choice of initial-final state pairs, by a semiautomaton S over A such that $T(S)$ is a group. A more general example are *district group monomials*, which are the languages of the form $K_1a_1 \cdots K_na_nK_{n+1}$ where, for all i, we have $a_i \in A$ and K_i is a group language over some alphabet $A_i \subseteq A$. Note that district group monomials are more expressive than the *group monomials* defined in earlier work [25] (which set $A_i := A$ for all i), and they also generalize the monomials that we studied in Section 4 (any A_i^* is trivially a group language over A_i, even though it is not a group language over A). In fact, to prove Theorem 6.1, what we need is to generalize Theorem 4.3 (for CSh) from monomials to district group monomials:

▶ **Theorem 6.2.** *Let K be a district group monomial. Then* CSh$[K]$ *is in NL.*

Note that this theorem, like Theorem 4.3, applies to the original phrasing of CSh, not the alternative phrasing with semiautomata and multi-letter DAGs. Thus, Theorem 6.2 implies that the original CSh-problem is tractable for many languages that we had not covered previously, e.g., $(ab^*a+b)^*c(ba^*b+a)^*$, the language testing whether there is one c preceded by an even number of a and followed by an even number of b. The proof of Theorem 6.2 is our main technical achievement, and we sketch it below (see the long version [5] for details):

Proof sketch. We focus on the simpler case of a group language, for a finite group H. The problem can be rephrased directly in terms of H: given a tuple I of strings over H and a target element $g \in H$, determine if there is an interleaving of I that evaluates to g under the group operation. Our approach partitions H into the *rare* elements H_{rare}, that occur in a constant number of strings, and the *frequent* elements H_{freq}, that occur in sufficiently many strings. For the frequent elements, we can build a large antichain C from the strings where they occur, with each element of H_{freq} occuring many times in C. Now, as topological sorts can choose any order on C, they can intuitively achieve all elements of the subgroup $\langle H_{\text{freq}} \rangle$ generated by H_{freq}, except that they cannot change "commutative information", e.g., the parity of the number of elements. We formalize the notion of "commutative information" using relational morphisms, and prove an *antichain lemma* that captures our intuition that all elements of $\langle H_{\text{freq}} \rangle$ with the right commutative information can be achieved.

For the rare elements, we can simply follow a dynamic algorithm on the constantly many strings where they occur. However, we must account for the possibility of inserting elements of $\langle H_{\text{freq}} \rangle$ from the other strings, and we must show that it suffices to do constantly many insertions, so that it was sufficient to impose a constant lower bound on $|C|$. We formalize this as an *insertion lemma*, which we prove using Ramsey's theorem. ◀

We close the section by commenting on the two main limitations of Theorem 6.1. The first limitation is that it is not a dichotomy: it does not cover the semiautomata with transition monoid in $\mathbf{DS} \setminus \mathbf{DO}$. We do not know if the corresponding languages are tractable or not; we have not identified intractable cases, but we can show tractability, e.g., for $(a^+b^+a^+b^+)^*$, the language of words with an even number of subfactors of the form a^+b^+.

▶ **Proposition 6.3.** *Let* $K = (a^+b^+a^+b^+)^*$. *Then* $\mathrm{CSh}[K]$ *is in NL.*

However, it would be difficult to show tractability for all of **DS**, because **DS** is still poorly understood in algebraic language theory. For instance, characterizing the languages with a syntactic monoid in **DS** has been open for over 20 years [2, Open problem 14, page 442].

The second limitation of Theorems 6.1 and 6.2 is that they only apply to CSh. New problems arise with CTS: for instance, an $\{a, b\}$-DAG G may contain large antichains C_a and C_b of a-labeled and b-labeled vertices, and yet contain no antichain with many a-labeled and b-labeled vertices (e.g., if G is the series composition of C_a and C_b). The missing proof ingredient seems to be an analogue of Dilworth's theorem for *labeled* DAGs (see also [3]).

7 Conclusion and Open Problems

We have studied the complexity of two problems, constrained topological sort (CTS) and constrained shuffle (CSh): fixing a regular language K, given a labeled DAG (for CTS) or a tuple of strings (for CSh), we ask if the input DAG has a topological sort achieving K. We have shown tractability and intractability for several regular languages using a variety of techniques. These results yield a coarser dichotomy (Theorem 5.2) in an alternate problem phrasing that imposes some closure assumptions.

Our work leaves the main dichotomy conjecture open (Conjecture 2.3). Even in the alternate problem phrasing of Theorem 5.2, our dichotomy only covers counter-free semiautomata: the restriction is lifted in Section 6 but only for CSh, and with a gap between tractability and intractability. In the original phrasing, there are many concrete languages that we do not understand: Does Proposition 4.4 extend to $(ab)^* + A^*a^iA^*$ for $i > 2$? Does Proposition 4.6 extend to $(a^i + b)^*$ for $i > 2$, or to CTS rather than CSh? Can we show Conjecture 3.10?

Another direction would be to connect CSh and CTS to the framework of *constraint satisfaction problems* (CSP) [11], which studies the complexity of homomorphism problems for fixed "constraints" (right-hand-side of the homomorphism). If this were possible, it could lead to a better understanding of our tractable and hard cases. However, CTS does not seem easy to rephrase in CSP terms: topological sorts and regular language constraints seems hard to express in terms of homomorphisms, even in extensions such as *temporal* CSPs [6, 7].

One last question would be to investigate CTS and CSh for *non-regular* languages. The simplest example is the Dyck language, which appears to be NP-hard for CTS (at least in the multi-letter setting), but tractable for CSh, via a connection to scheduling; see [12], problem SS7. More generally, CTS and CSh could be studied, e.g., for context-free languages, where the complexity landscape may be equally enigmatic.

References

1 Kunal Agrawal, Jing Li, Kefu Lu, and Benjamin Moseley. Scheduling parallel DAG jobs online to minimize average flow time. In *Proc. SODA*, 2016.

2 J. Almeida. *Finite Semigroups and Universal Algebra*. Series in algebra. World Scientific, 1994.

3 Antoine Amarilli. Generalization of Dilworth's theorem for labeled DAGs, 2016. URL: `https://cstheory.stackexchange.com/q/37062`.

4 Antoine Amarilli, M. Lamine Ba, Daniel Deutch, and Pierre Senellart. Possible and certain answers for queries over order-incomplete data. In *Proc. TIME*, 2017.

5 Antoine Amarilli and Charles Paperman. A dichotomy on constrained topological sorting. *CoRR*, abs/1707.04310, 2017. `arXiv:1707.04310`.

6 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *JACM*, 57(2):9, 2010.

7 Manuel Bodirsky, Barnaby Martin, and Antoine Mottet. Discrete temporal constraint satisfaction problems, 2015. `arXiv:1503.08572`.

8 Sam Buss and Michael Soltys. Unshuffling a square is NP-hard. *JCSS*, 80(4), 2014.

9 Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 1950.

10 Joey Eremondi, Oscar H Ibarra, and Ian McQuillan. On the complexity and decidability of some problems involving shuffle, 2016. `arXiv:1606.01199`.

11 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM J. Comput.*, 28(1), 1998.

12 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

13 Michael R. Garey and David S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM J. Comput*, 1975.

14 M. Holcombe. *Algebraic Automata Theory*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1982.

15 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput*, 17(5), 1988.

16 David S. Johnson. The NP-completeness column: an ongoing guide. *Journal of Algorithms*, 5(2), 1984.

17 John Kececioglu and Dan Gusfield. Reconstructing a history of recombinations from a set of sequences. *Discrete Applied Mathematics*, 88(1-3), 1998.

18 Takayuki Kimura. An algebraic system for process structuring and interprocess communication. In *Proc. STOC*, 1976.

19 Anthony Mansfield. On the computational complexity of a merge recognition problem. *Discrete Applied Mathematics*, 5(1), 1983.

20 Robert McNaughton and Seymour Papert. *Counter-Free Automata*. MIT Press, 1971.

21 W. F. Ogden, W. E. Riddle, and W.C. Rounds. Complexity of expressions allowing concurrency. In *Proc. POPL*, 1978.

22 Charles Paperman. Semigroup online, 2018. URL: `https://www.paperman.name/semigroup/`.

23 Jean-Éric Pin. Syntactic semigroups. In *Handbook of formal languages, Vol. 1*, pages 679–746. Springer, Berlin, 1997.

24 Jean-Éric Pin and Pascal Weil. Polynominal closure and unambiguous product. *TCS*, 30(4), 1997.

25 Jean-Éric Pin. Polynomial closure of group languages and open sets of the Hall topology. *TCS*, 169(2), 1996.

26 Romeo Rizzi and Stéphane Vialette. On recognizing words that are squares for the shuffle product. *TCS*, 2017.

27 M. P. Schützenberger. Sur le produit de concaténation non ambigu. *Semigroup forum*, 13, 1976/77.

28 Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3), 1988.

29 Pascal Tesson and Denis Thérien. The computing power of programs over finite monoids. *J. Autom. Lang. Comb.*, 7(2), 2001.

30 Pascal Tesson and Denis Thérien. Diamonds are forever: the variety DA. *Semigroups, algorithms, automata and languages*, 1, 2002.

31 Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *Proc. STOC*, 1998.

32 Manfred K. Warmuth and David Haussler. On the complexity of iterated shuffle. *JCSS*, 28(3), 1984.

On Zero-One and Convergence Laws for Graphs Embeddable on a Fixed Surface

Albert Atserias[1]
Universitat Politècnica de Catalunya, Barcelona, atserias@cs.upc.edu

Stephan Kreutzer[2]
Technical University Berlin, stephan.kreutzer@tu-berlin.de

Marc Noy[3]
Universitat Politècnica de Catalunya, Barcelona, marc.noy@upc.edu

Abstract

We show that for no surface except for the plane does monadic second-order logic (MSO) have a zero-one-law – and not even a convergence law – on the class of (connected) graphs embeddable on the surface. In addition we show that every rational in [0,1] is the limiting probability of some MSO formula. This strongly refutes a conjecture by Heinig et al. (2014) who proved a convergence law for planar graphs, and a zero-one law for connected planar graphs, and also identified the so-called gaps of [0,1]: the subintervals that are not limiting probabilities of any MSO formula. The proof relies on a combination of methods from structural graph theory, especially large face-width embeddings of graphs on surfaces, analytic combinatorics, and finite model theory, and several parts of the proof may be of independent interest. In particular, we identify precisely the properties that make the zero-one law work on planar graphs but fail for every other surface.

2012 ACM Subject Classification Mathematics of computing → Graphs and surfaces, Mathematics of computing → Enumeration, Theory of computation → Finite Model Theory

Keywords and phrases topological graph theory, monadic second-order logic, random graphs, zero-one law, convergence law

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.116

1 Introduction

We consider classes of labelled graphs with the uniform distribution on graphs with a fixed number of vertices. Let S be a closed compact surface, and let $\mathcal{G}_S$ be the class of graphs embeddable on S. Heinig et al. [9] studied limiting probabilities of first order (FO) and monadic second order (MSO) properties of graphs in the classes $\mathcal{G}_S$. They showed that a convergence law in FO holds for all surfaces S, and that a convergence law holds in MSO when S is the sphere, so that $\mathcal{G}_S$ is the class of planar graphs. They also showed that a zero-one law in FO holds for *connected* graphs in $\mathcal{G}_S$, and a zero-one law in MSO for connected planar graphs. They conjectured that these results extend to MSO properties on

[1] Partially funded by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement ERC-2014-CoG 648276 (AUTAR) and MICCIN grant TIN2016-76573-C2-1P (TASSAT3).
[2] Research supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC Consolidator Grant DISTRUCT, grant agreement No 648527).
[3] Partially supported by Grants MTM2014-54745-P and MDM-2014-0445.

an arbitrary surface. We strongly refute these conjectures. For each surface S other than the sphere we construct an MSO formula φ such that the probability that φ is satisfied in $\mathcal{G}_S$ does not converge when the number of vertices n tends to infinity, not even on connected graphs. In addition, telling whether the probability of a given MSO formula φ converges is an undecidable problem. We also show that every rational number in [0,1] is the limiting probability of some MSO formula for connected graphs in $\mathcal{G}_S$.

We sketch next the main ingredients in the proofs of our results. We fix a surface S different from the sphere. Embeddings of graphs on surfaces can be defined in purely combinatorial terms. If S is orientable, an embedding of a graph on S is given by a rotation system consisting, for each vertex v, of a cyclic orientation of the edges incident with v. If S is non-orientable one has to consider also signed edges [10]. Given a graph G embedded in S the *face-width* of the embedding is the minimum number of intersections of G with a non-contractible curve in S. In what follows, we say that a graph property holds with high probability (w.h.p.), or asymptotically almost surely (a.a.s.), if it holds with probability tending to 1 as n tends to infinity. It is known [10] that if the face-width of a 3-connected graph G is at least $2g$, where g is the genus of G (orientable or not), then G has a unique embedding in S. We also need the fact that the face-width of a random 3-connected graph G embedded on S is $\Omega(\log n)$ w.h.p. [1].

It is shown in [2] that w.h.p. a random graph G in $\mathcal{G}_S$ has a unique non-planar 3-connected component C. Since planarity is MSO expressible and 3-connected components are MSO definable, so is C. Using the fact that w.h.p. the face-width of C is large, we show the existence of an MSO definable grid structure M in G of size $\Omega(\log\log n)$. This is obtained starting with a non-contractible cycle, which is MSO definable, and extending it to a grid structure. Inspired by the capacity of MSO to emulate Turing machine computations on grid graphs, we are then able to define an MSO formula φ expressing the property that $\log^\dagger |M|$ is in $\{0,1,2,3\}$ modulo 8, where $\log^\dagger n$ is a variant of $\log^* n$ (see Section 5). Given that $\log\log n \leq |M| \leq n$, the value of $\log^\dagger |M|$ modulo 8 oscillates and, as a consequence, the probability that φ holds does not converge as n goes to infinity.

The non-converging formula φ, combined with the abovementioned capacity of MSO to emulate Turing machine computations on grid graphs, also gives the undecidability of the decision problem for converging probabilities. For each Turing machine M, we find an MSO formula φ_M that simulates partial runs of M along the definable growing grids. The formula φ_M has asymptotic probability 1 if M halts, and asymptotic probability 0 otherwise, and the probability of the conjunction $\varphi \wedge \varphi_M$ is thus converging if and only if M halts.

To prove that each rational in $[0,1]$ is the limiting probability of some MSO formula, we use the following facts. (a) A 3-connected graph in a surface of genus g has a spanning tree with maximum degree at most $4g$ [7]. (b) For the class $\mathcal{G}_S$, every property in MSO_2 (quantification over vertices and edges) is also expressible in MSO [5]. (c) The size X_n of the unique non-planar component obeys a limit local law related to a stable law [8]. Consider now a random graph G in $\mathcal{G}_S$ with n vertices, let C be the unique non-planar component, and let $X_n = |C|$. From properties (a) and (b) it follows that, for each integers a and b, we can express in MSO the property $\varphi_{a,b}$ that X_n is equal to a modulo b. Using property (c) we show that the probability that $\varphi_{a,b}$ is satisfied tends to $1/b$. Now given any rational non-negative $a/b \leq 1$, the property that X_n is less than a modulo b tends to a/b.

The key fact that makes MSO properties of graphs on non-planar surfaces so different from the plane is that w.h.p. there is a unique *non-planar* 3-connected component, which also happens to be the only one of linear size. For random planar graphs there is a unique 3-connected component of linear size as well, but it is indistinguishable in MSO from the smaller ones.

2 Graphs and Surfaces

In this section we present the background from graph theory and surface topology we need for our main result. We refer to [10, 6] for background on graphs and surfaces. Our notation on surfaces and embeddings follows [10].

Graphs. All graphs in this paper are finite, undirected, and simple, i.e. no parallel edges or self-loops. We denote the vertex and edge set of a graph G by $V(G)$ and $E(G)$, resp. For $d \geq 0$, a graph G is *d-degenerate* if every subgraph $H \subseteq G$ contains a vertex of degree at most d.

We need the concept of 3-connected components of a graph G. For $k > 0$, a graph G is *k-connected* if $|V(G)| > k$ and $G - S$ is connected for all $S \subseteq V(G)$ with $|S| < k$. A *connected component* of G is an inclusion-wise maximal connected subgraph $C \subseteq G$. A *2-connected component* of G is an inclusion-wise maximal 2-connected subgraph $C \subseteq G$. For defining *3-connected components* of G we need some preparation.

Let G be a 2-connected graph. A *separator of order* 2 in G is a set $X = \{u, v\}$ of distinct vertices such that $G - X$ is not connected. Let C be an inclusion-wise maximal subgraph of G such that for all 2-separations X of G there is one connected component of $G - X$ which contains all vertices of $C - X$. For any such 2-separation X of G, if $X \subseteq V(C)$ then we add an edge between the two vertices of X in C if it is not already there. This produces a new graph $\widetilde{C}$. For a 2-connected graph G, the graphs $\widetilde{C}$ obtained in this way which are not cycles are called the 3-connected components of G. Tutte proved that every 2-connected graph has a decomposition into a tree whose nodes are cycles or 3-connected components.

Surfaces and Graph Embeddings. We now present some fundamental properties of surfaces and embedded graphs. We refer to [10] for background.

A *surface* is a compact connected Hausdorff topological space in which every point has a neighbourhood homeomorphic to the plane. A surface can be constructed from the sphere by cutting a number of holes and pairs of holes into the sphere, each homeomorphic to an open disc. Every pair of holes is then closed by adding a *handle* – an open cylinder – connecting the boundary of the holes. The remaining holes are closed by adding a *crosscap*, that is, by identifying each point on the boundary of the hole with the corresponding point on the opposite side. The surface classification theorem shows that every surface is homeomorphic to a surface constructed in this way. Any surface obtained in this way that includes a crosscap is called *non-orientable*, otherwise it is called *orientable*. Our main result holds for orientable and non-orientable surfaces. Due to space constraints, we only explain the orientable case. The surface obtained from the sphere by adding k handles is denoted by $\mathbb{S}_k$. The number k is called the *genus* of $\mathbb{S}_k$.

Let $S = \mathbb{S}_k$ be a surface. The way we constructed S implies that we can reduce it to the plane by taking for each handle a closed curve that goes around the handle and cut the surface along the curves, closing the appearing holes by disks. This way, every handle is cut open and the resulting surface is homeomorphic to the plane. We call curves which cut a handle in this way *noncontractible*. There are two types of non-contractible curves: if we cut along a curve, we may either disconnect the surface or not. Curves which do not disconnect the surface when we cut along them are called *non-separating*.

Following [10], a graph G' is *embedded* on a surface S if its vertices are distinct points on S and every edge e of G' with endpoints u, v is a simple closed arc connecting u and v in S such that the interior of e is disjoint from other edges and vertices of G'. A graph G is

embeddable on S if it is isomorphic to an embedded graph G' on S. Let Π be an embedding of a graph G in S. The connected components of $S - \Pi$ are called the *faces* of Π.

An embedding of a graph in a surface S can be uniquely represented by an object called an *embedding scheme* or *rotation system*. If v is a vertex of G then the embedding of G embeds the edges incident with v in some order in clockwise orientation around v. That is, for any v the embedding defines a cyclic permutation of the incident edges, or a linear order on the edges that have v as an endpoint. We call this order the *clockwise order* around v. A *rotation system* is a set of clockwise orderings containing one order for every vertex.

Such a rotation system uniquely determines an embedding of G in S. Given a rotation system π of a graph G, we can construct the facial cycles of the embedding as follows. Let v be a vertex and let $e = \{u, v\}$ be an incident edge. Then v and e determine a walk in G where we start at v, follow the edge e to its other endpoint and then proceed with the next edge in clockwise order until we return to v. In this way we obtain exactly the facial cycles of the embedding. In particular, we can represent any face of the embedding by orienting an edge in its boundary cycle, which fixes the start vertex v and the edge e to choose first. This will be used in the MSO definition in the next section.

One of the main challenges in proving our main result is that we need to define MSO-formulas that encode in a given graph an embedding of it in a fixed surface S. Whitney proved that a planar graph G that is 3-connected has a unique embedding into the plane. In [3], Courcelle proved that there are MSO-formulas defining the rotation system for planar graphs.

To prove that rotation systems are also definable for graphs on other surfaces we will reduce the problem to the planar case as follows. As mentioned above, any surface can be reduced to the plane by cutting the surface along a finite set of closed curves each of which cuts a handle. We can generalize this concept of *cutting along a cycle* to cycles in embedded graphs.

In the sequel, let S be a surface. If we refer to a Π-*embedded graph* G we implicitly define Π to be an embedding of the graph G on S. Let $C = (v_0 e_1 v_1 e_2 .... v_k = v_0)$ be a cycle in G. For $i > 0$, let L_i be the set of edges incident to v_i which in the clockwise ordering around v_i appear after e_i but before e_{i+1} the R_i be the set of edges appearing after e_{i+1} and e_i. The edges in any L_i are called the *left* edges of the cycle C and the edges in any R_i are called the *right* edges of C.

▶ **Definition 2.1.** Let $C = (v_0 e_1 v_1 \dots v_k)$ be a surface nonseparating cycle of a Π-embedded graph G. Let G' be the graph obtained from G by replacing C by two isomorphic copies $C_r = (v_0^r e_1^r v_1^r \dots v_k^r)$ and $C_l = (v_0^r e_1^l v_1^l \dots v_k^l)$ such that all edges $e = \{u, v_i\}$ on the right of C are replaced by edges $\{u, v_i^r\}$ incident to the corresponding vertices on C_r and all edges $e = \{u, v_i\}$ on the left of C are replaced by edges $\{u, v_i^l\}$ incident to the corresponding vertices on C_l. We say that G' is obtained from G by *cutting along the cycle* C. The embedding Π defines an embedding Π' of G' in the surface obtained from S by cutting along C and closing the resulting holes by discs in the obvious way. Note that the two copies C_1, C_2 of C are now facial cycles. We now add two new vertices f_1, f_2 such that f_i has an edge to every vertex of C_i. This means that f_i is drawn in the face bounded by C_i. We call the resulting graph the *augmented graph obtained by cutting along* C.

This motivates the following definition.

▶ **Definition 2.2** (Planarizing set of cycles)**.** Let G be a Π-embedded graph. A *planarizing set of cycles* is a set $C_1, \dots, C_k$ of pairwise disjoint cycles in G such that cutting along all cycles $C_1, \dots, C_k$ results in a connected graph embedded in the sphere.

We will see below that for certain graphs embedded on a surface such sets of cycles always exist. For this, we need the concept of face-width.

▶ **Definition 2.3.** Let Π be an embedding of a graph G on S. The *face-width* $\mathrm{fw}(G,\Pi)$ of G is the smallest number k such that S contains a noncontractible closed curve that intersects G in k points, or ∞ if no noncontractible curve exists (i.e. S is the plane).

We prove next that the connectivity between two Π-noncontractible and Π-nonseparating cycles is at least as high as the face-width of Π.

▶ **Lemma 2.4.** *Let G be a 2-connected Π-embedded graph and let C and C' be two disjoint Π-noncontractible, Π-nonseparating cycles. Let $2 \leq k = \mathrm{fw}(G,\Pi) < \infty$. Then there are at least k pairwise vertex disjoint paths linking C and C'. Furthermore, if G is 3-connected with $3 \leq \mathrm{fw}(G,\Pi) < \infty$ and C is a Π-nonseparating cycle then be the augmented graph $\widetilde{G}$ obtained from G by cutting along C is 3-connected.*

See [10, Theorem 5.11.2] for a proof of the following theorem.

▶ **Theorem 2.5.** *Let G be a graph that is Π-embedded in a surface with Euler genus g and let d be a positive integer. If $\mathrm{fw}(G,\Pi) \geq 8(d+1)(2^g-1)$, then G contains a planarizing collection of induced cycles $C_1,\ldots,C_k$, for some $g/2 \leq k \leq g$, such that the distance between C_i and C_j, $i \neq j$, is at least d.*

The previous theorem and the lemma preceding it show that starting from a graph that is 3-connected and has an embedding of face-width at least 3, we can reduce it to a 3-connected planar graph by cutting along a constant number of cycles. Combining this with Whitney's theorem of unique embeddings of 3-connected planar graphs implies the following result which will be important later.

▶ **Theorem 2.6.** *Let G be a 3-connected graph that is Π-embedded in a surface with Euler genus g such that $3 \leq 8(d+1)(2^g-1) \leq \mathrm{fw}(G,\Pi) < \infty$. Let G' be the augmented graph obtained by cutting along a set of planarizing cycles. Then the facial cycles of Π are precisely the facial cycles of the unique plane embedding of G'. Moreover, an embedding scheme which is equivalent to Π can be deduced from the rotation system of the plane embedding of G'.*

The following definition captures the abstract properties of planarizing sets of cycles.

▶ **Definition 2.7.** Let G be a graph and let $k \geq 0$. A *potential system of planarizing cycles of order k* is a sequence $\big((C_i,L_i,R_i)\big)_{i=1}^k$ such that $C_1,\ldots,C_k$ are pairwise vertex disjoint cycles in G and L_i,R_i form a partition of the set of edges $e \notin E(C_i)$ incident with a vertex on C_i, for all i.

Note that the procedure of cutting along a cycle in a graph G and the augmented graph obtained from G in this way as defined above can be applied to any cycle C with a given partition L,R of the edges $e \notin E(C)$ incident with a vertex on C. Of course it may not always lead to the intended effect, e.g. if the sets L and R of left and right edges are chosen wrongly. But in any case it will produce a graph G' and if G and Π satisfy the conditions of Theorem 2.5, then G' will be the augmented graph obtained by cutting along the cycles $C_1,\ldots,C_k$.

Let G' be the graph obtained in this way. We call G' the graph *obtained from G by cutting along $\mathcal{S}$*. If G' is planar and Π' is a plane embedding of G' then we can obtain an embedding Π'' of G on some surface as follows. Let $f_1,\ldots,f_k$ be the extra vertices in G'. If we delete $f_1,\ldots,f_k$ from G' and Π' then we obtain a plane embedding Π'' which, for each i,

has two facial cycles corresponding to the two copies of C_i. We cut two holes, each having a copy of C_i as boundary cycle, and then identify the corresponding points on the two copies of C_i in the obvious way. In this way we obtain a surface S defined by an embedding Π''' of G. We call S, Π''' the surface and embedding *obtained from* G' *and* Π' *by* gluing *along* $\mathcal{S}$.

The results and constructions proved in this subsection imply the next theorem.

▶ **Theorem 2.8.** *Let S be a surface of genus g and let G be a 3-connected graph embedded in S by an embedding Π such that $8(d+1)(2^g-1) \leq \mathrm{fw}(G,\Pi) < \infty$.*

1. *Then, there exists a potential system of planarizing cycles $\mathcal{S}$ of order at most g such that the graph G' obtained from G by cutting along $\mathcal{S}$ is 3-connected and planar, and if Π' is the (uniquely determined) plane embedding of G', then S and Π are the surface and the embedding obtained from G' and Π' by gluing along $\mathcal{S}$.*
2. *Furthermore, for every potential system of planarizing cycles $\mathcal{S}$ of order at most g such that the graph G' obtained from G by cutting along $\mathcal{S}$ is 3-connected and planar, if Π' is the (uniquely determined) plane embedding of G', then S and Π are the surface and the embedding obtained from G' and Π' by gluing along $\mathcal{S}$.*

This theorem is the main tool for defining embeddings in monadic second-order logic later on. In the next section we exploit face-width for finding grids of controlled size.

Grid-like Structures in High-Face Width Embeddings. In this section we establish some graph theoretical properties of embedded graphs that will allow us to define grids in embedded graphs whose order is proportional to the face-width.

▶ **Definition 2.9.** Let G be a 2-connected Π-embedded graph such that $\mathrm{fw}(G,\Pi) \geq 2$ and let $F = F(G,\Pi)$ be the set of Π-faces. The *vertex-face graph* is the bipartite graph $\Gamma = \Gamma(G,\Pi)$ with vertex set $V(G) \cup F(G,\Pi)$ and an edge between $u \in V(G)$ and $f \in F(G,\Pi)$ if u is contained in the facial cycle bounding f.

Note that any closed curve on a surface corresponds to a cycle in the vertex-face graph in the obvious way. From now on, we will therefore consider cycles in the $\Gamma(G,\Pi)$. The length of such a cycle is the number of faces (or vertices) on it.

▶ **Lemma 2.10** (Prop. 5.5.10 of [10])**.** *Let G be a Π-embedded graph such that $2 < \mathrm{fw}(G,\Pi) < \infty$, and let $k := \lfloor \frac{\mathrm{fw}(G,\Pi)}{2} \rfloor - 1$. Let v be a Π-face and let $B_0(v)$ be the Π-boundary walk of v. For $i > 0$ we define $B_i(v)$ as the union of $B_{i-1}(v)$ and all Π-facial walks that have a vertex in $B_{i-1}(v)$. Then there exist $k+1$ disjoint Π-contractible cycles $C_0, \ldots, C_k$ such that for all $i = 0, 1, ..., k$, $C_i \subseteq \partial B_i(v)$ and $B_i(v) \subseteq Int(C_i)$.*

We are now ready to prove the graph theoretical properties we will use later to show that any 3-connected graph embedded by a face-width k embedding, for some $3 \leq k < \infty$, contains a $\frac{k}{4} \times \frac{k}{4}$-grid which, moreover, is *controlled* by a noncontractible cycle.

Let Π be an embedding of a graph G on a surface S of Euler genus g such that $\mathrm{fw}(G,\Pi) \geq 3$. Let $\Gamma = \Gamma(G,\Pi)$. Let $k := \lfloor \frac{\mathrm{fw}(G,\Pi)}{2} \rfloor - 1$.

▶ **Definition 2.11.** Let C be a Π-nonseparating, Π-non-contractible cycle of Γ of minimal length. Hence, the number of vertices and faces on C is exactly $\mathrm{fw}(G,\Pi)$. Let v be a face on C. A set $\mathcal{P} = \{P_0, \ldots, P_k\}$ of pairwise vertex disjoint cycles is *controlled by C and v*, if for all $0 \leq i \leq k$,

- P_i intersects C in exactly two vertices p_i, p'_i,
- for all $1 \leq i \leq k$, p_i and p_{i-1} and also p'_i and p'_{i-1} have a common neighbour on C, which is a face, and p_0 and p'_0 are the neighbours of the face v on C, and
- if $i < j$ then P_i is contained in the component of $G - P_j$, called the *interior* of P_j, that contains v.

Let C and v be as before and let $\mathcal{P} = \{P_0, \ldots, P_k\}$ be a set of pairwise vertex disjoint cycles controlled by C and v. Then either there is exactly one face $\bar{v}$ on C not in the interior of P_k or there is exactly one vertex $\bar{v}$ on C which is neither in the interior of P_k nor on P_k itself. We call the node $\bar{v}$ in the previous claim the *opposite node of v on C* and denote it by $\bar{v}$.

Let $C, v, \bar{v}$ and $\mathcal{P}$ be as before. Let a be a face on C and let $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ be a set of pairwise disjoint cycles controlled by C and a. Let $\bar{a}$ be the opposite node of a on C.

Let $s := \lceil \frac{k+1}{2} \rceil$ and let $s' := k - s$. If $\mathrm{fw}(G, \Pi)$ is odd then a is a face of facial distance s from v on C if, and only if, the cycle $Q \in \mathcal{Q}$ which contains the vertex $\bar{v}$ has v on its exterior but v is adjacent to a vertex on Q. If $\mathrm{fw}(G, \Pi)$ is even then a is a face of facial distance s from v on C if, and only if, v and $\bar{v}$ are adjacent to vertices on the same two cycles in $\mathcal{Q}$. If a and v satisfy these conditions, then we say that v and a *match*.

This observation will allow us to define $a, \bar{a}, \bar{v}$ from C and v. What is left is to give a topological condition for C to be a noncontractible cycle of minimal length. But this can easily be done using Lemma 2.4. Finally, it can be shown that $C, v, \bar{v}, a, \bar{a}$ uniquely determine a grid structure in $C \cup \mathcal{P} \cup \mathcal{Q}$ of size s'. The previous claims together establish the following theorem.

▶ **Theorem 2.12.** *Let C be a noncontractible, nonseparating cycle of length $\mathrm{fw}(G, \Pi)$ and let v be a face on C. Let $\mathcal{P}$ be a set of pairwise disjoint cycles controlled by C and v and let $\bar{v}$ be the node on the opposite of v on C. Let x be a face on C and let $\mathcal{Q}$ be a set of pairwise disjoint cycles controlled by C and x and let $\bar{x}$ be the node opposite of x on C. Finally, suppose v and x match as defined in the previous claim. Then $C, v, x, \bar{x}, \bar{v}$ determine an $s' \times s'$ grid.*

3 Monadic Second-Order Logic

In this section we introduce monadic second-order logic (MSO), and develop the MSO definability of embedding schemes and grids in large face-width embeddings.

Logic. A *vocabulary* is a set of relation symbols with associate arities. If L is a vocabulary, a *finite L-structure M* is given by a finite *domain* or *universe* $D(M)$ and a relation $R(M) \subseteq M^r$ for each relation symbol $R \in L$ of arity r

The class of formulas of monadic second-order logic (MSO) is the smallest class of formulas that contains the atomic formulas and is closed under negations, conjunctions, disjunctions, and existential and universal quantification of individual and set variables. An individual variable ranges over the domain, and a set variable ranges over the subsets of the domain. If X is a set variable and x is an individual variable, then $X(x)$ is the atomic formula that says that x is in the set X. Given a structure M, the relation defined by a formula $\varphi(\bar{x}, \bar{X})$ with individual and set free-variables $\bar{x}$ and $\bar{X}$, respectively, is the set of pairs of tuples $(\bar{a}, \bar{A})$ such that M makes $\varphi(\bar{a}, \bar{A})$ true. A formula without free variables is called a *sentence*.

Logic on Graphs, MSO_1 and MSO_2. There are two natural encodings of graphs G as structures. In the *standard encoding* the vocabulary has a single binary relation symbol E, the domain of the structure is $V(G)$, and the binary relation E is interpreted by $E(G)$. In the *incidence encoding* the vocabulary has two unary relation symbols V and E, one binary relation symbol I, the domain is $V(G) \cup E(G)$, and I is the incidence relation between vertices and their incident edges. The unary relations V and E are interpreted by $V(G)$ and $E(G)$. Whereas for FO it is irrelevant which encoding is used, the two encodings behave

differently for MSO: on the standard encoding the set quantifiers range over sets of vertices whereas on the incidence encoding they range also over sets of edges, e.g., over paths. MSO on the standard encoding is often referred to as MSO_1 and on the incidence encoding it is referred to as MSO_2. For graphs of bounded genus, and even more generally, for classes of graphs that are p-degenerate for some fixed $p > 0$, the two logics MSO_1 and MSO_2 are equivalent: see Theorem 5.22 in [5]. Since we only consider classes of graphs of bounded genus we will assume from now on that graphs are given by their incidence encoding.

MSO-definitions and interpretations. For an integer $k \geq 1$, an MSO-definition of L-structures of *order* k is a collection of formulas $\varphi_{D,1}(x), \ldots, \varphi_{D,k}(x)$ and $\varphi_{R,t}(\bar{x})$ for $R \in L$ and $t \in [k]^r$, where r is the arity of R, and x in $\varphi_{D,i}(x)$ is an individual variable, and $\bar{x}$ in $\varphi_{R,t}(\bar{x})$ is a tuple of individual variables of length r. If Ψ is such an MSO-definition and M is a structure of the vocabulary of the formulas in Ψ, then the structure defined by Ψ on M is the L-structure N with $D(N) = \{(a,i) \in D(M) \times [k] : M \models \varphi_{D,i}(a)\}$ and $R(N) = \{((a_1,i_1),\ldots,(a_r,i_r)) \in (D(M) \times [k])^r : M \models \varphi_{R,(i_1,\ldots,i_r)}(a_1,\ldots,a_r)\}$. An MSO-definition of order k with parameters is defined analogously, with each formula carrying additional parameter variables $\bar{z}$ and $\bar{Z}$, and an additional formula $\pi(\bar{z},\bar{Z})$ to tell if a choice of parameters $\bar{b},\bar{B}$ is *good*. We say that Ψ takes the structure M as input and produces the structure N as output, for the good choice of parameters $\bar{b}$ and $\bar{B}$, if the defining formulas produce N when $\bar{z}$ and $\bar{Z}$ are replaced by the parameters. If there is as least one good choice of parameters in M and the same structure N is produced under all good choices of parameters, then we omit any reference to them and say that Ψ takes M as input and produces N as output.

Finally, MSO-interpretations extend MSO-definitions to allow *factor structures*. Concretely, an MSO-interpretation (without parameters, of order 1) includes an additional *equality-defining* formula $\varphi_{\equiv}(x,y)$, that is required to define an equivalence relation on the domain defined by $\varphi_D(x)$ that is a congruence of the relations defined by the $\varphi_R(\bar{x})$'s. On a structure M as input, the MSO-interpretation produces the structure whose domain is the set of equivalence classes of the equivalence relation $\equiv$ defined by $\varphi_{\equiv}(x,y)$ on the domain defined by $\varphi_D(x)$, and whose relations are the relations that are defined by the $\varphi_R(\bar{x})$'s factored by $\equiv$. MSO-interpretations with parameters and of order $k > 1$ are defined analogously.

The composition of two MSO-interpretations is defined in the obvious way and is again an MSO-interpretation. As an example of MSO-interpretation we state the following easily derived consequence of Theorem 4.7 in [3].

▶ **Theorem 3.1.** *There is an MSO-definition Ψ that, for every graph G given as input and every 3-connected component C of G, there is a good choice of parameters for Ψ on which it produces C.*

Logic Representation of Embedding Schemes. Embedding schemes will be represented as graphs expanded by two relations that represent the cyclic orderings around the vertices. Concretely, the vocabulary has two unary relation symbols V and E for vertices and edges, one binary relation symbol I for the incidence relation between vertices and edges and one ternary relation symbol R for cyclic orderings. The predicate $R(v,e_1,e_2)$ holds if e_1 and e_2 are edges that are incident to vertex v, and e_1 is the immediate predecessor of e_2 in the cyclic ordering of the edges that are incident to v.

This encoding was used by Courcelle [4] to show that there is an MSO-definition that takes a 3-connected planar graph as input and produces an embedding in the plane as output, which is unique by Whitney's Theorem.

▶ **Theorem 3.2** ([4]). *There is an MSO-definition that, given a 3-connected planar graph G as input, produces an embedding scheme as output.*

It will be convenient to extend the embedding schemes to include the faces of the embedding. Accordingly, extended embedding schemes will include a set F of faces in their domain and store the incidence relation between edges and faces through the incidence relation I.

We aim at an MSO-interpretation that given an embedding scheme produces its extension with faces. A face is determined by one of its bounding edges together with an orientation. Specifying the orientation directly by specifying one of its endpoints would take us beyond the syntax of MSO-interpretations. Instead of that, we use a proper k-coloring of the graph with a small number k of colors and specify the orientation of the edge by specifying the colors of the left and right endpoints. The k-coloring is provided through k many parameter set variables, and we choose k large enough so that any graph in the surface under consideration has chromatic number at most k. It is well-known that the chromatic number of all graphs embeddable in a fixed surface is bounded (see [10]). In the case of planar graphs $k = 5$ suffices (and even $k = 4$ does).

▶ **Lemma 3.3.** *For every $k \geq 2$, there is an MSO-interpretation that, given an embedding scheme for a graph that is k-colorable, produces its extension by faces as an extended embedding scheme.*

Embeddings by Reduction to Planar Case. Our next goal is to prove the analogue of Theorem 3.2 for higher genus surfaces. We proceed by reduction to the planar case via Theorem 2.8. Let S be an orientable surface of genus g. For simplicity we start with the orientable case.

We start with two MSO-interpretations that implement the operations of cutting a graph along a potential system of planarizing cycles, and its reverse operation of gluing along it; cf. Definition 2.7 and the discussion immediately following it. Systems are represented most simply by a sequence of $3k$ set variables.

▶ **Lemma 3.4.** *There is an MSO-interpretation Ξ that, given a graph G and potential system of planarizing cycles $\mathcal{S}$, produces the graph G' obtained from G by cutting along $\mathcal{S}$. Conversely, there is an MSO-interpretation Ξ' that, given a graph G, a potential system of planarizing cycles $\mathcal{S}$, and a plane embedding scheme Π' for the output of Ξ on G and $\mathcal{S}$, produces the graph G'' and the embedding scheme Π'' that is obtained from G' and Π' by gluing along $\mathcal{S}$.*

With these objects at hand we are ready to prove the analogue of Theorem 3.2.

▶ **Theorem 3.5.** *Let S be a surface of genus g. There is an MSO-interpretation that, given a 3-connected graph G that has an embedding in S of finite face-width at least $8(d+1)(2^g-1)$ (which must be unique), produces such an embedding Π.*

We need to show that the conditions of Theorem 2.8 can be defined in MSO.

Defining Grids in High Face-Width Embeddings. Our final goal of this section is to develop an MSO-interpretation γ which defines large grids in 3-connected Π-embedded graphs.

▶ **Theorem 3.6.** *There is an* MSO*-interpretation γ that, given an extended embedding scheme for a graph G of finite face-width $k \geq 3$, produces a grid of order $\lfloor \frac{k}{4} \rfloor$.*

To define γ, we need to show that the conditions of Theorem 2.12 can be defined in MSO.

4 Size and face-width for random graphs

Fix a surface S of genus $g \geq 0$. It was shown in [2] that, a.a.s., a random graph G from $\mathcal{G}_S$ has genus g, and in particular it is not planar if $g > 0$. Moreover, a.a.s., G has a unique non-planar connected component, as well as a unique non-planar 2-connected and 3-connected components. Moreover, these components are all *giant*, i.e., of size linear in the number of vertices. Indeed the probability distributions of their sizes is well-understood. In the following, if f_n and g_n are sequences of positive real numbers, we use the notation $f_n \sim g_n$ to mean that $\lim_{n\to\infty} f_n/g_n = 1$.

Sizes of the components. Let L_n denote the size (i.e., number of vertices) of the largest connected component in a random n-vertex graph G from $\mathcal{G}_S$, and let $M_n = n - L_n$. Theorem 5.3 in [2] determines the distribution of M_n, and hence of L_n: for every fixed integer $k \geq 0$ we have $\Pr[M_n = k] \sim p \cdot g_k \frac{\gamma^{-k}}{k!}$ where p, g_k and γ are constants that do not depend on n, nor on the surface S. Moreover there exists constants $a > 0$ and $b > 0$ such that $\mathrm{E}(M_n) \sim a$ and $\mathrm{Var}(M_n) \sim b$. In particular, by Chebyshev's inequality this means that, for any $a(n)$ that grows to infinity however slowly, we have $M_n \leq a(n)$ a.a.s., and hence $L_n \geq n - a(n)$ a.a.s. Theorem 5.4 and 5.5 also in [2] determine the distributions of the sizes of the largest 2-connected and 3-connected components, but only for random *connected* graphs from $\mathcal{G}_S$. However, by composing the results it is still possible to determine the sizes for random arbitrary graphs from $\mathcal{G}_S$, as we do next.

A sequence of integer random variables $X_0, X_1, \ldots$ is said to admit a local limit law of the Airy type with parameters α and c if for every real finite interval $[a, b]$ it holds that $\Pr\left[X_n = \lfloor \alpha n + x n^{2/3} \rfloor\right] \sim n^{-2/3} c g(cx)$ uniformly for every $x \in [a, b]$, where $g(x) = 2e^{-2x^3/3}(x\mathrm{Ai}(x^2) - \mathrm{Ai}'(x^2))$ and $\mathrm{Ai}(x)$ is the Airy function. Here, uniformly for every $x \in [a, b]$ means that for every positive real $\varepsilon > 0$ and every large enough n the ratio is ε-close to 1 simultanesouly for all $x \in [a, b]$.

▶ **Theorem 4.1.** *Let X_n and Y_n denote the sizes of the largest 2-connected and 3-connected components of a random n-vertex graph in $\mathcal{G}_S$. Then X_n and Y_n admit local limit laws of the Airy type (with different parameters). Moreover, a.a.s., the largest connected, 2-connected and 3-connected components are unique and have maximal possible genus, and all other connected, 2-connected and 3-connected components are planar.*

Face-width of the components. For this section we assume that the genus of S is $g > 0$. Our goal is to show that the face-width of the largest 3-connected component of the random graph grows logarithmically. We will need the following facts:

Almost all 3-connected maps on S with m edges have face-width greater than $\delta \log m$ for some constant $\delta > 0$. This is proved in [1] for rooted maps. Since almost all 3-connected maps have no non-trivial automorphisms [12] this is also true for unrooted maps. Moreover, the largest 3-connected component of a random graph G from $\mathcal{G}_S$ is unique and non-planar by Theorem 4.1 a.a.s., and has face-width greater than any fixed constant also a.a.s. [2]. As a consequence it has a unique embedding in S, and it can be considered as an unrooted map [13].

▶ **Theorem 4.2.** *Let S be a surface of genus $g > 0$ and let F_n denote the face-width of the largest 3-connected component of a random n-vertex graph in $\mathcal{G}_S$. Then there exists γ such that $F_n \geq \gamma \log n$ asymptotically almost surely.*

Proof. Let G denote a random n-vertex graph in $\mathcal{G}_S$. Let L and T denote the largest 2-connected and 3-connected components of G. We start by arguing that, conditioned on the number of edges of L and T, the distribution of T is uniform over the 3-connected graphs in $\mathcal{G}_S$ with its number of edges. Indeed, L is obtained from T by (possibly) replacing each edge of T with a 2-connected graph, so each T with m edges gives rise to the same number L with k edges. For $m \geq k$, let $B_{m,k}$ denote the event that $e(T) = m$ and $e(L) = k$. By Theorem 4.1, the sizes of T and L are at least cn a.a.s., for some constant $c > 0$. Since they are at least 2-connected, also $e(T) \geq cn$ and $e(L) \geq cn$ a.a.s.. Now we combine the previous paragraph with the one just before the theorem: In the distribution conditioned on $B_{m,k}$, the 3-connected component T can be considered as a random m-edge 3-connected map embedded in S, and its facewidth is at least $\delta \log m$ if m is large enough. We conclude that the facewidth of T is indeed at least $\delta \log(cn)$ a.a.s. Precisely, if A denotes the event that T has facewidth at least $\delta \log(cn)$, then $\Pr[\, A \,] \geq \sum_{m \geq k \geq cn} \Pr[\, A \mid B_{m,k} \,] \Pr[\, B_{m,k} \,]$. For fixed $\varepsilon > 0$, if n is large enough, then $\Pr[\, A \mid B_{m,k} \,] \geq 1 - \varepsilon$ for any $m \geq k \geq cn$. It follows that, if n is large enough, then $\Pr[\, A \,] \geq (1-\varepsilon) \sum_{m \geq k \geq cn} \Pr[\, B_{m,k} \,] \geq (1-\varepsilon)^2$. Since $\varepsilon > 0$ was arbitrary, the claim is proved by choosing any $\gamma > 0$ smaller than δ. ◀

5 Limiting probabilities of MSO-sentences

In this section we put everything together. Let S be a surface of genus $g > 0$. The results so far show that a random n-vertex graph in $\mathcal{G}_S$ will have facewidth $\Omega(\log n)$ with high probability, and that on such graphs an $m \times m$ grid is MSO-definable, for some $m \geq \log \log n$. More precisely:

▶ **Theorem 5.1.** *Let S be a surface other than the sphere. There is an MSO-interpretation that, on a given n-vertex graph G from $\mathcal{G}_S$, produces an $m \times m$ grid for some $m \geq \log \log n$ for every and at least one good choice of parameters, asymptotically almost surely when G is a random n-vertex graph in $\mathcal{G}_S$.*

We use this to build MSO-sentences with non-converging probabilities. We use it also for proving the undecidability of the problem of determining the asymptotic probabilities of MSO-sentences. For building MSO-sentences whose probabilities converge to any given rational number in the interval $[0, 1]$ we use Theorem 4.1 from the previous section.

MSO-sentences with non-converging probabilities. For every base $b \geq 2$ and every natural number n, define $\mathrm{tow}_b(n)$ recursively by $\mathrm{tow}_b(0) = 1$ and $\mathrm{tow}_b(i+1) = b^{\mathrm{tow}_b(i)}$. For every real $x \geq 0$, let $\log_b^*(x)$ denote the smallest integer k such that $\mathrm{tow}_b(i) \geq x$, and let $\log_b^\dagger(x)$ denote the smallest integer k such that $\sum_{i=0}^{k} \mathrm{tow}_b(i) \geq x$. By induction on k one proves that $\sum_{i=0}^{k} \mathrm{tow}_b(i) \leq 2\mathrm{tow}_b(k)$, and hence $\log_b^*(x/2) \leq \log_b^\dagger(x) \leq \log_b^*(x)$ for every $b \geq 2$ and every real $x \geq 0$. Both $\log_b^*(n)$ and $\log_b^\dagger(n)$ are monotone non-decreasing functions of n that have all natural numbers in their range. When the base is 2 we omit b from the notation.

The source of divergence in our example is the following easily verified arithmetic fact:

▶ **Lemma 5.2.** *If $m_1, m_2, \ldots$ is an integer sequence such that $\log \log(n) \leq m_n \leq n$ for every large enough n, then the sequence given by $\log^\dagger(m_n) \bmod 8$ is not eventually always in $\{0,1,2,3\}$ and not eventually always in $\{4,5,6,7\}$.*

For the next technical lemma it will be more convenient to move, temporarily, to the vocabulary of directed grids. For every $i \in \{0, \ldots, 7\}$, we want an MSO-sentence $\mathtt{strange}_i$ that holds in the $n \times n$ directed grid if and only if $\log^\dagger(n)$ is congruent to i mod 8. We use the notation $G^{\mathrm{d}}_{n \times n}$ to denote the $n \times n$ directed grid.

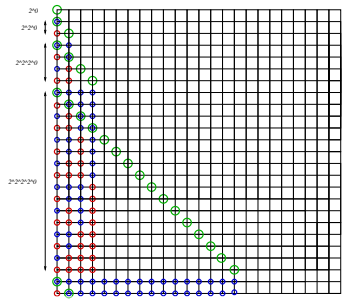


Figure 5.1 The red-green-blue pattern for the proof of Lemma 5.3.

▶ **Lemma 5.3.** *For every $i \in \{0,\dots,7\}$ there exists an MSO-sentence $\texttt{strange}_i$ in the vocabulary of directed grids such that, for every natural number $n \geq 1$, the sentence $\texttt{strange}_i$ is true in $G^{\mathrm{d}}_{n\times n}$ if and only if $\log^\dagger(n)$ is congruent to i mod 8.*

Proof. The sentence uses three existentially quantified monadic second-order variables R, G, and B, for red, green and blue. First it verifies that the colors satisfy the pattern of Figure 5.1. Once this is verified, the sentence states that the number of green vertices in the leftmost column is congruent to $i+1$ mod 8. The number of green vertices in the leftmost column is the smallest k such that $\sum_{i=0}^{k-1} \mathrm{tow}(i) \geq n$; i.e. $k = \log^\dagger(n)+1$, so the statement states that $\log^\dagger(n)+1 \equiv i+1 \bmod 8$, which is the same as $\log^\dagger(n) \equiv i \bmod 8$. ◀

We need to find a way of defining directed grids in the $\{\texttt{right},\texttt{down}\}$-vocabulary from undirected ones. One way to do this via MSO-interpretations can be extracted from Section 5.2.3 in [5].

▶ **Theorem 5.4.** *Let S be a surface other than the sphere. There is an MSO-sentence whose asymptotic probability on $\mathcal{G}_S$ does not converge.*

Proof. Let Θ be the composition of Ψ from Theorem 5.1 with interpretation that produces directed grids from undirected ones. For every $T \subseteq \{0,\dots,7\}$, let φ_T say that there is a good choice of the parameters for Θ that make it define a directed square grid on which the disjunction $\bigvee_{i\in T} \texttt{strange}_i$ holds. If $\varphi_{\{0,1,2,3\}}$ has probability 0, then $\varphi_{\{4,5,6,7\}}$ has probability 1 by Lemma 5.3. Let $T \in \{\{0,1,2,3\},\{4,5,6,7\}\}$ be such that φ_T does not have asymptotic probability 0.

We claim that the asymptotic probability of φ_T does not converge. Otherwise, it converges to a positive real, and a positive fraction of the n-vertex graphs in $\mathcal{G}_S$ satisfy φ_T. Therefore, for n there is a least one n-vertex graph in $\mathcal{G}_S$ for which there is a good choice of parameters that makes Ψ define a directed $m\times m$ grid with $\log^\dagger(m)$ mod 8 in T and such that at the same time $m \geq \log\log n$. In other words, we find a sequence m_n that contradicts Lemma 5.2. ◀

Undecidability of the decision problem. We proceed by reduction from the halting problem for Turing machines, which is of course undecidable. For every 1-tape Turing machine M we want an MSO sentence $\texttt{halts}_M$ of the vocabulary of directed grids, that holds in the $n \times n$ directed grid if and only if the computation of M on the empty input halts in time at most n using space at most n. This construction is standard.

▶ **Lemma 5.5.** *For every two-sided 1-tape Turing machine M there exists an MSO-sentence $\texttt{halts}_M$ of the vocabulary of directed grids such that, for every natural number $n \geq 1$, the sentence halts_M is true in $G^{\mathrm{d}}_{n\times n}$ if and only if the computation of M on the empty input halts in time at most n using space at most n.*

We combine the non-converging formula of Theorem 5.4 with Lemma 5.5.

▶ **Theorem 5.6.** *Let S be a surface other than the sphere. The problem of determining whether a given MSO-sentence has a converging asymptotic probability on $\mathcal{G}_S$ is undecidable.*

Proof. Let φ be the MSO-sentence with non-converging asymptotic probability from Theorem 5.4. As in the proof of Theorem 5.4, let Ψ be the MSO-interpretation in Theorem 5.1, and let Θ be the composition of Ψ with the MSO-interpretation that produces a directed grid from an undirected one. For every two-sided 1-tape Turing machine M, let ψ_M be the sentence that says that there is a good choice of parameters for Θ that makes it define a square grid, and the sentence $\texttt{halts}_M$ holds on some principal square subgrid of this grid. It is easy to see that ψ_M has asymptotic probability one if M halts, and probability zero otherwise. This $\varphi \wedge \psi_M$ converges if and only if M halts. ◀

All rationals in [0,1] as limiting probabilities. Our goal in this section is to construct an MSO sentence whose asymptotic probability over $\mathcal{G}_S$, for any fixed surface S other than the sphere, converges to any given rational number in the interval $[0, 1]$. The idea of the construction is the following. Let us say that we want to achieve the rational p/q as limiting probability. Suppose that we succeed to write a sentence that says that the unique non-planar 3-connected component of the random n-vertex graph has size that is congruent to some $a \in \{0, \ldots, p-1\}$ mod q. If we do, then by Theorem 4.1 the probability that this sentence holds on the random n-vertex graph is the probability that an integer random variable that admits a local limit law of the Airy type is congruent to some $a \in \{0, \ldots, p-1\}$ mod q. It turns out that this probability approaches p/q as n approaches infinity:

▶ **Lemma 5.7.** *Let $X_0, X_1, \ldots$ be a sequence of integer random variables that admits a local limit law of the Airy type with parameters α and c. Then for every integer $q \geq 1$ and every $a \in \{0, \ldots, q-1\}$ it holds that $\lim_{n\to\infty} \Pr\left[X_n \equiv a \pmod q\right] = 1/q$.*

For saying that the unique non-planar 3-connected component has a size that is congruent to a mod q we use the fact every 3-connected graph of bounded Euler characteristic $\chi \leq 0$ has a spanning tree of degree at most $\lceil (8 - 2\chi)/3 \rceil$ [7, 11]. Thus, the unique non-planar 3-connected component, which is embeddable in the surface S, has such a spanning tree, which can be guessed in MSO_2. Once available, the spanning tree of bounded degree can be used to define a linear order on the vertices of the 3-connected component, and MSO over the linear order can say that its length is congruent to a mod q. Taking the disjunction over all $a \in \{0, \ldots, p-1\}$, the asymptotic probability of the resulting sentence will be p/q.

▶ **Theorem 5.8.** *Let S be a surface other than the sphere. For every rational number $r \in [0, 1]$, there exists an MSO sentence whose asymptotic probability on $\mathcal{G}_S$ converges to r.*

References

1. E. A. Bender, Z. Gao, L. B. Richmond, and N. Wormald. Asymptotic properties of rooted 3-connected maps on surfaces. *J. Austral. Math. Soc., Series A*, 60:31–41, 1996.
2. G. Chapuy, E. Fusy, O. Giménez, B. Mohar, and M. Noy. Asymptotic enumeration and limit laws for graphs of fixed genus. *J. Combin. Theory Ser. A*, 118:748–777, 2011.
3. B. Courcelle. The monadic second-order logic of graphs XI: Hierarchical decompositions of connected graphs. *Theoretical Computer Science*, 224:35–58, 1999.
4. B. Courcelle. The monadic second-order logic of graphs xii: planar graphs and planar maps. *Theoretical Computer Science (TCS)*, 237:1–32, 2000.

5 B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic*. Cambridge University Press, 2012.

6 R. Diestel. *Graph Theory*. Springer-Verlag, 4th edition, 2010.

7 M.N. Ellingham. Spanning paths, cycles and walks for graphs on surfaces. In *Congressus Numerantium*, volume 115, pages 55–90, 1996.

8 O. Giménez, M. Noy, and J. Rué. Graph classes with given 3-connected components: asymptotic enumeration and random graphs. *Random Structures Algorithms*, 42(4):438–479, 2013.

9 P. Heinig, T. Müller, M. Noy, and A. Taraz. Logical limit laws for minor-closed classes of graphs. *J. Combin. Theory Ser. B*, to appear.

10 B. Mohar and C. Thomassen. *Graphs on Surfaces*. Johns Hopkins University Press, 2001.

11 K. Ota and K. Ozeki. Spanning trees in 3-connected $K_{3,t}$-minor-free graphs. *J. Combin. Theory Ser. B*, 102(5):1179–1188, 2012.

12 L. B. Richmond and N. C. Wormald. Almost all maps are asymmetric. *J. Combin. Theory Ser. B*, 63(1):1–7, 1995.

13 N. Robertson and R. Vitray. Representativity of surface embeddings. In *Paths, flows, and VLSI-layout (Bonn, 1988)*, volume 9 of *Algorithms Combin.*, pages 293–328. Springer, Berlin, 1990.

Bisimulation Invariant Monadic-Second Order Logic in the Finite

Achim Blumensath[1]
Masaryk University Brno
blumens@fi.muni.cz

Felix Wolf[2]
Technische Universität Darmstadt, Institute TEMF
Graduate School of Excellence Computational Engineering
wolf@gsc.tu-darmstadt.de

Abstract

We consider bisimulation-invariant monadic second-order logic over various classes of finite transition systems. We present several combinatorial characterisations of when the expressive power of this fragment coincides with that of the modal μ-calculus. Using these characterisations we prove for some simple classes of transition systems that this is indeed the case. In particular, we show that, over the class of all finite transition systems with Cantor–Bendixson rank at most k, bisimulation-invariant MSO coincides with L_μ.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases bisimulation, monadic second-order logic, composition method

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.117

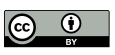
1 Introduction

A characterisation of the bisimulation-invariant fragment of a given classical logic relates this logic to a suitable modal logic. In this way, one obtains a correspondence between a family of classical logics and a family of modal logics. Such characterisation results therefore help with ordering the zoo of logics introduced (on both sides) over the years and with distinguishing between natural and artificial instances of such logics.

The study of bisimulation-invariant fragments of classical logics was initiated by a result of van Benthem [2] who proved that the bisimulation-invariant fragment of first-order logic coincides with standard modal logic. Inspired by this work, several other characterisations have been obtained. The table below summarises the results known so far.

bisimulation-invariant fragment	modal logic	reference
first-order logic	modal logic	[2]
monadic second-order logic	modal μ-calculus	[10]
monadic path logic	CTL*	[12, 13]
weak monadic second-order logic	continuous μ-calculus	[4]
weak chain logic	PDL	[4]

[1] Work supported by the Czech Science Foundation, grant No. GA17-01035S
[2] Work partially supported by the *Excellence Initiative* of the German Federal and State Governments and the Graduate School of Computational Engineering at Technische Universität Darmstadt



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 117; pp. 117:1–117:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

There are also similar characterisations for various variants of bisimulation like *guarded bisimulation* [1, 7] or bisimulation for *inquisitive modal logic* [5].

Researchers in finite model theory started to investigate to which extent these correspondences also hold when only considering *finite* structures, that is, whether every formula of a given classical logic that is bisimulation-invariant over the class of all finite transition systems is equivalent, over that class, to the corresponding modal logic. For first-order logic, a corresponding characterisation does indeed hold. Its proof by Rosen [15] uses tools from finite model theory and is very different to the proof by van Benthem.

The above mentioned result by Janin and Walukiewicz on bisimulation-invariant monadic second-order logic has so far defied all attempts at a similar transfer to the realm of finite structures. The main reason is that the original proof is based on automata-theoretic techniques and an essential ingredient is a reduction to trees, via the unravelling operation. As this operation produces infinite trees, we cannot use it for formulae that are only bisimulation-invariant over finite transition systems.

In this paper we start a fresh attempt at a finitary version of the result of Janin and Walukiewicz. Instead of automata-theoretic techniques we employ the composition method. For certain classes of very simple, finite transition systems we characterise the bisimulation-invariant fragments of monadic second-order logic over these classes. We hope that some day our techniques can be extended to the general case of *all* finite structures, but currently there are still a few technical obstacles to overcome.

We start in Section 2 by recalling the needed material on bisimulation and by listing all known results on bisimulation-invariant monadic second-order logic. We also collect some low-hanging fruit by proving two new results concerning (i) finite classes and (ii) the class of all finite trees. Finally, we lay the groundwork for the more involved proofs to follow by characterising bisimulation-invariance in terms of a combinatorial property called the *unravelling property.* In Section 3, we collect some tools from logic we will need. The emphasis in on so-called *composition lemmas.* Nothing in this section is new.

Finally we start in Section 4 in earnest by developing the technical machinery our proofs are based on. Sections 5 and 6 contain our first two applications: characterisations of bisimulation-invariant monadic second-order logic over (i) the class of lassos and (ii) certain classes of what we call *hierarchical* lassos. The former is already known and simply serves as an example of our techniques and to fix our notation for the second result, which is new.

Before presenting our last characterisation result, we develop in Section 7 some additional technical tools that allow us to reduce one characterisation result to another. This is then applied in Section 8 to the most complex of our results. We characterise bisimulation-invariant monadic second-order logic over the class of all transition systems of a given Cantor–Bendixson rank.

2 Bisimulation-invariance

We consider two logics in this paper: (i) *monadic second-order logic* (MSO), which is the extension of first-order logic by set variables and set quantifiers, and (ii) the *modal μ-calculus* (L_μ), which is the fixed-point extension of modal logic. A detailed introduction can be found, e.g., in [8]. Concerning the μ-calculus and bisimulation, we also refer to the survey [17]. *Transition systems* are directed graphs where the edges are labelled by elements of a given set A and vertices by elements of some set I. Formally, we consider a transition system as a structure of the form $\mathfrak{S} = \langle S, (E_a)_{a\in A}, (P_i)_{i\in I}, s_0\rangle$ where the $E_a \subseteq S \times S$ are (disjoint) binary edge relations, the $P_i \subseteq S$ are (disjoint) unary predicates, and s_0 is the initial state.

We write $\mathfrak{S}, s$ to denote the transition system obtained from $\mathfrak{S}$ by declaring s to be the initial state.

A central notion in modal logic is *bisimilarity* since modal logics cannot distinguish between bisimilar systems.

▶ **Definition 2.1.** Let $\mathfrak{S}$ and $\mathfrak{T}$ be transition systems.

(a) A *bisimulation* between $\mathfrak{S}$ and $\mathfrak{T}$ is a binary relation $Z \subseteq S \times T$ such that all pairs $\langle s,t\rangle \in Z$ satisfy the following conditions.

(prop) $s \in P_i^{\mathfrak{S}}$ iff $t \in P_i^{\mathfrak{T}}$, for all $i \in I$.

(forth) For each edge $\langle s,s'\rangle \in E_a^{\mathfrak{S}}$, there is some $\langle t,t'\rangle \in E_a^{\mathfrak{T}}$ such that $\langle s',t'\rangle \in Z$.

(back) For each edge $\langle t,t'\rangle \in E_a^{\mathfrak{T}}$, there is some $\langle s,s'\rangle \in E_a^{\mathfrak{S}}$ such that $\langle s',t'\rangle \in Z$.

(b) Let s_0 and t_0 be the initial states of, respectively, $\mathfrak{S}$ and $\mathfrak{T}$. We say that $\mathfrak{S}$ and $\mathfrak{T}$ are *bisimilar* if there exists a bisimulation Z between $\mathfrak{S}$ and $\mathfrak{T}$ with $\langle s_0,t_0\rangle \in Z$. We denote this fact by $\mathfrak{S} \sim \mathfrak{T}$.

(c) We denote by $\mathcal{U}(\mathfrak{S})$ the *unravelling* of a transition system $\mathfrak{S}$. ⌟

The next two observations show that the unravelling operation is closely related to bisimilarity. In fact, having the same unravelling can be seen as a poor man's version of bisimilarity.

▶ **Lemma 2.2.** *Let $\mathfrak{S}$ and $\mathfrak{T}$ be transition systems.*

(a) $\mathcal{U}(\mathfrak{S}) \sim \mathfrak{S}$.

(b) $\mathfrak{S} \sim \mathfrak{T}$ *implies* $\mathcal{U}(\mathfrak{S}) \sim \mathcal{U}(\mathfrak{T})$.

As already mentioned modal logics cannot distinguish between bisimilar systems. They are *bisimulation-invariant* in the sense of the following definition.

▶ **Definition 2.3.** Let $\mathcal{C}$ be a class of transition systems.

(a) An MSO-formula φ is *bisimulation-invariant* over $\mathcal{C}$ if

$$\mathfrak{S} \sim \mathfrak{T} \quad \text{implies} \quad \mathfrak{S} \models \varphi \;\Leftrightarrow\; \mathfrak{T} \models \varphi, \quad \text{for all } \mathfrak{S},\mathfrak{T} \in \mathcal{C}.$$

(b) We say that, *over the class $\mathcal{C}$, bisimulation-invariant* MSO *coincides with* L_μ if, for every MSO-formula φ that is bisimulation-invariant over the class $\mathcal{C}$, there exists an L_μ-formula ψ such that

$$\mathfrak{S} \models \varphi \quad \text{iff} \quad \mathfrak{S} \models \psi, \quad \text{for all } \mathfrak{S} \in \mathcal{C}. \qquad ⌟$$

A straightforward induction over the structure of formulae shows that every L_μ-formula is bisimulation-invariant over all transition systems. Hence, bisimulation-invariance is a necessary condition for an MSO-formula to be equivalent to an L_μ-formula.

The following characterisations of bisimulation-invariant MSO have been obtained so far. We start with the result of Janin and Walukiewicz.

▶ **Theorem 2.4** (Janin, Walukiewicz [10])**.** *Over the class of all transition systems, bisimulation-invariant* MSO *coincides with* L_μ.

The main part of the proof consists in proving the following variant, which implies the case of all structures by a simple reduction.

▶ **Theorem 2.5** (Janin, Walukiewicz)**.** *Over the class of all trees, bisimulation-invariant* MSO *coincides with* L_μ.

There have already been two attempts at a finitary version. The first one is by Hirsch who considered the class of all regular trees, i.e., unravellings of finite transition systems. The proof is based on the fact that a formula is bisimulation-invariant over all trees if, and only if, it is bisimulation-invariant over regular trees.

▶ **Theorem 2.6** (Hirsch [9]). *Over the class of all regular trees, bisimulation-invariant* MSO *coincides with* L_μ.

The second result is by Dawar and Janin who considered the class of finite lassos, i.e., finite paths leading to a cycle. We will present a proof in Section 5 below.

▶ **Theorem 2.7** (Dawar, Janin [6]). *Over the class of all lassos, bisimulation-invariant* MSO *coincides with* L_μ.

In this paper, we will extend this last result to larger classes. We start with two easy observations. The first one is nearly trivial.

▶ **Theorem 2.8.** *Over every finite class $\mathcal{C}$ of finite transition systems, bisimulation-invariant* MSO *coincides with* L_μ.

The second observation is much deeper, but fortunately nearly all of the work has already been done by Janin and Walukiewicz.

▶ **Theorem 2.9.** *Over the class of all finite trees, bisimulation-invariant* MSO *coincides with* L_μ.

As a preparation for the more involved characterisation results to follow, we simplify our task by introducing the following property of a class $\mathcal{C}$ of transition systems, which will turn out to be equivalent to having a characterisation result for bisimulation-invariant MSO over $\mathcal{C}$.

▶ **Definition 2.10.** We say that a class $\mathcal{C}$ of transition systems has the *unravelling property* if, for every MSO-formula φ that is bisimulation-invariant over $\mathcal{C}$, there exists an MSO-formula $\hat{\varphi}$ that is bisimulation-invariant over trees such that

$$\mathfrak{S} \models \varphi \quad \text{iff} \quad \mathcal{U}(\mathfrak{S}) \models \hat{\varphi}\,, \quad \text{for all } \mathfrak{S} \in \mathcal{C}\,. \qquad \lrcorner$$

Using Theorem 2.5, we can reformulate this definition as follows. This version will be our main tool to prove characterisation results for bisimulation-invariant MSO: it is sufficient to prove that the given class has the unravelling property.

▶ **Theorem 2.11.** *A class $\mathcal{C}$ of transition systems has the unravelling property if, and only if, over $\mathcal{C}$ bisimulation-invariant* MSO *coincides with* L_μ.

Let us also note the following result, which allows us to extend the unravelling property from a given class to certain superclasses.

▶ **Lemma 2.12.** *Let $\mathcal{C}_0 \subseteq \mathcal{C}$ be classes such that every system in $\mathcal{C}$ is bisimilar to one in $\mathcal{C}_0$. If $\mathcal{C}_0$ has the unravelling property, then so does $\mathcal{C}$.*

3 Composition lemmas

We have mentioned above that automata-theoretic methods have so far been unsuccessful at attacking the finite version of the Janin–Walukiewicz result. Therefore, we rely on the composition method instead. Let us recall how this method works.

▶ **Definition 3.1.** Let $\mathfrak{S}$ and $\mathfrak{T}$ be transition systems (or general structures) and $m < \omega$ a number. The *m-theory* $\mathrm{Th}_m(\mathfrak{S})$ of $\mathfrak{S}$ is the set of all MSO-formulae of quantifier-rank m that are satisfied by $\mathfrak{S}$. (The quantifier-rank of a formula is its nesting depths of (first-order and second-order) quantifiers.) We write

$$\mathfrak{S} \equiv_m \mathfrak{T} \quad :\text{iff} \quad \mathrm{Th}_m(\mathfrak{S}) = \mathrm{Th}_m(\mathfrak{T}) .$$ ⌟

Roughly speaking the composition method provides some machinery that allows us to compute the m-theory of a given transition system by breaking it down into several components and looking at the m-theories of these components separately. This approach is based on the realisation that several operations on transition systems are compatible with m-theories in the sense that the m-theory of the result can be computed from the m-theories of the arguments. Statements to that effect are known as *composition theorems*. For an overview we refer the reader to [3] and [11]. The following basic operations and their composition theorems will be used below. We start with disjoint unions.

▶ **Definition 3.2.** The *disjoint union* of two structures $\mathfrak{A} = \langle A, R_0^{\mathfrak{A}}, \dots, R_m^{\mathfrak{A}} \rangle$ and $\mathfrak{B} = \langle B, R_0^{\mathfrak{B}}, \dots, R_m^{\mathfrak{B}} \rangle$ is the structure

$$\mathfrak{A} \oplus \mathfrak{B} := \big\langle A \cup B,\ R_0^{\mathfrak{A}} \cup R_0^{\mathfrak{B}}, \dots,\ R_m^{\mathfrak{A}} \cup R_m^{\mathfrak{B}},\ \mathrm{Left},\ \mathrm{Right} \big\rangle$$

obtained by forming the disjoint union of the universes and relations of $\mathfrak{A}$ and $\mathfrak{B}$ and adding two unary predicates $\mathrm{Left} := A$ and $\mathrm{Right} := B$ that mark whether an element belongs to $\mathfrak{A}$ or to $\mathfrak{B}$. If $\mathfrak{A}$ and $\mathfrak{B}$ are transition systems, the initial state of $\mathfrak{A} \oplus \mathfrak{B}$ is that of $\mathfrak{A}$. ⌟

The corresponding composition theorem looks as follows. It can be proved by a simple induction on m.

▶ **Lemma 3.3.** $\mathfrak{A} \equiv_m \mathfrak{A}'$ *and* $\mathfrak{B} \equiv_m \mathfrak{B}'$ *implies* $\mathfrak{A} \oplus \mathfrak{B} \equiv_m \mathfrak{A}' \oplus \mathfrak{B}'$.

Two other operations we need are interpretations and fusion operations.

▶ **Definition 3.4.** An *interpretation* is an operation τ on structures that is given by a list $\langle \delta(x), (\varphi_R(\bar{x}))_{R \in \Sigma} \rangle$ of MSO-formulae. Given a structure $\mathfrak{A}$, it produces the structure $\tau(\mathfrak{A})$ whose universe consists of all elements of $\mathfrak{A}$ satisfying the formula δ and whose relations are those defined by the formulae φ_R. The *quantifier-rank* of an interpretation is the maximal quantifier-rank of a formula in the list. An interpretation is *quantifier-free* if its quantifier-rank is 0. ⌟

▶ **Lemma 3.5.** *Let τ be an interpretation of quantifier-rank k. Then*

$$\mathfrak{A} \equiv_{m+k} \mathfrak{A}' \quad \textit{implies} \quad \tau(\mathfrak{A}) \equiv_m \tau(\mathfrak{A}') .$$

▶ **Definition 3.6.** Let P be a predicate symbol. The *fusion operation* fuse_P merges in a given structure all elements of the set P into a single element, i.e., all elements of P are replaced by a single new element and all edges incident with one of the old elements are attached to the new one instead. ⌟

▶ **Lemma 3.7.** $\mathfrak{A} \equiv_m \mathfrak{A}'$ *implies* $\mathrm{fuse}_P(\mathfrak{A}) = \mathrm{fuse}_P(\mathfrak{A}')$.

Using the composition theorems for these basic operations we can prove new theorems for derived operations. As an example let us consider *pointed paths,* i.e., paths where both end-points are marked by special colours.

▶ **Definition 3.8.** We denote the *concatenation* of two paths $\mathfrak{A}$ and $\mathfrak{B}$ by $\mathfrak{A} + \mathfrak{B}$. And we write $\mathfrak{A}^\bullet$ for the expansion of a path $\mathfrak{A}$ by two new constants for the end-points. ⌟

▶ **Corollary 3.9.** *Let* $\mathfrak{A}, \mathfrak{A}', \mathfrak{B}, \mathfrak{B}'$ *be paths. Then* $\mathfrak{A}^\bullet \equiv_m \mathfrak{A}'^\bullet$ *and* $\mathfrak{B}^\bullet \equiv_m \mathfrak{B}'^\bullet$ *implies* $(\mathfrak{A} + \mathfrak{B})^\bullet \equiv_m (\mathfrak{A}' + \mathfrak{B}')^\bullet$.

Proof. As the end-points are given by constants, we can construct a quantifier-free interpretation τ mapping $\mathfrak{A}^\bullet \oplus \mathfrak{B}^\bullet$ to $(\mathfrak{A} + \mathfrak{B})^\bullet$. ◀

Note that, since the concatenation operation is associative, it in particular follows that the set of m-theories of paths forms a semigroup.

Finally let us mention one more involved operation with a composition theorem. Let $\mathfrak{S}$ be a transition system and $\mathfrak{C} \subseteq \mathfrak{S}$ a subsystem. We say that $\mathfrak{C}$ is *attached* at the state $s \in S$ if there is a unique edge (in either direction) between a state in $S \setminus C$ and a state in C and this edge leads from s to the initial state of $\mathfrak{C}$.

▶ **Proposition 3.10.** *Let* $\mathfrak{S}$ *be a (possibly infinite) transition system and let* $\mathfrak{S}'$ *be the system obtained from* $\mathfrak{S}$ *by replacing an arbitrary number of attached subsystems by subsystems with the same* m*-theories (as the corresponding replaced ones). Then* $\mathfrak{S} \equiv_m \mathfrak{S}'$.

For a finite system $\mathfrak{S}$ this statement can be proved in the same way as Corollary 3.9 by expressing $\mathfrak{S}$ as a disjoint union followed by a quantifier-free interpretation. For infinite systems, we need a more powerful version of the disjoint union operation called a *generalised sum* (see [16]).

As presented above these tools work with m-theories, which is not quite what we need since we have to also account for bisimulation-invariance. To do so we modify the definitions as follows.

▶ **Definition 3.11.** Let $\mathcal{C}$ be a class of transition systems and $m < \omega$ a number.

(a) We denote by $\simeq^m_{\mathcal{C}}$ the transitive closure of the union $\equiv_m \cup \sim$ restricted to the class $\mathcal{C}$. Formally, we define $\mathfrak{S} \simeq^m_{\mathcal{C}} \mathfrak{T}$ if there exist systems $\mathfrak{C}_0, \dots, \mathfrak{C}_n \in \mathcal{C}$ such that

$$\mathfrak{C}_0 = \mathfrak{S}\,, \quad \mathfrak{C}_n = \mathfrak{T}\,, \quad \text{and} \quad \mathfrak{C}_i \equiv_m \mathfrak{C}_{i+1} \quad \text{or} \quad \mathfrak{C}_i \sim \mathfrak{C}_{i+1}\,, \qquad \text{for all } i < n\,.$$

(b) We denote by $\mathrm{Th}^m_{\mathcal{C}}(\mathfrak{S})$ the set of all MSO-formulae of quantifier-rank m that are bisimulation-invariant over $\mathcal{C}$ and that are satisfied by $\mathfrak{S}$, and we define

$$\mathfrak{S} \equiv^m_{\mathcal{C}} \mathfrak{S}' \quad :\text{iff} \quad \mathrm{Th}^m_{\mathcal{C}}(\mathfrak{S}) = \mathrm{Th}^m_{\mathcal{C}}(\mathfrak{S}')\,.$$

We also set $\mathrm{TH}^m_{\mathcal{C}} := \{\, \mathrm{Th}^m_{\mathcal{C}}(\mathfrak{S}) \mid \mathfrak{S} \in \mathcal{C} \,\}$. ⌟

Note that, up to logical equivalence, there are only finitely many formulae of a given quantifier-rank. Hence, each set $\mathrm{TH}^m_{\mathcal{C}}$ is finite and the relations $\equiv_m$, $\equiv^m_{\mathcal{C}}$ and $\simeq^m_{\mathcal{C}}$ have finite index.

▶ **Lemma 3.12.** *If* φ *is a* MSO*-formula of quantifier-rank* m *that is bisimulation-invariant over* $\mathcal{C}$*, then* $\mathfrak{S} \simeq^m_{\mathcal{C}} \mathfrak{T}$ *implies* $\mathfrak{S} \models \varphi \Leftrightarrow \mathfrak{T} \models \varphi$.

Some of the above composition theorems also hold for the relation $\simeq^m_{\mathcal{C}}$. This is immediate if the operation in question also preserves bisimilarity. We mention only two such results. The second one will be needed below.

▶ **Lemma 3.13.** *Let* $\mathcal{C}$ *be a class that is closed under disjoint unions.*

$$\mathfrak{A} \simeq^m_{\mathcal{C}} \mathfrak{A}' \quad \textit{and} \quad \mathfrak{B} \simeq^m_{\mathcal{C}} \mathfrak{B}' \quad \textit{implies} \quad \mathfrak{A} \oplus \mathfrak{B} \simeq^m_{\mathcal{C}} \mathfrak{A}' \oplus \mathfrak{B}'\,.$$

▶ **Proposition 3.14.** *Let $\mathcal{C}$ and $\mathcal{D}$ be two classes, $\mathfrak{S} \in \mathcal{C}$ a (possibly infinite) transition system and let $\mathfrak{S}'$ be the system obtained from $\mathfrak{S}$ by replacing an arbitrary number of attached subsystems by subsystems which are $\simeq^m_{\mathcal{D}}$-equivalent. Then $\mathfrak{S} \simeq^m_{\mathcal{C}} \mathfrak{S}'$ provided that the class $\mathcal{C}$ is closed under the operation of replacing attached subsystems in $\mathcal{D}$.*

4 Types

Our strategy to prove the unravelling property for a class $\mathcal{C}$ is as follows. For every quantifier-rank m, we assign to each tree $\mathfrak{T}$ a so-called *m-type* $\tau_m(\mathfrak{T})$. We choose the functions τ_m such that we can compute the theory $\mathrm{Th}^m_{\mathcal{C}}(\mathfrak{C})$ of a system $\mathfrak{C} \in \mathcal{C}$ from the m-type $\tau_m(\mathcal{U}(\mathfrak{C}))$ of its unravelling. Furthermore, we need to find MSO-formulae checking whether a tree has a given m-type. The formal definition is as follows.

▶ **Definition 4.1.** Let $\mathcal{C}$ be a class of transition systems and $\mathcal{T}$ the class of all trees.

(a) A *family of type functions* for $\mathcal{C}$ is a family of functions $\tau_m : \mathcal{T} \to \Theta_m$, for $m < \omega$, where the co-domains Θ_m are finite sets and each τ_m satisfies the following two axioms.

(S1) $\tau_m(\mathcal{U}(\mathfrak{C})) = \tau_m(\mathcal{U}(\mathfrak{C}'))$ implies $\mathrm{Th}^m_{\mathcal{C}}(\mathfrak{C}) = \mathrm{Th}^m_{\mathcal{C}}(\mathfrak{C}')$, for $\mathfrak{C}, \mathfrak{C}' \in \mathcal{C}$.

(S2) $\mathfrak{T} \sim \mathfrak{T}'$ implies $\tau_m(\mathfrak{T}) = \tau_m(\mathfrak{T}')$, for all $\mathfrak{T}, \mathfrak{T}' \in \mathcal{T}$.

(b) A family $(\tau_m)_m$ of type functions is *definable* if, for every $\theta \in \Theta_m$, there exists an MSO-formula ψ_θ such that

(S3) $\mathfrak{T} \models \psi_\theta$ iff $\tau_m(\mathfrak{T}) = \theta$, for all trees $\mathfrak{T}$. ⌟

Let us start by showing how to prove the unravelling property using type functions. The following characterisation theorem can be considered to be the main theoretical result of this article.

▶ **Theorem 4.2.** *Let $\mathcal{C}$ be a class of transition systems and $\mathcal{T}$ the class of all trees. The following statements are equivalent.*

(1) *Over $\mathcal{C}$, bisimulation-invariant* MSO *coincides with* L_μ.

(2) *$\mathcal{C}$ has the unravelling property.*

(3) *There exists a definable family $(\tau_m)_m$ of type functions for $\mathcal{C}$.*

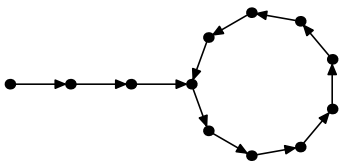
(4) *There exist functions $g : \omega \to \omega$ and $h_m : \mathrm{TH}^{g(m)}_{\mathcal{T}} \to \mathrm{TH}^m_{\mathcal{C}}$, for $m < \omega$, such that*

$$h_m\big(\mathrm{Th}^{g(m)}_{\mathcal{T}}(\mathcal{U}(\mathfrak{C}))\big) = \mathrm{Th}^m_{\mathcal{C}}(\mathfrak{C}), \quad \text{for all } \mathfrak{C} \in \mathcal{C}$$

(in other words, the $g(m)$-theory of $\mathcal{U}(\mathfrak{C})$ determines the m-theory of $\mathfrak{C}$).

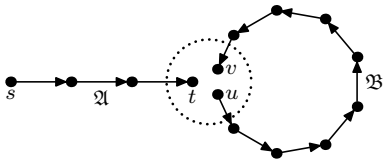
5 Lassos

As an application of type functions, we consider a very simple example, the class of *lassos*. Our proof is based on more or less the same arguments as that by Dawar and Janin [6], just the presentation differs. A lasso is a transition system consisting of a directed path ending in a cycle.



We allow the borderline cases where the initial path has length 0 or the cycle consists of only a single edge.

To define the type of a lasso, note that we can construct every lasso $\mathfrak{L}$ from two finite paths $\mathfrak{A}$ and $\mathfrak{B}$ by identifying three of their end-points.



The paths $\mathfrak{A}$ and $\mathfrak{B}$ are uniquely determined by $\mathfrak{L}$. We will refer to $\mathfrak{A}$ as the *tail* of the lasso and to $\mathfrak{B}$ as the *loop.* We introduce two kinds of types for lassos, a strong one and a weak one.

▶ **Definition 5.1.** The *strong m-type* of a lasso $\mathfrak{L}$ with tail $\mathfrak{A}$ and loop $\mathfrak{B}$ is the pair

$$\mathrm{stp}_m(\mathfrak{L}) := \langle \alpha, \beta \rangle\,, \quad \text{where} \quad \alpha := \mathrm{Th}_m(\mathfrak{A}^\bullet) \quad \text{and} \quad \beta := \mathrm{Th}_m(\mathfrak{B}^\bullet)\,. \qquad \lrcorner$$

The strong m-type of a lasso uniquely determines its m-theory.

▶ **Lemma 5.2.** *Let* $\mathfrak{L}_0$ *and* $\mathfrak{L}_1$ *be lassos.*

$$\mathrm{stp}_m(\mathfrak{L}_0) = \mathrm{stp}_m(\mathfrak{L}_1) \quad \textit{implies} \quad \mathfrak{L}_0 \equiv_m \mathfrak{L}_1\,.$$

The problem with the strong type of a lasso $\mathfrak{L}$ is that we cannot recover it from the unravelling of $\mathfrak{L}$ as the decomposition of $\mathcal{U}(\mathfrak{L})$ into the parts of $\mathfrak{L}$ is uncertain. Therefore we introduce another notion of a type where this recovery is possible. For this we recall some facts from the theory of ω-semigroups.

Recall that we have noted in Corollary 3.9 that the m-theories of pointed paths form a finite semigroup with respect to concatenation. Furthermore, every element a of a finite semigroup has an *idempotent power* a^π, which is defined as the value a^n where n is the least natural number such that $a^n \cdot a^n = a^n$.

▶ **Definition 5.3.** (a) A *factorisation* of an infinite path $\mathfrak{A}$ is a sequence $(\mathfrak{A}_i)_{i<\omega}$ of finite paths whose concatenation is $\mathfrak{A}$. Such a factorisation has *m-type* $\langle \alpha, \beta \rangle$ if

$$\alpha := \mathrm{Th}_m(\mathfrak{A}_0^\bullet) \quad \text{and} \quad \beta := \mathrm{Th}_m(\mathfrak{A}_i^\bullet)\,, \quad \text{for } i > 0\,.$$

(b) Two pairs $\langle \alpha, \beta \rangle$ and $\langle \gamma, \delta \rangle$ of m-theories are *conjugate* if there are m-theories ξ and η such that

$$\gamma\delta^\pi = \alpha\beta^\pi\xi\,, \quad \beta^\pi = \xi\eta\,, \quad \text{and} \quad \delta^\pi = \eta\xi\,.$$

Being conjugate is an equivalence relation. We denote the equivalence class of a pair $\langle \alpha, \beta \rangle$ by $[\alpha, \beta]$.

(c) The *weak m-type* of a lasso $\mathfrak{L}$ with parts $\mathfrak{A}$ and $\mathfrak{B}$ is

$$\mathrm{wtp}_m(\mathfrak{L}) := [\alpha, \beta]\,, \quad \text{where} \quad \alpha := \mathrm{Th}_m(\mathfrak{A}^\bullet) \quad \text{and} \quad \beta := \mathrm{Th}_m(\mathfrak{B}^\bullet)\,.$$

(d) The *m-type* of an infinite tree $\mathfrak{T}$ is

$$\tau_m(\mathfrak{T}) := [\alpha, \beta]\,,$$

where α and β is an arbitrary pair of m-theories such that every branch of $\mathfrak{T}$ has a factorisation of m-type $\langle \alpha, \beta \rangle$. If there is no such pair, we set $\tau_m(\mathfrak{T}) := \bot$. $\lrcorner$

▶ **Lemma 5.4.** *Let $\mathcal{L}$ be the class of all lassos and let $\mathfrak{L}_0, \mathfrak{L}_1 \in \mathcal{L}$.*

$$\mathrm{wtp}_m(\mathfrak{L}_0) = \mathrm{wtp}_m(\mathfrak{L}_1) \quad \textit{implies} \quad \mathfrak{L}_0 \simeq^m_{\mathcal{L}} \mathfrak{L}_1 \,.$$

To show that the functions $(\tau_m)_m$ form a family of type functions, we need the following standard facts about factorisations and their types (see, e.g., Section II.2 of [14]).

▶ **Proposition 5.5.** *Let $\mathfrak{A}$ be an infinite path.*
(a) *$\mathfrak{A}$ has a factorisation of type $\langle \alpha, \beta \rangle$, for some α and β.*
(b) *If $\mathfrak{A}$ has factorisations of type $\langle \alpha, \beta \rangle$ and $\langle \gamma, \delta \rangle$, then $\langle \alpha, \beta \rangle$ and $\langle \gamma, \delta \rangle$ are conjugate.*
Note that these two statements imply in particular that the type $\tau_m(\mathfrak{T})$ of a tree $\mathfrak{T}$ is well-defined.

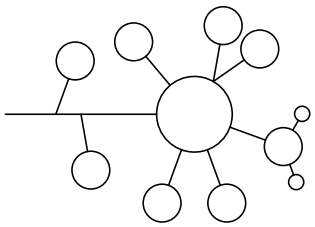
▶ **Lemma 5.6.** *The functions $(\tau_m)_m$ defined above form a definable family of type functions for the class of all lassos.*

By Theorem 4.2, it therefore follows that the class of lassos has the unravelling property.

▶ **Theorem 5.7.** *The class of all lassos has the unravelling property.*

6 Hierarchical Lassos

After the simple example in the previous section, let us give a more substantial application of the type machinery. We consider *hierarchical* (or *nested*) lassos. These are obtained from a lasso by repeatedly attaching sublassos to some states. More precisely, a 1-lasso is just an ordinary lasso, while inductively a $(k+1)$-lasso is obtained from a k-lasso by attaching one or more lassos to some of the states. (Each state may have several sublassos attached.)



Alternatively, we can obtain a $(k+1)$-lasso $\mathfrak{M}$ from a 1-lasso $\mathfrak{L}$ by attaching k-lassos. We will call this lasso $\mathfrak{L}$ the *main lasso* of $\mathfrak{M}$.

The types we use for k-lassos are based on the same principles as those for simple lassos, but we have to nest them in order to take the branching of a hierarchical lasso into account.

▶ **Definition 6.1.** Let $t : \mathrm{dom}(t) \to C$ be a labelled tree and $m < \omega$.
(a) For a branch β of t, we set

$$\mathrm{wtp}_m(\beta) := [\sigma, \tau] \,,$$

if β has a factorisation of m-type $\langle \sigma, \tau \rangle$. (By Proposition 5.5, this is well-defined.)
(b) For $k < \omega$, we define

$$\mathrm{tp}^0_m(t) := \big\{\, \mathrm{wtp}_m(\beta) \;\big|\; \beta \text{ a branch of } t \,\big\} \,,$$
$$\mathrm{tp}^{k+1}_m(t) := \mathrm{tp}^0_m(\mathrm{TP}^k_m(t)) \,,$$

where $\mathrm{TP}^k_m(t) : T \to C \times \mathcal{P}(\Theta^k_m)$ is the tree with labelling

$$\mathrm{TP}^k_m(t)(v) := \big\langle t(v),\ \{\, \mathrm{tp}^k_m(t|_u) \mid u \text{ a successor of } v \,\} \big\rangle \,. \qquad \lrcorner$$

We will prove that the functions tp_m^k form a family of type functions. Note that it follows immediately from the definition that they satisfy Properties (S2) and (S3). Hence, it only remains to check (S1).

▶ **Lemma 6.2.** (a) *Let $\mathfrak{M}$ be a k-lasso and $\mathfrak{N}$ a k'-lasso. Then*

$$\mathcal{U}(\mathfrak{M}) \sim \mathcal{U}(\mathfrak{N}) \quad \textit{implies} \quad \mathrm{tp}_m^k(\mathfrak{M}) = \mathrm{tp}_m^k(\mathfrak{N})\,.$$

(b) *For every type τ, there exists an* MSO*-formula φ such that*

$$\mathcal{U}(\mathfrak{M}) \models \varphi \quad \text{iff} \quad \mathrm{tp}_m^k(\mathfrak{M}) = \tau\,.$$

Thus, to prove that the class of k-lassos has the unravelling property it is sufficient to show that tp_m^k also satisfies Property (S1). We will do so by induction on k. The base case of this induction rests on the following lemma.

▶ **Lemma 6.3.** *Let $\mathcal{L}_k$ be the class of all k-lassos and let $\mathfrak{M}$ be a k-lasso such that, for every vertex v and all branches β and γ starting at a successor of v, we have $\mathrm{wtp}_m(\beta) = \mathrm{wtp}_m(\gamma)$. Then $\mathfrak{M} \simeq^m_{\mathcal{L}_k} \mathfrak{N}$, for some 1-lasso $\mathfrak{N}$.*

▶ **Proposition 6.4.** *Let $\mathfrak{M}$ be a k-lasso and $\mathfrak{N}$ a k'-lasso. For $m \geq 1$,*

$$\mathrm{tp}_m^k(\mathfrak{M}) = \mathrm{tp}_m^k(\mathfrak{N}) \quad \textit{implies} \quad \mathfrak{M} \simeq^m_{\mathcal{L}_K} \mathfrak{N}\,,$$

where $\mathcal{L}_K$ is the class of all K-lassos with $K := \max(k, k')$.

Using Theorem 4.2 we now immediately obtain the following statement.

▶ **Theorem 6.5.** *For every k, the class of all k-lassos has the unravelling property.*

7 Reductions

We would like to define reductions that allow us to prove that a certain class has the unravelling property when we already know that some other class has this property. To do so, we encode every transition system of the first class by some system in the second one. The main example we will be working with is a function ϱ that removes certain attached subsystems and uses additional vertex labels to remember the m-theories of all deleted system. Up to equivalence of m-theories, we can undo this operation by a function η that attaches to each vertex labelled by some m-theory θ some fixed system with theory θ. Let us give a general definition of such pairs of maps.

▶ **Definition 7.1.** Let $\mathcal{C}$ and $\mathcal{D}$ be classes of transition systems and $k, m < \omega$. A function $\varrho : \mathcal{C} \to \mathcal{D}$ is a (k, m)-*encoding map* if there exists a function $\eta : \mathcal{D} \to \mathcal{C}$ such that

(E1) $\varrho(\eta(\mathfrak{D})) \simeq^k_{\mathcal{D}} \mathfrak{D}\,, \quad$ for all $\mathfrak{D} \in \mathcal{D}\,.$

(E2) $\varrho(\mathfrak{C}) \simeq^k_{\mathcal{D}} \varrho(\mathfrak{C}') \quad$ implies $\quad \mathfrak{C} \simeq^m_{\mathcal{C}} \mathfrak{C}'\,, \quad$ for all $\mathfrak{C}, \mathfrak{C}' \in \mathcal{C}\,.$

In this case, we call the function η a (k, m)-*decoding map* for ϱ. ⌟

These two axioms imply dual axioms with the functions ϱ and η exchanged.

▶ **Lemma 7.2.** *Let $\eta : \mathcal{D} \to \mathcal{C}$ be a (k, m)-decoding map for $\varrho : \mathcal{C} \to \mathcal{D}$.*

(E3) $\eta(\varrho(\mathfrak{C})) \simeq^m_{\mathcal{C}} \mathfrak{C}\,, \quad$ *for all* $\mathfrak{C} \in \mathcal{C}\,.$

(E4) $\mathfrak{D} \simeq^k_{\mathcal{D}} \mathfrak{D}' \quad$ *implies* $\quad \eta(\mathfrak{D}) \simeq^m_{\mathcal{C}} \eta(\mathfrak{D}')\,, \quad$ *for all* $\mathfrak{D}, \mathfrak{D}' \in \mathcal{D}\,.$

The axioms of an encoding map were chosen to guarantee the property stated in the following lemma. It will be used below to prove that encoding maps can be used to transfer the unravelling property from one class to another.

▶ **Lemma 7.3.** *Let $\varrho : \mathcal{C} \to \mathcal{D}$ a (k, m)-encoding map and $\eta : \mathcal{D} \to \mathcal{C}$ a (k, m)-decoding map for ϱ. For every* MSO*-formula φ of quantifier-rank m that is bisimulation-invariant over $\mathcal{C}$, there exists an* MSO*-formula $\hat{\varphi}$ of quantifier-rank k that is bisimulation-invariant over $\mathcal{D}$ such that*

$$\mathfrak{C} \models \varphi \quad \text{iff} \quad \varrho(\mathfrak{C}) \models \hat{\varphi}\,, \quad \textit{for all } \mathfrak{C} \in \mathcal{C}\,.$$

It remains to show how to use encoding maps to transfer the unravelling property. Just the existence of such a map is not sufficient. It also has to be what we call definable.

▶ **Definition 7.4.** Let $\mathcal{C}$ be a class of transition systems.

(a) A (k, m)-encoding map $\varrho : \mathcal{C} \to \mathcal{D}$ is *definable* if, for every MSO-formula φ that is bisimulation-invariant over trees, there exists an MSO-formula $\hat{\varphi}$ that is bisimulation-invariant over trees such that

$$\mathcal{U}(\varrho(\mathfrak{C})) \models \varphi \quad \text{iff} \quad \mathcal{U}(\mathfrak{C}) \models \hat{\varphi}\,, \quad \text{for all } \mathfrak{C} \in \mathcal{C}\,.$$

(b) We say that $\mathcal{C}$ is *reducible* to a family $(\mathcal{D}_m)_{m<\omega}$ of classes if there exist a map $g : \omega \to \omega$ and, for each $m < \omega$, functions $\varrho_m : \mathcal{C} \to \mathcal{D}_m$ and $\eta_m : \mathcal{D}_m \to \mathcal{C}$ such that ϱ_m is a definable $(g(m), m)$-encoding map and η_m a corresponding $(g(m), m)$-decoding map. ⌟

(The only reason why we use a family of classes to reduce to, instead of a single one is so that we can have the labellings of systems in $\mathcal{D}_m$ depend on the quantifier-rank m.)

▶ **Theorem 7.5.** *Suppose that $\mathcal{C}$ is reducible to $(\mathcal{D}_m)_{m<\omega}$. If every class $\mathcal{D}_m$ has the unravelling property, so does $\mathcal{C}$.*

8 Finite Cantor–Bendixson rank

One common property of k-lassos is that the trees we obtain by unravelling them all have finite Cantor–Bendixson rank. In this section we will generalise our results to cover transition systems with this more general property. The proof below consists in a two-step reduction to the class of k-lassos.

▶ **Definition 8.1.** Let $\mathfrak{T}$ be a finitely branching tree. The *Cantor–Bendixson derivative* of $\mathfrak{T}$ is the tree $\mathfrak{T}'$ obtained from $\mathfrak{T}$ by removing all subtrees that have only finitely many infinite branches. The *Cantor-Bendixson rank* of a tree $\mathfrak{T}$ is the least ordinal α such that applying $\alpha + 1$ Cantor–Bendixson derivatives to $\mathfrak{T}$ results in an empty tree. The *Cantor–Bendixson rank* of a transition system $\mathfrak{S}$ is equal to the Cantor–Bendixson rank of its unravelling. ⌟

We can go from the class of k-lassos to that of systems with bounded Cantor–Bendixson rank in two steps.

▶ **Definition 8.2.** (a) A transition system is a *generalised k-lasso* if it is obtained from a finite tree by attaching (one or several) k-lassos to every leaf.

(b) A transition system $\mathfrak{T}$ is a *tree extension* of $\mathfrak{S}$ if $\mathfrak{T}$ is obtained from $\mathfrak{S}$ by attaching an arbitrary number of finite trees to some of the vertices. ⌟

With these two notions we can characterise the property of having bounded Cantor–Bendixson rank as follows.

▶ **Proposition 8.3.** *Let $\mathfrak{S}$ be a finite transition system.*

(a) *For every $k < \omega$, the following statements are equivalent.*

(1) *$\mathfrak{S}$ has Cantor–Bendixson rank at most k.*

(2) *$\mathfrak{S}$ is bisimilar to a tree extension of a generalised $(k+1)$-lasso.*

(b) *The following statements are equivalent.*

(1) *$\mathfrak{S}$ has finite Cantor–Bendixson rank.*

(2) *$\mathfrak{S}$ is bisimilar to a tree extension of a generalised k-lasso, for some $k < \omega$.*

(3) *Every strongly connected component of $\mathfrak{S}$ is either a singleton or a cycle.*

To prove the unravelling property for the transition systems of bounded Cantor–Bendixson rank, we proceed in two steps. First we consider generalised k-lassos and then their tree extensions.

▶ **Theorem 8.4.** *For fixed k, the class of all generalised k-lassos has the unravelling property.*

Using this intermediate step, we obtain the following proof for transition systems with bounded Cantor–Bendixson rank.

▶ **Theorem 8.5.** *The class of all finite transition systems of Cantor–Bendixson rank at most k has the unravelling property.*

▶ **Corollary 8.6.** *Over the class of all finite transition systems with Cantor–Bendixson rank at most k, bisimulation-invariant* MSO *coincides with* L_μ.

9 Conclusion

We have shown in several simple examples how to characterise bisimulation-invariant MSO in the finite. In particular, we have proved that it coincides with L_μ over

- every finite class (Theorem 2.8),
- the class of all finite trees (Theorem 2.9),
- the classes of all lassos, k-lassos, and generalised k-lassos (Theorems 5.7, 6.5, and 8.4),
- the class of all systems of Cantor–Bendixson rank at most k (Theorem 8.5).

Our main tool in these proofs was the unravelling property (Theorem 2.11). It will be interesting to see how far our methods can be extended to more complicated classes. For instance, can they be used to prove the following conjecture?

Conjecture. *If a class $\mathcal{C}$ of transition systems has the unravelling property, then so does the class of all subdivisions of systems in $\mathcal{C}$.*

A good first step seems to be the class of all finite transition systems that have Cantor–Bendixson rank k, for some $k < \omega$ that is not fixed.

In this paper we have considered only transition systems made out of paths with very limited branching. To extend our techniques to classes allowing for more branching seems to require new ideas. A simple test case that looks promising is the class of systems with a 'lasso-decomposition' of width k, i.e., something like a tree decomposition but where the pieces are indexed by a lasso instead of a tree.

References

1. H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.
2. J. van Benthem. *Modal Correspondence Theory.* Ph. D. Thesis, University of Amsterdam, Amsterdam, 1976.
3. A. Blumensath, T. Colcombet, and C. Löding. Logical Theories and Compatible Operations. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives*, pages 73–106. Amsterdam University Press, 2007.
4. F. Carreiro. *Fragments of Fixpoint Logics.* PhD Thesis, Institute for Logic, Language and Computation, Amsterdam, 2015.
5. I. Ciardelli and M. Otto. Bisimulation in Inquisitive Modal Logic. In *Proc. 16th Conference on Theoretical Aspects of Rationality and Knowledge, TARK 2017*, pages 151–166, 2017.
6. A. Dawar and D. Janin. On the bisimulation invariant fragment of monadic Σ_1 in the finite. In *Proc. of the 24th Int. Conf. on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2004*, pages 224–236, 2004.
7. E. Grädel, C. Hirsch, and M. Otto. Back and Forth Between Guarded and Modal Logics. *ACM Transactions on Computational Logics*, pages 418–463, 2002.
8. E. Grädel, W. Thomas, and T. Wilke. *Automata, Logic, and Infinite Games.* LNCS 2500. Springer-Verlag, 2002.
9. C. Hirsch. *Guarded Logics: Algorithms and Bisimulation.* Ph. D. Thesis, RWTH Aachen, Aachen, 2002.
10. D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *Proc. of the 7th International Conference on Concurrency Theory, CONCUR 1996*, pages 263–277, 1996.
11. J. A. Makowsky. Algorithmic aspects of the Feferman-Vaught Theorem. *Annals of Pure and Applied Logic*, 126:159–213, 2004.
12. F. Moller and A. Rabinovitch. On the expressive power of CTL*. In *Proc. 14th IEEE Symp. on Logic in Computer Science, LICS*, pages 360–369, 1999.
13. F. Moller and A. Rabinovitch. Counting on CTL*: on the expressive power of monadic path logic. *Information and Computation*, 184:147–159, 2003.
14. D. Perrin and J.-É. Pin. *Infinite Words – Automata, Semigroups, Logic and Games.* Elsevier, 2004.
15. E. Rosen. Modal logic over finite structures. *Journal of Logic, Language and Information*, 6:427–439, 1997.
16. S. Shelah. The Monadic Second Order Theory of Order. *Annals of Mathematics*, 102:379–419, 1975.
17. C. Stirling. Bisimulation and logic. In D. Sangiorgi and J. Rutten, editors, *Advanced topics in Bisimulation and Coinduction*, pages 172–196. Cambridge University Press, 2011.

Binary Reachability of Timed Pushdown Automata via Quantifier Elimination and Cyclic Order Atoms

Lorenzo Clemente[1]
University of Warsaw
clementelorenzo@gmail.com
https://orcid.org/0000-0003-0578-9103

Sławomir Lasota[2]
University of Warsaw
sl@mimuw.edu.pl
https://orcid.org/0000-0001-8674-4470

Abstract

We study an expressive model of timed pushdown automata extended with modular and fractional clock constraints. We show that the binary reachability relation is effectively expressible in hybrid linear arithmetic with a rational and an integer sort. This subsumes analogous expressibility results previously known for finite and pushdown timed automata with untimed stack. As key technical tools, we use quantifier elimination for a fragment of hybrid linear arithmetic and for cyclic order atoms, and a reduction to register pushdown automata over cyclic order atoms.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models, Theory of computation → Logic and verification, Theory of computation → Formal languages and automata theory

Keywords and phrases timed automata, reachability relation, timed pushdown automata, linear arithmetic

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.118

1 Introduction

Timed automata (TA) are one of the most studied models of reactive timed systems. The fundamental result that paved the way to automatic verification of timed systems is decidability (and PSPACE-completeness) of the reachability problem for TA [2]. However, in certain applications, such as in parametric verification, deciding reachability is insufficient, and one needs to construct the more general *binary reachability relation*, i.e., the entire (possibly infinite) set of of pairs of configurations (c_i, c_f) s.t. there is an execution from c_i to c_f. The reachability relation for TA has been shown to be effectively expressible in hybrid linear arithmetic with rational and integer sorts [11, 13, 15, 18]. Since hybrid logic is decidable, this yields an alternative proof of decidability of the reachability problem.

In this paper, we compute the reachability relation for timed automata extended with a stack. An early model of *pushdown timed automata* (PTDA) extending TA with a (classical, untimed) stack has been considered by Bouajjani *et al.* [5]. More recently, *dense-timed pushdown automata* (dtPDA) have been proposed by Abdulla *et al.* [1] as an extension of PTDA. In dtPDA, stack symbols are equipped with rational *ages*, which initially are 0 and increase with the elapse of time at the same rate as global clocks; when a symbol is popped,

[1] Partially supported by Polish NCN grant 2017/26/D/ST6/00201.
[2] Partially supported by Polish NCN grant 2016/21/B/ST6/01505.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 118; pp. 118:1–118:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

its age is tested for membership in an interval. While dtPDA syntactically extend PTDA by considering a timed stack, timed constraints can in fact be removed while preserving the timed language recognised by the dtPDA, and thus they semantically collapse to PTDA [8]. This motivates the quest for a strictly more expressive generalisation of PTDA and dtPDA with a truly timed stack. It has been observed in [21] that adding fractional stack constraints prevents the stack from being untimed, and thus strictly enriches the expressive power[3].

We embrace this observation and propose the model of *timed pushdown automata* (TPDA), which extends timed automata with a timed stack and integer, fractional, and modulo diagonal/non-diagonal constraints. The model features local clocks and stack clocks. As time elapses, all clocks increase their values, and they do so at the same rate. Local clocks can be reset and compared according to the generalised constraints above. At the time of a push operation, new stack clocks are created whose values are initialised, possibly non-deterministically, as to satisfy a given push constraint between stack clocks and local clocks; similarly, a pop operation requires that stack clocks to be popped satisfy a given pop constraint of analogous form. Stack push/pop constraints are also of the form of diagonal/non-diagonal integer, modulo, and fractional constraints.

Contributions. We compute the *binary reachability relation* of TPDA, i.e., the family of binary relations $\{\rightsquigarrow_{\ell r}\} \subseteq \mathbb{Q}^X_{\geq 0} \times \mathbb{Q}^X_{\geq 0}$ for control locations ℓ, r s.t. from the initial clock valuation $\mu \in \mathbb{Q}^X_{\geq 0}$ and control location ℓ we can reach the final clock valuation $\nu \in \mathbb{Q}^X_{\geq 0}$ and control location r, written $\mu \rightsquigarrow_{\ell r} \nu$. The stack is empty at the beginning and at the end of the computation. The main contribution of the paper is the effective computation of the TPDA reachability relation in the existential fragment of *linear arithmetic* $\mathcal{L}_{\mathbb{Z},\mathbb{Q}}$, a two-sorted logic combining Presburger arithmetic $(\mathbb{Z}, \leq, (\equiv_m)_{m\in\mathbb{N}}, +, 0)$ and linear rational arithmetic $(\mathbb{Q}, \leq, +, 0)$. As a byproduct of our constructions, we actually characterise the more general *ternary reachability relation* $\mu \overset{\pi}{\rightsquigarrow}_{\ell r} \nu$, where μ, ν are as above and $\pi : \mathbb{N}^\Sigma$ additionally counts the number of occurrences of input letters over a finite alphabet Σ, i.e., the Parikh image of the run. To our knowledge, the ternary reachability relation was not previously considered. As an application of ternary reachability, we can model, for instance, letter counts of initial and final, possibly non-empty, stack contents. Thus, ternary reachability is an expressive extension of binary reachability.

The computation of the ternary reachability relation is achieved by two consecutive translations. First, we transform a TPDA into a *fractional* TPDA, which uses only fractional constraints. In this step we exploit *quantifier elimination* for a fragment of linear arithmetic corresponding to clock constraints. Quantifier elimination is a pivotal tool in this work, and to our knowledge its use in the study of timed models is novel. The final integer value of clocks is reconstructed by letting the automaton input special tick symbol $\checkmark_x$ every time clock x reaches an integer value (provided it is not reset anymore later); it is here that ternary reachability is more suitable than binary reachability.

Secondly, a fractional TPDA is transformed into a PDA with registers (RPDA) over the so called *cyclic order atoms* $(\mathbb{Q} \cap [0,1), K)$ [7], where K is the ternary cyclic order relation

$$K(a,b,c) \equiv a < b < c \vee b < c < a \vee c < a < b, \qquad \text{for } a,b,c \in \mathbb{Q} \cap [0,1). \tag{1}$$

In other words, $K(a,b,c)$ holds if, distributing a, b, c on the unit circle and going clockwise from a, then we fist visit b and afterwards c. Since fractional values are wrapped around 0 when time increases, K is invariant under time elapse. We use registers to store the fractional

[3] For TA, fractional constraints can be handled by the original region construction and do not make the model harder to analyse [2].

parts of absolute times of last clock resets; fractional constraints on clocks are simulated by constraints on registers using K. In order to compute the reachability relation for RPDA we use again quantifier elimination, this time over cyclic order atoms. The latter property holds since cyclic order atoms constitute a homogeneous structure [16]. Therefore, another contribution of this work is the solution of a nontrivial problem such as computing the reachability relation for TPDA, which is a clock model, as an application of RPDA, which is a register model. The analysis of RPDA is substantially easier than a direct analysis of (fractional) TPDA.

From the complexity standpoint, the formula characterising the reachability relation of a TPDA is computable in double exponential time. However, when cast down to TA or TPDA with timeless stack (which subsume PTDA and, a posteriori, dtPDA), the complexity drops to singly exponential, matching the previously known complexity for TA [18]. For PTDA, no complexity was previously given in [12], and thus the result is new. For TPDA, the binary reachability problem has not been studied before. Since the existential fragment of $\mathcal{L}_{\mathbb{Z},\mathbb{Q}}$ is decidable in NP (because so is existential linear rational arithmetic [19] and existential Presburger arithmetic [23]), we can solve the reachability problem of TPDA in 2NEXP by reduction to satisfiability for $\mathcal{L}_{\mathbb{Z},\mathbb{Q}}$. Since our constructions preserve the languages of all the models involved, untimed TPDA languages are context-free.

Discussion. From a syntactic point of view, TPDA significantly lifts the restrictions of dtPDA– which allow only classical non-diagonal constraints, i.e., interval tests, and thus has neither diagonal, nor modulo, nor fractional constraints – and of the model of [21] – which additionally allows diagonal/non-diagonal fractional tests, and thus does not have modulo constraints. Since classical diagonal constraints reduce to classical non-diagonal constraints, and, *in the presence of fractional constraints*, integer and modulo constraints can be removed altogether (cf. Sec. 4), TPDA are expressively equivalent to [21]. However, while [21] solves the control state reachability problem, we solve the more general problem of computing the binary reachability relation. Our reduction technique not only preserves reachability, like [21], but additionally enables the reconstruction of the reachability relation.

Our expressivity result generalises analogous results for TA [11, 13, 15, 18] and PTDA [12]. The proof of [11] for TA has high technical difficulty and does not yield complexity bounds. The proof of [13] for TA uses an automata representation for sets of clock valuations; the idea of *reset-point semantics* employed in [13] is analogous to using registers instead of clocks. The paper [15] elegantly expresses the reachability relation for TA with *clock difference relations* (CDR) over the fractional values of clocks. It is remarkable that the formulas expressing the reachability relations that we obtain are of the same shape as CDR. The recent paper [18] shows that the TA binary reachability relation can be expressed in the same fragment of hybrid linear arithmetic that we use for TPDA, which we find very intriguing. Their proof converts the integer value of clocks into counters, and then observes that, thanks to the specific reset policy of clocks, these counter machines have a semilinear reachability relation; the latter is proved by encoding the value of counters into the language. In our proof, we bring the encoding of the integer value of clocks into the language to the forefront, via the introduction of the ternary reachability relation. The proof of [12] for PTDA also separates clocks into their integer and fractional part. It is not clear how any of the previous approaches could handle a timed stack.

Another approach for computing the reachability relation for TPDA would be to reduce it directly to a more expressive register model, such as *timed register pushdown automata* (TRPDA) [8, 9], which considers both integer $(\mathbb{Z}, \leq, +1)$ and rational registers $(\mathbb{Q}_{\geq 0}, \leq)$. While

such a reduction for the reachability problem is possible since (the integer part of) large clock values can be "forgotten", e.g., along the lines of [8], this does not hold anymore if we want to preserve the reachability relation. For this reason, in the present work we first remove the integer part of clocks (by encoding it in the untimed language) and then we reduce to RPDA, which have only fractional registers and no integer register, and are thus easier to analyse than TRPDA[4]. The method of quantifier elimination was recently applied to the analysis of another timed model, namely *timed communicating automata* [6].

Finally, another expressive extension of TA, called *recursive timed automata* (RTA), has been proposed [20, 3]. RTA use a timed stack to store the current clock valuation, which does not evolve as time elapses and can be restored at the time of pop. This facility makes RTA expressively incomparable to all models previously mentioned.

Missing proofs can be found in the technical report [10].

Notations. Let $\mathbb{Q}$, $\mathbb{Q}_{\geq 0}$, $\mathbb{Z}$, and $\mathbb{N}$ denote the rationals, the non-negative rationals, the integers, and the natural numbers; let $\mathbb{I} = \mathbb{Q}_{\geq 0} \cap [0, 1)$ be the unit rational interval. Let $\equiv_m$ denote the congruence modulo $m \in \mathbb{N} \setminus \{0\}$ in $\mathbb{Z}$. For $a \in \mathbb{Q}$, let $\lfloor a \rfloor \in \mathbb{Z}$ denote the largest integer k s.t. $k \leq a$, and let $\{a\} = a - \lfloor a \rfloor$ denote its fractional part. Let $\mathbb{1}_{C?}$, for a condition C, be 1 if C holds, and 0 otherwise.

2 Linear arithmetic and quantifier elimination

Consider the two-sorted structure $\mathcal{A} = \mathcal{A}_{\mathbb{Z}} \uplus \mathcal{A}_{\mathbb{Q}}$, where $\mathcal{A}_{\mathbb{Z}} = (\mathbb{Z}, \leq, (\equiv_m)_{m\in\mathbb{N}}, +, (k)_{k\in\mathbb{Z}})$ and $\mathcal{A}_{\mathbb{Q}} = (\mathbb{Q}, \leq, +, (k)_{k\in\mathbb{Q}})$. We consider "+" as a binary function, and we have a constant k for every integer/rational number. By *linear arithmetic*, denoted $\mathcal{L}_{\mathbb{Z},\mathbb{Q}}$, we mean the two-sorted first-order language in the vocabulary of $\mathcal{A}$. Restriction to the integer sort yields Presburger arithmetic $\mathcal{L}_{\mathbb{Z}}$ (*integer formulas*), and restriction to the rational sort yields linear rational arithmetic $\mathcal{L}_{\mathbb{Q}}$ (*rational formulas*). We assume constants are encoded in binary.

Two formulas are *equivalent* if they are satisfied by the same valuations. It is well-known that the theories of $\mathcal{A}_{\mathbb{Z}}$ [17] and $\mathcal{A}_{\mathbb{Q}}$ [14] admit effective elimination of quantifiers: Every formula can effectively be transformed in an equivalent quantifier-free one. Therefore, the theory of $\mathcal{A}$ also admits quantifier elimination, by the virtue of the following general fact (when speaking of a structure admitting quantifier elimination, we have in mind its theory).

▶ **Lemma 1.** *If the structures $\mathcal{A}_1$ and $\mathcal{A}_2$ admit (effective) elimination of quantifiers, then the two-sorted structure $\mathcal{A}_1 \uplus \mathcal{A}_2$ also does so. For conjunctive formulas, the complexity is the maximum of the two complexities.*

For clock constraints, we will use the first-order language over the two sorted structure $\mathcal{A}^{\mathrm{c}} = \mathcal{A}^{\mathrm{c}}_{\mathbb{N}} \uplus \mathcal{A}^{\mathrm{c}}_{\mathbb{I}}$, where the integer sort is restricted to $\mathcal{A}^{\mathrm{c}}_{\mathbb{N}} = (\mathbb{N}, \leq, (\equiv_m)_{m\in\mathbb{N}}, +1, 0)$ – the domain is now $\mathbb{N}$ and full addition "+" is replaced by the unary successor operation "+1") – and the rational sort to $\mathcal{A}^{\mathrm{c}}_{\mathbb{I}} = (\mathbb{I}, \leq, 0)$ – the domain is now the unit interval, there is no addition, and the only constant is 0. Let $\mathcal{L}^{\mathrm{c}}_{\mathbb{N},\mathbb{I}}$ be such a sub-logic. (As syntactic sugar we allow to use addition of arbitrary, even negative, integer constants in integer formulas, e.g. $x - 4 \leq y + 2$.) As before, $\mathcal{L}^{\mathrm{c}}_{\mathbb{N}}$ and $\mathcal{L}^{\mathrm{c}}_{\mathbb{I}}$ are the restrictions to the respective sorts. All the sub-logics above admit effective elimination of quantifiers.

[4] TRPDA are more general than RPDA– cyclic order atoms can be interpreted into $(\mathbb{Q}_{\geq 0}, \leq)$. The binary reachability relation for TRPDA can be computed by refining the reductions of [9] used for deciding the reachability problem. However, we do not know how to use the reachability relation of TRPDA to compute that of TPDA.

▶ **Lemma 2.** *The structures $\mathcal{A}^{\mathrm{c}}_{\mathbb{N}}$ and $\mathcal{A}^{\mathrm{c}}_{\mathbb{I}}$ admit effective elimination of quantifiers. For $\mathcal{A}^{\mathrm{c}}_{\mathbb{N}}$ the complexity is singly exponential for conjunctive formulas, while for $\mathcal{A}^{\mathrm{c}}_{\mathbb{I}}$ is quadratic.*

Notice that since $\mathcal{L}^{\mathrm{c}}_{\mathbb{N}}$ is a fragment of Presburger arithmetic $\mathcal{L}_{\mathbb{Z}}$, we could apply the quantifier elimination for $\mathcal{L}_{\mathbb{Z}}$ to get a quantifier-free $\mathcal{L}_{\mathbb{Z}}$ formula. Our result is stronger since we get a quantifier-free formula of the more restrictive fragment $\mathcal{L}^{\mathrm{c}}_{\mathbb{N}}$.

▶ **Corollary 3.** *The structure $\mathcal{A}^{\mathrm{c}}$ admits effective quantifier elimination. The complexity is exponential for conjunctive formulas.*

3 Timed pushdown automata

Clock constraints. Let X be a finite set of clocks. We consider constraints which can separately speak about the integer $\lfloor x \rfloor$ and fractional value $\{x\}$ of a clock $x \in X$. A *clock constraint* over X is a boolean combination of *atomic clock constraints* of one of the forms

	(integer)	(modular)	(fractional)
(non-diagonal)	$\lfloor x \rfloor \leq k$	$\lfloor x \rfloor \equiv_m k$	$\{x\} = 0$
(diagonal)	$\lfloor x \rfloor - \lfloor y \rfloor \leq k$	$\lfloor x \rfloor - \lfloor y \rfloor \equiv_m k$	$\{x\} \leq \{y\}$

where $x, y \in X$, $m \in \mathbb{N}$,a and $k \in \mathbb{Z}$. Since we allow arbitrary boolean combinations, we consider also the constraint **true**, which is always satisfied, and variants with any $\sim \in \{\leq, <, \geq, >\}$ in place of $\leq$. A *clock valuation* is a mapping $\mu \in \mathbb{Q}^X_{\geq 0}$ assigning a non-negative rational number to every clock in X; we write $\lfloor \mu \rfloor$ for the valuation in $\mathbb{N}^X$ s.t. $\lfloor \mu \rfloor(x) := \lfloor \mu(x) \rfloor$ and $\{\mu\}$ for the valuation in $\mathbb{I}^X$ s.t. $\{\mu\}(x) := \{\mu(x)\}$. For a valuation μ and a clock constraint φ we say that μ *satisfies* φ if φ is satisfied when integer clock values $\lfloor x \rfloor$ are evaluated according to $\lfloor \mu \rfloor$ and fractional values $\{x\}$ according to $\{\mu\}$.

▶ Remark (Clock constraints as quantifier-free $\mathcal{L}^{\mathrm{c}}_{\mathbb{N},\mathbb{I}}$ formulas). Up to syntactic sugar, a clock constraint over clocks $\{x_1, \ldots, x_n\}$ is the same as a quantifier-free $\mathcal{L}^{\mathrm{c}}_{\mathbb{N},\mathbb{I}}$ formula $\varphi(\lfloor x_1 \rfloor, \ldots, \lfloor x_n \rfloor, \{x_1\}, \ldots, \{x_n\})$ over n integer and n rationals variables.

▶ Remark (Classical clock constraints). Integer and fractional constraints subsume classical ones. For clocks x, y, since $x = \lfloor x \rfloor + \{x\}$ (and similarly for y)[5], $x - y \leq k$ for an integer k is equivalent to $(\lfloor x \rfloor - \lfloor y \rfloor \leq k \wedge \{x\} \leq \{y\}) \vee \lfloor x \rfloor - \lfloor y \rfloor \leq k - 1$, and similarly for $x \leq k$. On the other hand, the fractional constraint $\{x\} = 0$ is not expressible as a classical constraint.

▶ Remark ($\lfloor x \rfloor - \lfloor y \rfloor$ versus $\lfloor x - y \rfloor$). In the presence of fractional constraints, the expressive power would not change if, instead of atomic constraints $\lfloor x \rfloor - \lfloor y \rfloor \equiv_m k$ and $\lfloor x \rfloor - \lfloor y \rfloor \leq k$ speaking of the *difference of the integer parts*, we would choose $\lfloor x - y \rfloor \equiv_m k$ and $\lfloor x - y \rfloor \leq k$ speaking of the *integer part of the difference*, since the two are inter-expressible:

$$\lfloor x - y \rfloor = \lfloor x \rfloor - \lfloor y \rfloor - \mathbb{1}_{\{x\} < \{y\}?} \qquad \text{and} \qquad \{x - y\} = \{x\} - \{y\} + \mathbb{1}_{\{x\} < \{y\}?}. \tag{2}$$

The model. A *timed pushdown automaton* (TPDA) is a tuple $\mathcal{P} = \langle \Sigma, \Gamma, L, X, Z, \Delta \rangle$ where Σ is a finite input alphabet, Γ is a finite stack alphabet, L is a finite set of control locations, X is a finite set of *global clocks*, and Z is a finite set of *stack clocks* disjoint from X. The last item Δ is a set of transition rules $\langle \ell, \mathsf{op}, r \rangle$ with $\ell, r \in L$ control locations, where op determines the type of transition:

[5] We often identify a clock x with its value for simplicity of notation.

- *time elapse* $\mathsf{op} = \mathsf{elapse}$,
- *input* $\mathsf{op} = a \in \Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$ an input letter,
- *test* $\mathsf{op} = \varphi$ a *transition constraint* over clocks X,
- *reset* $\mathsf{op} = \mathsf{reset}(Y)$ with $Y \subseteq X$ a set of clocks to be reset,
- *push* $\mathsf{op} = \mathsf{push}(\alpha : \psi)$ with $\alpha \in \Gamma$ a stack symbol to be pushed on the stack under the *stack constraint* ψ over clocks $X \cup Z$, or
- *pop* $\mathsf{op} = \mathsf{pop}(\alpha : \psi)$ similarly as push.

We assume that every atomic constraint in a stack constraint contains some stack variable from Z. Throughout the paper, let x_0 be a global clock that is never reset (and thus measures the total elapsed time), and let z_0 be a stack clock that is 0 when pushed. A TPDA has *untimed stack* if the only stack constraint is **true**. Without push/pop operations, we obtain nondeterministic timed automata (TA).

▶ Remark (Complexity). For complexity estimations, we assume that constraints are conjunctions of atomic constraints, that constants therein are encoded in binary, that M is the maximal constant, and that all modular constraints use the same modulus M.

▶ Remark (Time elapse). The standard semantics of timed automata where time can elapse freely in every control location is simulated by adding explicit time elapse transitions $\langle \ell, \mathsf{elapse}, \ell \rangle$ for suitable locations ℓ. Our explicit modelling of the elapse of time will simplify the constructions in Sec. 4.

▶ Remark (Comparison with dtPDA). The dtPDA model [1] allows only one stack clock $Z = \{z\}$ and stack constraints of the form $z \sim k$. As shown in [8], this model is equivalent to TPDA with untimed stack. Our extension is two-fold. First, our definition of stack constraint is more liberal, since we allow more general *diagonal stack constraints* of the form $z - x \sim k$. Second, we also allow *modular* $\lfloor y \rfloor - \lfloor x \rfloor \equiv_m k$ and *fractional constraints* $\{x\} \sim \{y\}$, where clocks x, y can be either global or stack clocks. As demonstrated in Example 4 below, this model is not reducible to untimed stack, and thus TPDA are more expressive than dtPDA.

Semantics. Every stack symbol is equipped with a fresh copy of clocks from Z. At the time of $\mathsf{push}(\alpha : \psi)$, the push constraint ψ specifies possibly nondeterministically the initial value of all clocks in Z w.r.t. global clocks in X. Both global and stack clocks evolve at the same rate when a time elapse transition is executed. At the time of $\mathsf{pop}(\alpha : \psi)$, the pop constraint ψ specifies the final value of all clocks in Z w.r.t. global clocks in X. A *timed stack* is a sequence $w \in (\Gamma \times \mathbb{Q}_{\geq 0}^Z)^*$ of pairs (γ, μ), where γ is a stack symbol and μ is a valuation for stack clocks in Z. For a clock valuation μ and a set of clocks Y, let $\mu[Y \mapsto 0]$ be the same as μ except that clocks in Y are mapped to 0. For $\delta \in \mathbb{Q}_{\geq 0}$, let $\mu + \delta$ be the clock valuation which adds δ to the value of every clock, i.e., $(\mu + \delta)(x) := \mu(x) + \delta$, and for a timed stack $w = (\gamma_1, \mu_1) \cdots (\gamma_k, \mu_k)$, let $w + \delta$ be $(\gamma_1, \mu_1 + \delta) \cdots (\gamma_k, \mu_k + \delta)$. A *configuration* is a triple $\langle \ell, \mu, w \rangle \in L \times \mathbb{Q}_{\geq 0}^X \times (\Gamma \times \mathbb{Q}_{\geq 0}^Z)^*$ where ℓ is a control location, μ is a clock valuation over the global clocks X, and w is a timed stack. Let $\langle \ell, \mu, u \rangle, \langle r, \nu, v \rangle$ be two configurations. For every input symbol or time increment $a \in (\Sigma_\varepsilon \cup \mathbb{Q}_{\geq 0})$ we have a transition $\langle \ell, \mu, u \rangle \xrightarrow{a} \langle r, \nu, v \rangle$ whenever there exists a rule $\langle \ell, \mathsf{op}, r \rangle \in \Delta$ s.t. one of the following holds:

- $\mathsf{op} = \mathsf{elapse}$, $a \in \mathbb{Q}_{\geq 0}$, $\nu = \mu + a$, $v = u + a$.
- $\mathsf{op} = a \in \Sigma_\varepsilon$, $\nu = \mu$, $u = v$.
- $\mathsf{op} = \varphi$, $a = \varepsilon$, $\mu \models \varphi$, $\nu = \mu$, $u = v$.
- $\mathsf{op} = \mathsf{reset}(Y)$, $a = \varepsilon$, $\nu = \mu[Y \mapsto 0]$, $v = u$.
- $\mathsf{op} = \mathsf{push}(\gamma : \psi)$, $a = \varepsilon$, $\mu = \nu$, $v = u \cdot \langle \gamma, \mu_1 \rangle$ if $\mu_1 \in \mathbb{Q}_{\geq 0}^Z$ satisfies $(\mu, \mu_1) \models \psi$, where $(\mu, \mu_1) \in \mathbb{Q}_{\geq 0}^{X \cup Z}$ is the unique clock valuation that agrees with μ on X and with μ_1 on Z.
- $\mathsf{op} = \mathsf{pop}(\gamma : \psi)$, $a = \varepsilon$, $\mu = \nu$, $u = v \cdot \langle \gamma, \mu_1 \rangle$ provided that $\mu_1 \in \mathbb{Q}_{\geq 0}^Z$ satisfies $(\mu, \mu_1) \models \psi$.

A *timed word* is a sequence $w = \delta_1 a_1 \cdots \delta_n a_n \in (\mathbb{Q}_{\geq 0}\Sigma_\varepsilon)^*$ of alternating time elapses and input symbols; the one-step transition relation $\langle \ell, \mu, u\rangle \xrightarrow{a} \langle r, \nu, v\rangle$ is extended on timed words w as $\langle \ell, \mu, u\rangle \xrightarrow{w} \langle r, \nu, v\rangle$ in the natural way. The *timed language* from location ℓ to r is $L(\ell, r) := \left\{ \pi_\varepsilon(w) \in (\mathbb{Q}_{\geq 0}\Sigma)^* \;\middle|\; \langle \ell, \mu_0, \varepsilon\rangle \xrightarrow{w} \langle r, \mu_0, \varepsilon\rangle \right\}$ where $\pi_\varepsilon(w)$ removes the ε's from w and μ_0 is the valuation that assigns $\mu_0(x) = 0$ to every clock x. The corresponding *untimed language* $L^{\text{un}}(\ell, r)$ is obtained by removing the time elapses from $L(\ell, r)$.

▶ **Example 4.** Let L be the timed language of even length palindromes s.t. the time distance between every pair of matching symbols is an integer:

$$L = \{\delta_1 a_1 \cdots \delta_{2n} a_{2n} \mid \forall (1 \leq i \leq n) \cdot a_i = a_{2n-i+1} \wedge \delta_{i+1} + \cdots + \delta_{2n-i+1} \in \mathbb{N}\}\,.$$

L can be recognised by a TPDA over input and stack alphabet $\Sigma = \Gamma = \{a, b\}$, with locations ℓ, r, no global clock, one stack clock $Z = \{z\}$, and the following transition rules (omitting some intermediate states), where α ranges over $\{a, b\}$:

$$\langle \ell, \alpha; \mathsf{push}(\alpha : \{z\} = 0), \ell\rangle \qquad \langle \ell, \varepsilon, r\rangle$$
$$\langle r, \alpha; \mathsf{pop}(\alpha : \{z\} = 0), r\rangle \qquad \langle \ell, \mathsf{elapse}, \ell\rangle, \langle r, \mathsf{elapse}, r\rangle$$

We have $L = L(\ell, r)$. Since L cannot be recognised by TPDA with untimed stack (cf. [21]), fractional stack constraints strictly increase the expressive power of the model.

The reachability relation. The *Parikh image* of a timed word w is the mapping $\text{PI}_w \in \mathbb{N}^\Sigma$ s.t. $\text{PI}_w(a)$ is the number of a's in w, ignoring the elapse of time and ε's. For two control locations ℓ, r, clock valuations $\mu, \nu \in \mathbb{Q}^X_{\geq 0}$, and a timed word $w \in (\mathbb{Q}_{\geq 0}\Sigma_\varepsilon)^*$, we write $\mu \overset{w}{\rightsquigarrow}_{\ell r} \nu$ if $\langle \ell, \mu, \varepsilon\rangle \xrightarrow{w} \langle r, \nu, \varepsilon\rangle$. We overload the notation and, for $\pi \in \mathbb{N}^\Sigma$, we write $\mu \overset{\pi}{\rightsquigarrow}_{\ell r} \nu$ if there exists a timed word w s.t. $\mu \overset{w}{\rightsquigarrow}_{\ell r} \nu$ and $\pi = \text{PI}_w$. We see $\{\rightsquigarrow_{\ell r}\}_{\ell, r \in L}$ as a family of subsets of $\mathbb{Q}^X_{\geq 0} \times \mathbb{N}^\Sigma \times \mathbb{Q}^X_{\geq 0}$ and we call it the *ternary reachability relation*.

Let $\left\{\psi_{\ell r}(\lfloor \overline{x}\rfloor, \{\overline{x}\}, \overline{f}, \lfloor \overline{y}\rfloor, \{\overline{y}\})\right\}_{\ell, r \in L}$ be a family of $\mathcal{L}_{\mathbb{Z},\mathbb{Q}}$ formulas, where $\lfloor \overline{x}\rfloor, \lfloor \overline{y}\rfloor$ represent the integer values of initial and final clocks, $\{\overline{x}\}, \{\overline{y}\}$ their fractional values, and $\overline{f}$ letter counts. The reachability relation $\{\rightsquigarrow_{\ell r}\}_{\ell, r \in L}$ is *expressed* by the family of formulas $\{\psi_{\ell r}\}_{\ell, r \in L}$ if the following holds: For every control locations $\ell, r \in L$, clock valuations $\mu, \nu \in \mathbb{Q}^X_{\geq 0}$ and $\pi \in \mathbb{N}^\Sigma$, $\mu \overset{\pi}{\rightsquigarrow}_{\ell r} \nu$ holds, if, and only if, $(\lfloor \mu\rfloor, \{\mu\}, \pi, \lfloor \nu\rfloor, \{\nu\}) \models \psi_{\ell r}$ holds.

Main results. As the main result of the paper we show that the reachability relation of TPDA and TA is expressible in linear arithmetic $\mathcal{L}_{\mathbb{Z},\mathbb{Q}}$.

▶ **Theorem 5.** *The reachability relation of a* TPDA *is expressed by a family of existential $\mathcal{L}_{\mathbb{Z},\mathbb{Q}}$ formulas computable in double exponential time. For* TA*, the complexity is exponential.*

This is a strengthening of analogous results for TA [11, 18] since our model, even without stack, is more expressive than classical TA due to fractional constraints. As a side effect of the proofs we get:

▶ **Theorem 6.** *Untimed* TPDA *languages $L^{un}(\ell, r)$ are effectively context-free.*

The following two sections are devoted to proving the two theorem above.

4 Fractional TPDA

A TPDA is *fractional* if it contains only fractional constraints. We show that computing the reachability relation reduces to the same problem for fractional TPDA. Our transformation is done in three steps, each one further restricting the set of allowed constraints.

A The TPDA is *push-copy*, that is, push operations can only copy global clocks into stack clocks. There is one stack clock z_x for each global clock x, and the only push constraint is

$$\psi_{\mathsf{copy}}(\overline{x}, \overline{z}_{\overline{x}}) \equiv \bigwedge_{x \in X} \lfloor z_x \rfloor = \lfloor x \rfloor \wedge \{z_x\} = \{x\}\,. \tag{3}$$

By pushing copies of global clocks into the stack, we can postpone checking all non-trivial stack constraints to the time of pop. This steps uses quantifier elimination. The blowup of the number of pop constraints and stack alphabet is exponential.

B The TPDA is *pop-integer-free*, that is, pop transitions do not contain integer constraints. The construction is similar to a construction from [8] and is presented in the technical report [10]. Removing pop integer constraints is crucial towards removing all integer clocks (modulo constraints will be removed by the next step). This step strongly relies on the fact that stack clocks are copies of global clocks, which allows one to remove integer pop constraints by reasoning about analogous constraints between global clocks at the time of push and their future values at the time of pop, thus bypassing the stack altogether. We introduce one global clock for each integer pop constraint, exponentially many locations in the number of clocks and pop constraints, and exponentially many stack symbols in the number of pop constraints. When combined with the previous step, altogether exponentially many new clocks are introduced, and doubly exponentially many locations/stack symbols. It is remarkable that pop integer constraints can be removed by translating them into finitely many transition constraints on global clocks.

C The TPDA is fractional. All integer clocks are removed. In order to recover their values (which are needed to express the reachability relation), a special symbol $\checkmark_x$ is produced when an integer clock elapses one time unit. This step introduces a further exponential blowup of control locations w.r.t. global clocks and polynomial in the maximal constant M. The overall complexity of control locations thus stays double exponential.

By **A**+**B**+**C** (in this order, since the latter properties are ensured assuming the previous ones), we get the following theorem.

▶ **Theorem 7.** *A* TPDA $\mathcal{P}$ *can be effectively transformed into a fractional* TPDA $\mathcal{Q}$ *s.t. a family of* $\mathcal{L}_{\mathbb{Z},\mathbb{Q}}$ *formulas* $\{\varphi_{\ell r}\}$ *expressing the reachability relation of* $\mathcal{P}$ *can effectively be computed from a family of* $\mathcal{L}_{\mathbb{Z},\mathbb{Q}}$ *formulas* $\{\varphi'_{\ell' r'}\}$ *expressing the reachability relation of* $\mathcal{Q}$*. The number of control locations and the size of the stack alphabet in* $\mathcal{Q}$ *have a double exponential blowup, and the number of clocks has an exponential blowup.*

If there is no stack, then we do not need the first two steps, and we can do directly **C**.

▶ **Corollary 8.** *The reachability relation of push-copy* TPDA/TA *effectively reduces to the reachability relation of fractional* TPDA/TA *with an exponential blowup in control locations.*

(A) The TPDA is push-copy

Let $K_{\leq}$ be the non-strict variant of the ternary cyclic order K from (1), defined as $K_{\leq}(a,b,c) \equiv K(a,b,c) \vee a = b \vee b = c$ for $a, b, c \in \mathbb{I}$. Let $\psi_{\mathsf{push}}(\overline{x}, \overline{z})$ be a push constraint, and let $\psi_{\mathsf{pop}}(\overline{x}', \overline{z}')$ be the corresponding pop constraint. Since stack clock z_0 is 0 when pushed

on the stack, z'_0 is the total time elapsed between push and pop; let $\overline{z}'_0 = (z'_0, \dots, z'_0)$ (the length of which depends on the context). Let $\overline{z}'_{\overline{x}}$ be a vector of stack variables representing the value of *global clocks* at the time of pop, provided they were not reset since the matching push. Since all clocks evolve at the same rate, for every global clock x and stack clock z, we have

$$x = z'_x - z'_0 \qquad \text{and} \qquad z = z' - z'_0. \tag{4}$$

If at the time of push, instead of pushing $\overline{z}$, we push on the stack a copy of global clocks $\overline{x}$, then at the time of pop it suffices to check that the following formula holds

$$\psi'_{\mathsf{pop}}(\overline{x}', \overline{z}'_{\overline{x}}) \;\equiv\; \exists \overline{z}' \geq \overline{0} \cdot \psi_{\mathsf{push}}(\overline{z}'_{\overline{x}} - \overline{z}'_0, \overline{z}' - \overline{z}'_0) \wedge \psi_{\mathsf{pop}}(\overline{x}', \overline{z}'). \tag{5}$$

Note that the assumption that $z_0 = 0$ at the time of push makes the existential quantification satisfiable by exactly one value of z'_0, namely the total time elapsed between push and pop. However, $\psi_{\mathsf{push}}(\overline{z}'_{\overline{x}} - \overline{z}'_0, \overline{z}' - \overline{z}'_0)$ is not a constraint anymore, since variables are replaced by differences of variables. We resolve this issue by showing that the latter is in fact equivalent to a clock constraint. Thanks to (4), for every clock x we have $\lfloor x \rfloor = \lfloor z'_x - z'_0 \rfloor, \{x\} = \{z'_x - z'_0\}$, and $\lfloor z \rfloor = \lfloor z' - z'_0 \rfloor, \{z\} = \{z' - z'_0\}$. Thus, a fractional constraint $\{y\} \leq \{z\}$ in ψ_{push} is equivalent to $\{z'_y - z'_0\} \leq \{z' - z'_0\}$, which is in turn equivalent to $C = K_{\leq}(\{z'_0\}, \{z'_y\}, \{z'\})$, which is definable from $\leq$. Moreover, $\lfloor y \rfloor - \lfloor z \rfloor = \lfloor z'_y - z'_0 \rfloor - \lfloor z' - z'_0 \rfloor = (z'_y - z'_0 - \{z'_y - z'_0\}) - (z' - z'_0 - \{z' - z'_0\}) = (z'_y - z') - \{z'_y - z'_0\} + \{z' - z'_0\} = (z'_y - z') - \{z'_y - z'\} + \mathbb{1}_{D?} = \lfloor z'_y - z' \rfloor + \mathbb{1}_{D?}$, with $D = C \wedge \{z'_y\} \neq \{z'\}$. (Notice that $\lfloor z'_0 \rfloor$ disappears in this process: This is not a coincidence, since diagonal integer/modular/fractional constraints are invariant under the elapse of an *integer* amount of time.) Thus by (2) we obtain a constraint $\psi'_{\mathsf{push}}(\overline{z}'_{\overline{x}}, \overline{z}')$ logically equivalent to $\psi_{\mathsf{push}}(\overline{z}'_{\overline{x}} - \overline{z}'_0, \overline{z}' - \overline{z}'_0)$, and, by separating the fractional and integer constraints (cf. Remark 3), $\psi'_{\mathsf{pop}}(\overline{x}', \overline{z}'_{\overline{x}}) \equiv \exists \lfloor \overline{z}' \rfloor, \{\overline{z}'\} \cdot \psi'_{\mathsf{push}}(\lfloor \overline{z}'_{\overline{x}} \rfloor, \{\overline{z}'_{\overline{x}}\}, \lfloor \overline{z}' \rfloor, \{\overline{z}'\}) \wedge \psi_{\mathsf{pop}}(\lfloor \overline{x}' \rfloor, \{\overline{x}'\}, \lfloor \overline{z}' \rfloor, \{\overline{z}'\})$. By Corollary 3, we can perform quantifier elimination and we obtain a logically equivalent clock constraint of exponential size (in DNF) $\xi_{\psi_{\mathsf{push}}, \psi_{\mathsf{pop}}}(\lfloor \overline{x}' \rfloor, \{\overline{x}'\}, \lfloor \overline{z}'_{\overline{x}} \rfloor, \{\overline{z}'_{\overline{x}}\})$, where the subscript indicates that this formula depends on the pair $(\psi_{\mathsf{push}}, \psi_{\mathsf{pop}})$ of push and pop constraints. The construction of $\mathcal{P}'$ consists in checking $\xi_{\psi_{\mathsf{push}}, \psi_{\mathsf{pop}}}$ in place of ψ_{pop}, assuming that the push constraint was ψ_{push}. The latter is replaced by ψ_{copy}. Control states are the same in the two automata; we can break down the $\xi_{\psi_{\mathsf{push}}, \psi_{\mathsf{pop}}}$ in DNF and record each conjunct in the stack, yielding a new stack alphabet of exponential size.

▶ **Lemma 9.** *Let $\{\rightsquigarrow_{\ell r}\}_{\ell, r \in L}$, $\{\rightsquigarrow'_{\ell r}\}_{\ell, r \in L}$ be the reachability relations of $\mathcal{P}$, resp., $\mathcal{P}'$. Then, $\rightsquigarrow_{\ell r} = \rightsquigarrow'_{\ell r}$ for every $\ell, r \in L$, and $\mathcal{P}'$ has stack alphabet exponential in the size of $\mathcal{P}$.*

(C) The TPDA is fractional

Assume that the TPDA $\mathcal{P}$ is both push-copy (**A**) and pop-integer-free (**B**). We remove diagonal integer $\lfloor y \rfloor - \lfloor x \rfloor \sim k$ and modulo $\lfloor y \rfloor - \lfloor x \rfloor \equiv_m k$ constraints on global clocks x, y as in TA [2]. In the rest of the section, transition and stack constraints of $\mathcal{P}$ are of the form

$$\begin{array}{lllll}
\text{(trans.)} & \lfloor x \rfloor \leq k, & \lfloor x \rfloor \equiv_m k, & \{x\} = 0, & \{x\} \leq \{y\}, \quad (6) \\
\text{(push)} & \lfloor z_x \rfloor = \lfloor x \rfloor, & & & \{z_x\} = \{x\}, \quad (7) \\
\text{(pop)} & & \lfloor y \rfloor - \lfloor z_x \rfloor \equiv_m k, & \{z_x\} = 0, & \{y\} \leq \{z_x\}, \quad (8) \\
 & & \lfloor z_y \rfloor - \lfloor z_x \rfloor \equiv_m k, & & \{z_y\} \leq \{z_x\}.
\end{array}$$

Unary abstraction. We replace the integer value of clocks by their *unary abstraction*: Valuations $\mu, \nu \in \mathbb{Q}_{\geq 0}^X$ are *M-unary equivalent*, written $\mu \approx_M \nu$, if, for every clock $x \in X$, $\lfloor \mu(x) \rfloor \equiv_M \lfloor \nu(x) \rfloor$ and $\lfloor \mu(x) \rfloor \leq M \Leftrightarrow \lfloor \nu(x) \rfloor \leq M$. Let Λ_M be the (finite) set of M-unary equivalence classes of clock valuations. For $\lambda \in \Lambda_M$ we abuse notation and write $\lambda(x)$ to indicate $\mu(x)$ for some $\mu \in \lambda$, where the choice of representative μ does not matter. We write $\lambda[Y \mapsto 0]$ for the equivalence class of $\nu[Y \mapsto 0]$ and we write $\lambda[x \mapsto x+1]$ for the equivalence class of $\nu[x \mapsto \nu(x)+1]$, for some $\nu \in \lambda$ (whose choice is irrelevant). Let $\varphi_\lambda(\overline{x}) \equiv \bigwedge_{x \in X} \lfloor x \rfloor \equiv_M \lambda(x) \wedge (\lfloor x \rfloor < M \Leftrightarrow \lambda(x) < M)$ say that clocks belong to λ. For φ containing transition constraints of the form (6), $\varphi|_\lambda$ is φ where every integer $\lfloor x \rfloor \leq k$ or modulo constraint $\lfloor x \rfloor \equiv_M k$ is uniquely resolved to be **true** or **false** by replacing every occurrence of $\lfloor x \rfloor$ with $\lambda(x)$. Similarly, for ψ a pop constraint of the form (8), $\psi|_{\lambda_{\mathsf{push}}, \lambda_{\mathsf{pop}}}$ is obtained by resolving modulo constraints $\lfloor y \rfloor - \lfloor z_x \rfloor \equiv_M k$ and $\lfloor z_y \rfloor - \lfloor z_x \rfloor \equiv_M k$ to be **true** or **false** by replacing every occurrence of $\lfloor y \rfloor$ by its abstraction at the time of pop $\lambda_{\mathsf{pop}}(y)$, and every occurrence of $\lfloor z_x \rfloor$ by $\lambda_{\mathsf{push}}(x) + \Delta(\lambda_{\mathsf{push}}, \lambda_{\mathsf{pop}})$, i.e., the initial value of clock x plus the total integer time elapsed until the pop, defined as $\Delta(\lambda_{\mathsf{push}}, \lambda_{\mathsf{pop}}) = \lambda_{\mathsf{pop}}(x_0) - \lambda_{\mathsf{push}}(x_0) - \mathbb{1}_{\{z_0\} > \{x_0\}?}$, i.e., we take the difference of x_0 (which is never reset) between push and pop, possibly corrected by "-1" if the last time unit only partially elapsed; the substitution for $\lfloor z_y \rfloor$ is analogous. Fractional constraints are unchanged.

Sketch of the construction. Given a push-copy and pop-integer-free TPDA $\mathcal{P}$, we build a fractional TPDA $\mathcal{Q}$ over the extended alphabet $\Sigma' = \Sigma \cup \{\checkmark_x \mid x \in X\}$ as follows. We eliminate integer $\lfloor x \rfloor \leq k$ and modulo constraints $\lfloor x \rfloor \equiv_M k$ by storing in the control the M-unary abstraction λ. To reconstruct the reachability relation of $\mathcal{P}$, we store the set of clocks Y which will not be reset anymore in the future. Thus, control locations L' of $\mathcal{Q}$ are of the form $\langle \ell, \lambda, Y \rangle$. In order to properly update the M-unary abstraction λ, the automaton checks how much time elapses by looking at the fractional values of clocks. When λ is updated to $\lambda[x \mapsto x+1]$, a symbol $\checkmark_x$ is optionally produced if $x \in Y$ was guessed not to be reset anymore in the future. A test transition $\langle \ell, \varphi, r \rangle$ is simulated by $\langle \langle \ell, \lambda, Y \rangle, \varphi|_\lambda, \langle r, \lambda, Y \rangle \rangle$. A push-copy transition $\langle \ell, \mathsf{push}(\alpha : \psi_{\mathsf{copy}}), r \rangle$ is simulated by $\langle \langle \ell, \lambda, Y \rangle, \mathsf{push}(\langle \alpha, \lambda \rangle : \bigwedge_{x \in X} \{z_0\} = 0 \wedge \{z_x\} = \{x\}), \langle r, \lambda, Y \rangle \rangle$ copying only the fractional parts and the unary class of global clocks. A pop-integer-free transition $\langle \ell, \mathsf{pop}(\alpha : \psi), r \rangle$ is simulated by $\langle \langle \ell, \lambda_{\mathsf{pop}}, Y \rangle, \mathsf{pop}(\langle \alpha, \lambda_{\mathsf{push}} \rangle : \psi|_{\lambda_{\mathsf{push}}, \lambda_{\mathsf{pop}}}), \langle r, \lambda_{\mathsf{pop}}, Y \rangle \rangle$. The reachability formula $\varphi_{\ell r}$ for $\mathcal{P}$ can be expressed by guessing the initial and final abstractions λ, μ, and the set of clocks Y which is never reset in the run. For clocks $x \in Y$, we must observe precisely $\lfloor x' \rfloor - \lfloor x \rfloor$ ticks $\checkmark_x$, and for the others, $\lfloor x' \rfloor$, where x is the initial and x' the final value. Let $g_x^Y = \lfloor x' \rfloor - \lfloor x \rfloor$ if $x \in Y$, and $\lfloor x' \rfloor$ otherwise.

▶ **Lemma 10.** *Let $\{\psi_{\ell' r'}(\{\overline{x}\}, (\overline{f}, \overline{g}), \{\overline{x}'\})\}_{\ell', r' \in L'}$ express the reachability relation of the fractional $\mathcal{Q}$ where $\{\overline{x}\}, \{\overline{x}'\}$ are the fractional values of clocks (we ignore integer values), $\overline{f}$ is the Parikh image of the original input letters from Σ, and $\overline{g}$ of the new input letters $\checkmark_x$'s. The reachability relation of $\mathcal{P}$ is expressed by $\varphi_{\ell r}(\lfloor \overline{x} \rfloor, \{\overline{x}\}, \overline{f}, \lfloor \overline{x}' \rfloor, \{\overline{x}'\}) \equiv \bigvee_{\lambda, Y, \mu} \varphi_\lambda(\lfloor \overline{x} \rfloor) \wedge \psi_{\langle \ell, \lambda, Y \rangle \langle r, \mu, X \rangle}(\{\overline{x}\}, (\overline{f}, \overline{g}^Y), \{\overline{x}'\})$.*

5 From fractional TPDA to register PDA

The aim of this section is to prove the following result which, together with Theorem 7, completes the proof of our main result Theorem 5.

Figure 1 (a) Relation K. (b) The cyclic difference $b \ominus a$.

▶ **Theorem 11.** *The fractional reachability relation of a fractional* TPDA $\mathcal{P}$ *is expressed by existential* $\mathcal{L}_{\mathbb{Z},\mathbb{Q}}$ *formulas, computable in time exponential in the number of clocks and polynomial in the number of control locations and stack alphabet.*

Cyclic atoms. We model fractional clock values by the *cyclic atoms* structure $(\mathbb{I}, K)$ with universe $\mathbb{I} = \mathbb{Q} \cap [0, 1)$, where K is the ternary cyclic order (1). Since K is invariant under cyclic shift, it is convenient to think of elements of $\mathbb{I}$ as placed clockwise on a circle of unit perimeter; cf. Fig. 1(a). An *automorphism* is a bijection α that preserves and reflects K, i.e., $K(a, b, c)$ iff $K(\alpha(a), \alpha(b), \alpha(c))$; automorphisms are extended to tuples $\mathbb{I}^n$ point-wise. Cyclic atoms are homogeneous [16] and thus $\mathbb{I}^n$ splits into exponentially many *orbits* $\mathrm{Orb}(\mathbb{I}^n)$, where $u, v \in \mathbb{I}^n$ are in the same orbit if some automorphism maps u to v. An orbit is an equivalence class of indistinguishable tuples, similarly as regions for clock valuations, but in a different logical structure: For instance $(0.2, 0.3, 0.7)$, $(0.7, 0.2, 0.3)$, and $(0.8, 0.2, 0.3)$ belong to the same orbit, while $(0.2, 0.3, 0.3)$ belongs to a different orbit.

Register PDA. We extend classical pushdown automata with additional $\mathbb{I}$-valued *registers*, both in the finite control (i.e., global registers) and in the stack. Registers can be compared by quantifier-free formulas with equality and K, called K*-constraints*. For simplicity, we assume that there are the same number of global and stack registers. A *register pushdown automaton* (RPDA) is a tuple $\mathcal{Q} = \langle \Sigma, \Gamma, L, X, Z, \Delta \rangle$ where Σ is a finite input alphabet, Γ is a finite stack alphabet, L is a finite set of control locations, X is a finite set of *global registers*, Z is a finite set of *stack registers*, and the last item Δ is a set of transition rules $\langle \ell, \mathsf{op}, r \rangle$ with $\ell, r \in L$ control locations, where op is either: 1) an input letter $a \in \Sigma_\varepsilon$, 2) a $2k$-ary K-constraint $\psi(\overline{x}, \overline{x}')$ relating pre- and post-values of global registers, 3) a push operation $\mathsf{push}(\alpha : \psi(\overline{x}, \overline{z}))$ with $\alpha \in \Gamma$ a stack symbol to be pushed on the stack under the $2k$-ary K-constraint ψ relating global $\overline{x}$ and stack $\overline{z}$ registers, or 4) a pop operation $\mathsf{pop}(\alpha : \psi(\overline{x}, \overline{z}))$, similarly as push. We consider RPDA as symbolic representations of classical PDA with infinite sets of control states $\widetilde{L} = L \times \mathbb{I}^X$ and infinite stack alphabet $\widetilde{\Gamma} = \Gamma \times \mathbb{I}^Z$. A *configuration* is thus a tuple $\langle \ell, \mu, w \rangle \in L \times \mathbb{I}^X \times \widetilde{\Gamma}^*$ where ℓ is a control location, μ is a valuation of the global registers, and w is the current content of the stack. Let $\langle \ell, \mu, u \rangle, \langle r, \nu, v \rangle$ be two configurations. For every input symbol $a \in \Sigma_\varepsilon$ we have a transition $\langle \ell, \mu, u \rangle \xrightarrow{a} \langle r, \nu, v \rangle$ whenever there exists a rule $\langle \ell, \mathsf{op}, r \rangle \in \Delta$ s.t. one of the following holds: 1) $\mathsf{op} = a \in \Sigma_\varepsilon$, $\mu = \nu$, $u = v$, or 2) $\mathsf{op} = \varphi$, $a = \varepsilon$, $(\mu, \nu) \models \varphi$, $u = v$, or 3) $\mathsf{op} = \mathsf{push}(\gamma : \psi)$, $a = \varepsilon$, $\mu = \nu$, $v = u \cdot \langle \gamma, \mu_1 \rangle$ if $\mu_1 \in \mathbb{I}^Z$ satisfies $(\mu, \mu_1) \models \psi$, or 4) $\mathsf{op} = \mathsf{pop}(\gamma : \psi)$, $a = \varepsilon$, $\mu = \nu$, $u = v \cdot \langle \gamma, \mu_1 \rangle$ if $\mu_1 \in \mathbb{I}^Z$ satisfies $(\mu, \mu_1) \models \psi$.

Reachability relation. The reachability relations $\mu \overset{w}{\rightsquigarrow}_{\ell r} \nu$ and $\mu \overset{f}{\rightsquigarrow}_{\ell r} \nu$ are defined as for TPDA by extending one-step transitions $\langle \ell, \mu, u \rangle \xrightarrow{a} \langle r, \nu, v \rangle$ to words $w \in \Sigma^*$ and their Parikh images $f = \mathrm{PI}_w \in \mathbb{N}^\Sigma$. Thus, $\mu \overset{f}{\rightsquigarrow}_{\ell r} \nu$ is a subset of $\mathbb{I}^X \times \mathbb{N}^\Sigma \times \mathbb{I}^X$, which is furthermore

invariant under orbits. In the following let X' be a copy of global clocks. An initial valuation μ belongs to $\mathbb{I}^X$, a final valuation ν to $\mathbb{I}^{X'}$, and the joint valuation (μ,ν) belongs to $\mathbb{I}^{X\times X'}$. The following two lemmas hold for RPDA with homogeneous atoms; cf. [7], or Sec. 9 in [4].

▶ **Lemma 12.** *If $(\mu,\nu),(\mu',\nu')$ belong to the same orbit of $\mathbb{I}^{X\times X'}$, then $\mu \overset{f}{\rightsquigarrow}_{\ell r} \nu$ iff $\mu' \overset{f}{\rightsquigarrow}_{\ell r} \nu'$.*

▶ **Lemma 13.** *Given a RPDA $\mathcal{Q}$ one can construct a context-free grammar G of exponential size with nonterminals of the form $X_{\ell ro}$, for control locations ℓ, r and an orbit $o \in \mathrm{Orb}(\mathbb{I}^{X\times X'})$, recognising the language $L(X_{\ell ro}) = \left\{ \pi_\Sigma(w) \in \Sigma^* \;\middle|\; \exists (\mu,\nu) \in o \cdot \mu \overset{w}{\rightsquigarrow}_{\ell r} \nu \right\}$, where $\pi_\Sigma(w)$ is w without the ε's. Consequently, RPDA recognise context-free languages.*

▶ **Lemma 14** (Theorem 4 of [22])**.** *The Parikh image of $L(X_{\ell ro})$ is expressed by an existential Presburger formula $\varphi^{\mathbb{Z}}_{\ell ro}$ computable in time linear in the size of the grammar.*

▶ **Corollary 15.** *Let $\varphi^{\mathbb{I}}_o$ be the characteristic K-constraint of the orbit $o \in \mathrm{Orb}(\mathbb{I}^{X\times X'})$. The reachability relation $\rightsquigarrow_{\ell r}$ of an RPDA $\mathcal{Q}$ is expressed by $\varphi_{\ell r}(\overline{x},\overline{f},\overline{x}') \equiv \bigvee_{o\in \mathrm{Orb}(\mathbb{I}^{X\times X'})} \varphi^{\mathbb{Z}}_{\ell ro}(\overline{f}) \wedge \varphi^{\mathbb{I}}_o(\overline{x},\overline{x}')$. The size of $\varphi_{\ell r}$ is exponential in the size of $\mathcal{Q}$.*

Proof of Theorem 11. Define *cyclic sum* and *difference* of $a,b \in \mathbb{Q}$ to be $a \oplus b = \{a+b\}$, resp., $a \ominus b := \{a-b\}$. For a set of clocks X, let $X_{x_0} = X \cup \{x_0\}$ be its extension with an extra clock $x_0 \notin X$ which is never reset, and let $\hat{X}_{x_0} = \{\hat{x} \mid x \in X_{x_0}\}$ be a corresponding set of registers. The special register $\hat{x}_0$ stores the (fractional part of the) current timestamp, and register $\hat{x}$ stores the (fractional part of the) timestamp of the last reset of x. In this way we can recover the fractional value of x as the cyclic difference $\{x\} = \hat{x}_0 \ominus \hat{x}$. Let (cf. Fig. 1(b))

$$\varphi_\ominus(\overline{x},\overline{\hat{x}}) \equiv \bigwedge_{x\in X} \{x\} = \hat{x}_0 \ominus \hat{x}. \tag{9}$$

Resetting clocks in $Y \subseteq X$ is simulated by $\varphi_{\mathsf{reset}(Y)} \equiv \hat{x}'_0 = \hat{x}_0 \wedge \bigwedge_{x\in Y} \hat{x}' = \hat{x}_0 \wedge \bigwedge_{x\in X\setminus Y} \hat{x}' = \hat{x}$ and time elapse by $\varphi_{\mathsf{elapse}} \equiv \bigwedge_{x\in X} \hat{x}' = \hat{x}$. The equality $\hat{x}'_0 = \hat{x}_0$ in $\varphi_{\mathsf{reset}(Y)}$ says that time does not elapse, and the absence of constraints on $\hat{x}_0, \hat{x}'_0$ in $\varphi_{\mathsf{elapse}}$ allows for an arbitrary elapse of time. A clock constraint φ is converted into a K-constraint $\hat{\varphi}$ by replacing $\{x\} = 0$ with $\hat{x} = \hat{x}_0$ and $\{x\} \le \{y\}$ by $K_\le(\hat{y},\hat{x},\hat{x}_0)$, for $x,y \in X \cup Z$. For a TPDA $\mathcal{P} = \langle \Sigma,\Gamma,L,X,Z,\Delta\rangle$, we define the following RPDA $\mathcal{Q} = \left\langle \Sigma,\Gamma,L,\hat{X}_{x_0},\hat{Z},\hat{\Delta}\right\rangle$. The input rules are preserved. A reset rule $\langle \ell,\mathsf{reset}(Y),r\rangle \in \Delta$, is simulated by $\langle \ell,\varphi_{\mathsf{reset}(Y)},r\rangle \in \hat{\Delta}$, a time elapse rule $\langle \ell,\mathsf{elapse},r\rangle \in \Delta$ is simulated by $\langle \ell,\varphi_{\mathsf{elapse}},r\rangle \in \hat{\Delta}$, a push rule $\langle \ell,\mathsf{push}(\gamma:\varphi),r\rangle \in \Delta$ is simulated by $\langle \ell,\mathsf{push}(\gamma:\hat{\varphi}),r\rangle \in \hat{\Delta}$, and similarly for pop rules. By Corollary 15, let $\varphi_{\ell r}(\overline{\hat{x}},\overline{f},\overline{\hat{x}'})$ express the reachability relation of $\mathcal{Q}$, and define $\xi^{\mathbb{I}}_o(\overline{x},\overline{x}') \equiv \exists \overline{\hat{x}},\overline{\hat{x}'} \cdot \varphi^{\mathbb{I}}_o(\overline{\hat{x}},\overline{\hat{x}'}) \wedge \varphi_\ominus(\overline{x},\overline{\hat{x}}) \wedge \varphi_\ominus(\overline{x}',\overline{\hat{x}'})$. The reachability relation of $\mathcal{P}$ is recovered as

$$\psi_{\ell r}(\overline{x},\overline{f},\overline{x}') \equiv \bigvee \{\varphi^{\mathbb{Z}}_{\ell ro}(\overline{f}) \wedge \xi^{\mathbb{I}}_o(\overline{x},\overline{x}') | o \in \mathrm{Orb}(\mathbb{I}^{X\times X'})\}. \tag{10}$$

Intuitively, we guess the value for registers $\overline{\hat{x}},\overline{\hat{x}'}$ and we check that they correctly describe the fractional values of global clocks as prescribed by $\varphi_\ominus$. We now remove the quantifiers from $\xi^{\mathbb{I}}_o$ to uncover the structure of fractional value comparisons. Introduce a new variable $\delta = \hat{x}_0 \ominus \hat{x}'_0$, and perform the following substitutions in $\varphi^{\mathbb{I}}_o$ (c.f. the definition of $\varphi_\ominus$ in (9)): $\hat{x} \mapsto \hat{x}_0 \ominus \{x\}$, $\hat{x}' \mapsto (\hat{x}_0 \ominus \delta) \ominus \{x'\}$, and $\hat{x}'_0 \mapsto \hat{x}_0 \ominus \delta$. By writing $(\hat{x}_0 \ominus \delta) \ominus \{x'\}$ as $\hat{x}_0 \ominus (\delta \oplus \{x'\})$, we have only atomic constraints of the forms $K(\hat{x}_0 \ominus u, \hat{x}_0 \ominus v, \hat{x}_0 \ominus t)$ and $\hat{x}_0 \ominus u = \hat{x}_0 \ominus v$, where terms u,v,t are of one of the forms 0, $\{x\}$, $\delta \oplus \{x'\}$, δ. These constraints are equivalent, respectively, to $K(t,v,u)$ and $u = v$. By expanding the definition of K (cf. (1)), we obtain only constraints of the form $u \precsim v$ with $\precsim \in \{<,\le\}$. Since δ appears

at most once on either side, it can either be eliminated if it appears on both u, v, or otherwise exactly one of u, v is of the form δ or $\delta \oplus \{x'\}$, and the other of the form 0 or $\{x\}$. By moving $\{x'\}$ on the other side of the inequality in constraints containing $\delta \oplus \{x'\}$, $\xi_o^{\mathbb{I}}$ is equivalent to $\bigwedge_i s_i \precsim t_i \ \wedge \ \exists 0 \leq \delta < 1 \cdot \bigwedge_j u_j \precsim \delta \wedge \bigwedge_k \delta \precsim v_k$, where the terms s_i, t_i, u_j, v_k's are of the form 0, $\{x\}$, or $\{x\} \ominus \{y'\}$. We can now eliminate the quantification on δ and get a constraint of the form $\bigwedge_h s_h \precsim t_h$. Finally, by expanding $b \ominus a$ as $b - a + 1$ if $b < a$ and $b - a$ otherwise (since $a, b \in \mathbb{I}$) we have $\xi_o^{\mathbb{I}}(\overline{x}, \overline{x}') \ \equiv \ \bigwedge_h s'_h \precsim t'_h$, where the s'_h, t'_h's are of one of the forms: 0, $\{x\}$, $\{x\} - \{y'\}$, or $\{x\} - \{y'\} + 1$. ◀

References

1. P. A. Abdulla, M. F. Atig, and J. Stenman. Dense-timed pushdown automata. In *Proc. LICS'12*, pages 35–44. IEEE, 2012. doi:10.1109/LICS.2012.15.
2. Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126:183–235, 1994.
3. M. Benerecetti, S. Minopoli, and A. Peron. Analysis of timed recursive state machines. In *Proc. TIME'10*, pages 61–68. IEEE, sept. 2010. doi:10.1109/TIME.2010.10.
4. Mikołaj Bojańczyk. Slightly infinite sets. URL: https://www.mimuw.edu.pl/~bojan/paper/atom-book.
5. Ahmed Bouajjani, Rachid Echahed, and Riadh Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In *Proc. Hybrid Systems '94*, volume 999 of *LNCS*, pages 64–85. Springer, 1995.
6. Lorenzo Clemente. Decidability of timed communicating automata. *ArXiv e-prints*, 04 2018. arXiv:1804.07815.
7. Lorenzo Clemente and Slawomir Lasota. Reachability analysis of first-order definable pushdown systems. In *Proc. of CSL'15*, volume 41 of *LIPIcs*, pages 244–259. Dagstuhl, 2015.
8. Lorenzo Clemente and Slawomir Lasota. Timed pushdown automata revisited. In *Proc. LICS'15*, pages 738–749. IEEE, July 2015.
9. Lorenzo Clemente, Sławomir Lasota, Ranko Lazić, and Filip Mazowiecki. Timed pushdown automata and branching vector addition systems. In *Proc. of LICS'17*, 2017.
10. Lorenzo Clemente and Sławomir Lasota. Binary reachability of timed pushdown automata via quantifier elimination and cyclic order atoms. *ArXiv e-prints*, 04 2018. arXiv:1804.10772.
11. Hubert Comon and Yan Jurski. Timed automata and the theory of real numbers. In *Proc. of CONCUR'99*, CONCUR '99, pages 242–257, London, UK, UK, 1999. Springer-Verlag.
12. Zhe Dang. Pushdown timed automata: a binary reachability characterization and safety verification. *Theor. Comput. Sci.*, 302(1–3):93–121, 2003. doi:10.1016/S0304-3975(02)00743-0.
13. C. Dima. Computing reachability relations in timed automata. In *In Proc. of LICS'02*, pages 177–186, 2002.
14. Jeanne Ferrante and Charles Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM Journal on Computing*, 4(1):69–76, 1975.
15. Pavel Krčál and Radek Pelánek. On sampled semantics of timed systems. In Sundar Sarukkai and Sandeep Sen, editors, *In Proc. of FSTTCS'05*, volume 3821 of *LNCS*, pages 310–321. Springer, 2005.
16. Dugald Macpherson. A survey of homogeneous structures. *Discrete Mathematics*, 311(15):1599–1634, 2011.
17. Mojżesz Presburger. Über der vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchen die addition als einzige operation hervortritt. *Comptes Rendus Premier Congrès des Mathématicienes des Pays Slaves*, 395:92–101, 1930.

18 K. Quaas, M. Shirmohammadi, and J. Worrell. Revisiting reachability in timed automata. In *Proc. of LICS'17*, pages 1–12, June 2017. doi:10.1109/LICS.2017.8005098.

19 Eduardo D. Sontag. Real addition and the polynomial hierarchy. *Information Processing Letters*, 20(3):115–120, 1985.

20 Ashutosh Trivedi and Dominik Wojtczak. Recursive timed automata. In *Proc. ATVA'10*, volume 6252 of *LNCS*, pages 306–324. Springer, 2010.

21 Yuya Uezato and Yasuhiko Minamide. Synchronized recursive timed automata. In *Proc. of LPAR'15*, 2015.

22 Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. On the complexity of equational Horn clauses. In *Proc. CADE-20, 2005*, pages 337–352, 2005. doi:10.1007/11532231_25.

23 Volker Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5(1):3–27, 1988.

Unboundedness Problems for Languages of Vector Addition Systems

Wojciech Czerwiński[1]
University of Warsaw, Poland
https://orcid.org/0000-0002-6169-868X

Piotr Hofman[2]
University of Warsaw, Poland
https://orcid.org/0000-0001-9866-3723

Georg Zetzsche[3]
IRIF (Université Paris-Diderot & CNRS), France
https://orcid.org/0000-0002-6421-4388

Abstract

A vector addition system (VAS) with an initial and a final marking and transition labels induces a language. In part because the reachability problem in VAS remains far from being well-understood, it is difficult to devise decision procedures for such languages. This is especially true for checking properties that state the existence of infinitely many words of a particular shape. Informally, we call these *unboundedness properties*.

We present a simple set of axioms for predicates that can express unboundedness properties. Our main result is that such a predicate is decidable for VAS languages as soon as it is decidable for regular languages. Among other results, this allows us to show decidability of (i) separability by bounded regular languages, (ii) unboundedness of occurring factors from a language K with mild conditions on K, and (iii) universality of the set of factors.

2012 ACM Subject Classification Theory of computation → Concurrency, Theory of computation → Formal languages and automata theory

Keywords and phrases vector addition systems, decision problems, unboundedness, separability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.119

Acknowledgements We are indebted to Mohamed Faouzi Atig for suggesting to study separability by bounded languages, which was the starting point for this work. We are also grateful to Sławomir Lasota and Sylvain Schmitz for important discussions. This collaboration started at the Gregynog 71717 workshop organized by Ranko Lazić and Patrick Totzke.

1 Introduction

Vector addition systems (VAS) and, essentially equivalent, Petri nets are among the most widely used models of concurrent systems. Although they are used extensively in practice, there are still fundamental questions that are far from being well understood.

[1] Partially supported by the Polish National Science Centre grant 2016/21/D/ST6/01376
[2] Partially supported by the Polish National Science Centre grant 2016/21/B/ST6/01505
[3] Supported by a fellowship of the Fondation Sciences Mathématiques de Paris and partially funded by the DeLTA project (ANR-16-CE40-0007). Part of the research was conducted when the author was supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD) and by Labex DigiCosme, Univ. Paris-Saclay, project VERICONISS.

This is reflected in what we know about decidability questions regarding the most expressive class of languages associated to VAS: The languages of (arbitrarily) labeled VAS with a given initial and final configuration, which we call *VAS languages*. In the 1970s, this class has been characterized in terms of closure properties and Dyck languages by Greibach [27] and Jantzen [36]. Almost all decidability results about these languages use a combination of these closure properties and the decidability of the reachability problem for VAS [43] (or for Reinhardt's extension [47], such as in [1, 51]). Of course, this method is confined to procedures that somehow reduce to the existence of one or finitely many runs of VAS.

There are two notable exceptions to this and they both rely on an extensions of decision procedures for reachability in VAS. The first is Hauschildt and Jantzen's result [33] from 1994 that finiteness of VAS languages is decidable, which employs Hauschildt's algorithm to decide semilinearity of reachability sets [32]. The second is the much more recent result of Habermehl, Meyer, and Wimmel from 2010 [28]. It shows computability of downward closures for VAS languages (which significantly generalizes decidability of finiteness) and involves a careful inspection of marked graph-transition sequences (MGTS) of Lambert's algorithm. (As observed in [15, 52], computability of downward closures can also be derived from [6].) This sparsity of results is due to the fact that the algorithms for the reachability problem are still quite unwieldy and have been digested by few researchers.

In particular, it currently seems difficult to decide whether there exist infinitely many words of some shape in a given language—unless the problem reduces to computing downward closures. Informally, we call problems of this type *unboundedness problems*. Such problems are important for two reasons. The first concerns *separability problems*, which have attracted attention in recent years [5, 11, 26, 44, 45]. Here, instead of deciding whether two languages are disjoint, we are looking for a (typically finite-state) certificate for disjointness, namely a set that includes one language and is disjoint from the other. For general topological reasons, inseparability is usually witnessed by a common pattern, whose presence in a language is an unboundedness property. The second reason is that unboundedness problems tend to be *decidable where exact queries are not*. This phenomenon also occurs in the theory of regular cost functions [12]. Moreover, as it turns out in this work, this is true for VAS languages as well.

Contribution. We present a simple notion of an *unboundedness predicate* on languages and show that such predicates are decidable for VAS languages as soon as they are decidable for regular languages. On the one hand, this provides an easy and general way to obtain new decidability results for VAS languages without the need to understand the details of the KLMST decomposition. On the other hand, we apply this framework to prove:

(i) Boundedness in the sense of Ginsburg and Spanier [24] is decidable. A language $L \subseteq \Sigma^*$ is *bounded* if $L \subseteq w_1^* \cdots w_n^*$ for some $w_1, \ldots, w_n \in \Sigma^*$. Moreover, it is decidable whether two given VAS languages are separable by a bounded regular language.

(ii) Computability of downward closures can be recovered as well.

(iii) Suppose that $K \subseteq \Sigma^*$ is chosen so that it is decidable whether K intersects a given regular language. Then, it is decidable for a given VAS language L whether L contains words with arbitrarily many factors from K. Moreover, in case the number of factor occurrences in L is bounded, we can even compute an upper bound.

(iv) Under the same assumptions as above on $K \subseteq \Sigma^*$, one can decide if every word from K^* appears as a factor of a given VAS language $L \subseteq \Sigma^*$. In particular, it is decidable whether L contains every word from Σ^* as a factor.

It should be stressed that results (iii) and (iv) came deeply unexpected to the authors for two reasons. First, the assumptions are already satisfied when K is induced by a system model as powerful as well-structured transition systems (WSTS) or higher-order recursion schemes. In these cases, it is in general *undecidable* whether a given VAS language contains a factor from K *at least once*, because intersection emptiness easily reduces to this problem (see the remarks after Theorem 4.8). We therefore believe that these results might lead to new approaches to verifying systems with concurrency and (higher-order) recursion, where the latter undecidability (or the unknown status in the case of simple recursion [40]) is usually a barrier for decision procedures. Secondly, the problems in (iii) and (iv) are undecidable as soon as L is just slightly beyond the realm of VAS: Already for one-counter languages L, both (iii) and (iv) become undecidable. Thus, compared to other infinite-state systems, VAS languages turn out to be extraordinarily amenable to unboundedness problems.

Related work. Other authors have investigated general notions of unboundedness properties for VAS [2, 4, 16, 50], usually with the goal of obtaining EXPSPACE upper bounds. However, those properties a priori concern the state space itself. While they can sometimes be used to reason about languages [4, 16], this has been confined to coverability languages, which are significantly less expressive than the reachability languages studied here. Specifically, every problem we consider here is hard for the reachability problem (see Remark 3.2).

An early attempt was Yen's work [50], which claimed an EXPSPACE upper bound for a powerful logic concerning paths in VAS. Unfortunately, a serious flaw in the latter was discovered by Atig and Habermehl [2], who presented a corrected proof for a restricted version of Yen's logic. Demri [16] then introduced a notion of *generalized unboundedness properties*, which covers more properties from Yen's logic and proved an EXPSPACE procedure to check them. Examples include reversal-boundedness, place boundedness, and regularity of firing sequences of unlabeled VAS. Finally, Blockelet and Schmitz [4] introduce an extension of computation tree logic (CTL) that can express "coverability-like properties" of VAS. The authors prove an EXPSPACE upper bound for model checking this logic on VAS.

2 Preliminaries

Let Σ be a finite alphabet. For $w \in \Sigma^*$, we denote its length by $|w|$. The i-th letter of w, for $i \in [1, |w|]$ is denoted $w[i]$. Moreover, we write $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$. A *(d-dimensional) vector addition system* (VAS) V consists of finite set of *transitions* $T \subseteq \mathbb{Z}^d$, *source* and *target* vectors $s, t \in \mathbb{N}^d$ and a *labeling* $h\colon T \to \Sigma_\varepsilon$, whose extension to a morphism $T^* \to \Sigma^*$ is also denoted h. Vectors $v \in \mathbb{N}^d$ are also called *configurations*. A transition $t \in T$ can be *fired* in a configuration $v \in \mathbb{N}^d$ if $v + t \in \mathbb{N}^d$. Then, the result of firing t is the configuration $v + t$ and we write $v \xrightarrow{h(t)} v'$ for $v' = v + t$. For $w \in \Sigma^*$, we write $v \xrightarrow{w} v'$ if there exist $v_1, \ldots, v_k \in \mathbb{N}^d$ such that $v = v_0 \xrightarrow{x_1} v_1 \xrightarrow{x_2} \ldots \xrightarrow{x_k} v_k \xrightarrow{x_{k+1}} v_{k+1} = v'$, where $w = x_1 \cdots x_{k+1}$ for some $x_1, \ldots, x_{k+1} \in \Sigma_\varepsilon$. The *language* of V, denoted $L(V)$, is the set of all labels of runs from source to target, i.e. $L(V) = \{w \in \Sigma^* \mid s \xrightarrow{w} t\}$. The languages of the form $L(V)$ for VAS V are called *VAS languages*. A word $u = a_1 \cdots a_n$ with $a_i \in \Sigma$ is a *subword* of a word $v \in \Sigma^*$ if $v \in \Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$, which is denoted $u \preceq v$. For a language $L \subseteq \Sigma^*$ its *downward closure* is the language $L{\downarrow} = \{u \in \Sigma^* \mid \exists v \in L\colon u \preceq v\}$. It is known that $L{\downarrow}$ is regular for every $L \subseteq \Sigma^*$ [34, 31]. A *language class* is a collection of languages, together with some way of finitely describing these languages (such as by grammars, automata, etc.). If $\mathcal{C}$ is a language class so that given a description of a language L from $\mathcal{C}$, we can compute an automaton for $L{\downarrow}$, we say that *downward closures are computable* for $\mathcal{C}$.

A *full trio* is a language class that is effectively closed under rational transductions [3], which are relations defined by nondeterministic two-tape automata. Equivalently, a full trio is a class that is effectively closed under morphisms, inverse morphisms, and regular intersection [3]. Full trios are abundant among infinite-state models: If a nondeterministic machine model involves a finite-state control, the resulting language class is a full trio. Examples include *VAS languages* [36], *coverability languages of WSTS* [23], *one-counter languages* (which are accepted by one-counter automata *with zero tests*) [35], and languages of *higher-order pushdown automata* [42] and *higher-order recursion schemes* [30]. The context-sensitive languages do not constitute a full trio, as they are not closed under erasing morphisms.

3 Main result

Here, we introduce our notion of unboundedness predicates and present our main result.

For didactic purposes, we begin our exposition of unboundedness predicates with a simplified (but already useful) version. An important aspect of the definition is that technically, an unboundedness predicates is not a property of the language $L \subseteq \Sigma^*$ we want to analyze, but of the set of its factors. In other words, we have a unary predicate $\mathfrak{p}$ on languages and we want to decide whether $\mathfrak{p}(F(L))$, where $F(L) = \{w \in \Sigma^* \mid L \cap \Sigma^* w \Sigma^* \neq \emptyset\}$ is the set of factors of L. For the definition, it is helpful to keep in mind the simplest example of an unboundedness predicate, the *infinity predicate* $\mathfrak{p}_{\mathsf{inf}}$, where $\mathfrak{p}_{\mathsf{inf}}(K)$ if and only if K is infinite. Then, $\mathfrak{p}_{\mathsf{inf}}(F(L))$ if and only if L is infinite. A unary predicate $\mathfrak{p}$ on languages over Σ^* is called *1-dimensional unboundedness predicate* if for every $K, L \subseteq \Sigma^*$, we have:

(i*) if $\mathfrak{p}(K)$ and $K \subseteq L$, then $\mathfrak{p}(L)$.
(ii*) if $\mathfrak{p}(K \cup L)$, then either $\mathfrak{p}(L)$ or $\mathfrak{p}(K)$.
(iii*) if $\mathfrak{p}(F(KL))$, then either $\mathfrak{p}(F(K))$ or $\mathfrak{p}(F(L))$.

Part of our result will be that for such predicates, if we can decide whether $\mathfrak{p}(F(R))$ for regular languages R, we can decide whether $\mathfrak{p}(F(L))$ for VAS languages L. Before we come to that, we want to generalize a bit. There are predicates we want to decide that fail to satisfy axiom (iii*), such as the one stating $a^*b^* \subseteq L{\downarrow}$ for $L \subseteq \Sigma^*$: It is satisfied for a^*b^*, but neither for a^* nor for b^*. (Deciding such predicates is useful for computing downward closures [52] and separability by piecewise testable languages [15]) To capture such predicates, which intuitively ask for several quantities being unbounded simultaneously, we present a more general set of axioms. Here, the idea is to formulate predicates over simultaneously occurring factors. For a language $L \subseteq \Sigma^*$ and $n \in \mathbb{N}$, let

$$F_n(L) = \{(w_1, \ldots, w_n) \in (\Sigma^*)^n \mid \Sigma^* w_1 \Sigma^* \cdots w_n \Sigma^* \cap L \neq \emptyset\}.$$

We will speak of *n-dimensional* predicates, i.e., predicates $\mathfrak{p}$ on subsets of $(\Sigma^*)^n$, and we want to decide whether $\mathfrak{p}(F_n(L))$ for a given language L. The following are axioms referring to all subsets $S, T \subseteq (\Sigma^*)^n$, languages $L_i \subseteq \Sigma^*$, and all $k \in \mathbb{N}$. We call $\mathfrak{p}$ an *(n-dimensional) unboundedness predicate* if

(i) if $\mathfrak{p}(S)$ and $S \subseteq T$, then $\mathfrak{p}(T)$.
(ii) if $\mathfrak{p}(S \cup T)$, then $\mathfrak{p}(S)$ or $\mathfrak{p}(T)$.
(iii) if $\mathfrak{p}(F_n(L_1 \cdots L_k))$, then $n = n_1 + \cdots + n_k$ such that $\mathfrak{p}(F_{n_1}(L_1) \times \cdots \times F_{n_k}(L_k))$.

Intuitively, the last axiom says that if a concatenation satisfies the predicate, then this is already witnessed by factors in at most n participants of the concatenation. Note that for

$n = 1$, the axioms coincide with the simplified axioms (i*) to (iii*) above. An n-dimensional unboundedness predicate $\mathfrak{p}$ is *decidable for a language class* $\mathcal{C}$ if, given a language L from $\mathcal{C}$, it is decidable whether $\mathfrak{p}(F_n(L))$. The following is our main result.

▶ **Theorem 3.1.** *Given a VAS language $L \subseteq \Sigma^*$, one can compute a regular $R \subseteq \Sigma^*$ such that $L \subseteq R$ and for every n-dim. unboundedness predicate $\mathfrak{p}$, we have $\mathfrak{p}(F_n(L))$ iff $\mathfrak{p}(F_n(R))$.*

Note that this implies that decidability of $\mathfrak{p}$ for regular languages implies decidability of $\mathfrak{p}$ for VAS languages for any n-dim. unboundedness predicate $\mathfrak{p}$. In addition, when our unboundedness predicate expresses that a certain quantity is unbounded, then in the bounded case, Theorem 3.1 sometimes allows us to compute an upper bound (see, e.g. Theorem 4.8).

▶ Remark 3.2. Let us comment on the complexity of deciding whether $\mathfrak{p}(F_n(L))$ for a VAS language L. Call $\mathfrak{p}$ *non-trivial* if there is at least one $S \subseteq (\Sigma^*)^n$ that satisfies $\mathfrak{p}$ and least one $S' \subseteq (\Sigma^*)^n$ for which $\mathfrak{p}$ is not satisfied. Then, deciding whether $\mathfrak{p}(F_n(L))$ is at least as hard as the reachability problem. Indeed, in this case axiom (i) implies that $F_n(\Sigma^*) = (\Sigma^*)^n$ satisfies $\mathfrak{p}$, but $F_n(\emptyset) = \emptyset$ does not. Given a VAS V and two vectors μ_1 and μ_2, it is easy to construct a VAS V' so that $L(V') = \Sigma^*$ if V can reach μ_2 from μ_1 and $L(V') = \emptyset$ otherwise.

4 Applications

Bounded languages. Our first application concerns bounded languages. A language $L \subseteq \Sigma^*$ is *bounded* if there exist words $w_1, \ldots, w_n \in \Sigma^*$ such that $L \subseteq w_1^* \cdots w_n^*$. This notion was introduced by Ginsburg and Spanier [24]. Since a bounded language as above can be characterized by the set of vectors $(x_1, \ldots, x_n) \in \mathbb{N}^n$ for which $w_1^{x_1} \cdots w_n^{x_n} \in L$, bounded languages are quite amenable to analysis. This has led to a number of applications to concurrent recursive programs [18, 19, 20, 21, 41], but also counter systems [17] and WSTS [9].

Boundedness has been shown decidable for *context-free languages* by Ginsburg and Spanier [24] (PTIME-completeness by Gawrychowski et al. [22]) and hence also for *regular languages* (NL-completeness also in [22]), for *equal matrix languages* by Siromoney [49], and for trace languages of *complete deterministic well-structured transition systems* by Chambart et al. [9]. The latter implies that boundedness is decidable for coverability languages of deterministic vector addition systems, in which case EXPSPACE-completeness was shown by Chambart et al. [9] (the upper bound had been established by Blockelet and Schmitz [4]).

We use Theorem 3.1 to show the following.

▶ **Theorem 4.1.** *Given a VAS, it is decidable whether its language is bounded.*

The rest of this section is devoted to the proof of Theorem 4.1. Let $\mathfrak{p}_{\mathsf{notb}}$ be the 1-dimensional predicate that holds for a language $K \subseteq \Sigma^*$ if and only if K it is not bounded. We plan to apply Theorem 3.1 to $\mathfrak{p}_{\mathsf{notb}}$, but it allows us to decide only whether $\mathfrak{p}_{\mathsf{notb}}(F(L))$ for a given VAS language L. Thus we need the following fact, which we prove in a moment.

▶ **Fact 4.2.** *A language $L \subseteq \Sigma^*$ is bounded if and only if $F(L)$ is bounded.*

We also need to show that $\mathfrak{p}_{\mathsf{notb}}$ is indeed an unboundedness predicate, meaning that it satisfies axioms (i*) to (iii*). By definition of boundedness, $\mathfrak{p}_{\mathsf{notb}}$ clearly fulfills axiom (i*). Axioms (ii*) and (iii*) are implied by Fact 4.2 and the following.

▶ **Fact 4.3.** *If K and L are bounded then both $K \cup L$ and KL are bounded as well.*

Let us prove Facts 4.2 and 4.3 and begin with Fact 4.3. If K and L are bounded, then $K \subseteq w_1^* \cdots w_n^*$ and $L \subseteq w_{n+1}^* \cdots w_m^*$ for some $w_1, \ldots, w_m \in \Sigma^*$. Then we have $K \cup L, KL \subseteq w_1^* \cdots w_m^*$, which shows Fact 4.3. In order to show Fact 4.2, observe first that for each individual word $w \in \Sigma^*$, the language $F(w)$ is bounded because it is finite. Thus, if $L \subseteq w_1^* \cdots w_n^*$, then $F(L)$ is included in $F(w_1)w_1^*F(w_1) \cdots F(w_n)w_n^*F(w_n)$, which is bounded as a concatenation of bounded languages by Fact 4.3. Thus, $F(L)$ is bounded as well. Conversely, L inherits boundedness from its superset $F(L)$.

To conclude Theorem 4.1, we need to show that given regular language $R \subseteq \Sigma^*$, it is decidable whether $\mathfrak{p}_{\mathsf{notb}}(F(R))$. By Fact 4.2, this amounts to checking whether R is bounded. This is decidable even for context-free languages [24] (and in NL for regular ones [22]).

Separability. We can also use our results to decide whether two VAS languages are separable by a bounded regular language. Very generally, if $\mathcal{S}$ is a class of sets, we say that a set K is *separable from* a set L *by* a set from $\mathcal{S}$ if there is a set S in $\mathcal{S}$ so that $K \subseteq S$ and $L \cap S = \emptyset$.

The separability problem was recently investigated for VAS languages and several subclasses. In [15] it is shown that separability of VAS languages by piecewise testable languages (a subclass of regular languages) is decidable. Decidability of separability of VAS languages by regular languages is still open, but it is known for several subclasses thereof [10, 11, 13]. In [14] it is shown that any two disjoint VAS coverability languages are separable by a regular language. Here, using Theorem 4.1 we are able to show the following.

▶ **Theorem 4.4.** *Given two VAS languages K and L, it is decidable whether K is separable from L by a bounded regular language.*

Clearly, in order for that to hold, K has to be bounded, which we can decide. Moreover, by enumerating expressions $w_1^* \cdots w_n^*$, we can find one with $K \subseteq w_1^* \cdots w_n^*$. Since the bounded regular languages (BRL) are closed under intersection (recall that a subset of a bounded language is again bounded), K and L are separable by a BRL if and only if $L_0 = K$ and $L_1 = L \cap w_1^* \cdots w_n^*$ are separable by a BRL. Since now both input languages are included in $w_1^* \cdots w_n^*$, we can reformulate the problem into one over vector sets.

▶ **Lemma 4.5.** *Let $L_0, L_1 \subseteq w_1^* \cdots w_n^*$ and $U_i = \{(x_1, \ldots, x_n) \in \mathbb{N}^n \mid w_1^{x_1} \cdots w_n^{x_n} \in L_i\}$ for $i \in \{0, 1\}$. Then, L_0 is separable from L_1 by a BRL if and only if U_0 is separable from U_1 by a recognizable subset of $\mathbb{N}^n$.*

Recall that a subset $S \subseteq \mathbb{N}^n$ is *recognizable* if there is a morphism $\varphi\colon \mathbb{N}^n \to F$ into a finite monoid F with $S = \varphi^{-1}(\varphi(S))$. Lemma 4.5 is a straightforward application of Ginsburg and Spanier's characterization of BRL [25].

Since in our case, L_0 and L_1 are VAS languages, a standard construction shows that U_0 and U_1 are (effectively computable) sections of VAS reachability sets. Here, sections are defined as follows. For a subset $I \subseteq [1, n]$, let $\pi_I\colon \mathbb{N}^n \to \mathbb{N}^{|I|}$ be the projection onto the coordinates in I. Then, every set of the form $\pi_{[1,n]\setminus I}(S \cap \pi_I^{-1}(x))$ for some $I \subseteq [1, n]$ and $x \in \mathbb{N}^{|I|}$ is called a *section* of $S \subseteq \mathbb{N}^n$. Thus, the following result by Clemente et al. [11] allows us to decide separability by BRL.

▶ **Theorem 4.6** ([11])**.** *Given two sections $S_0, S_1 \subseteq \mathbb{N}^n$ of reachability sets of VAS, it is decidable whether S_0 is separable from S_1 by a recognizable subset of $\mathbb{N}^n$.*

Downward closures and simultaneous unboundedness. We now illustrate how to compute downward closures using our results. First of all, computability of downward closures for VAS languages follows directly from Theorem 3.1 because it implies $R{\downarrow} = L{\downarrow}$: For each word

$w = a_1 \cdots a_n$ with $a_1, \ldots, a_n \in \Sigma$, consider the n-dimensional predicate $\mathfrak{p}_w$ which is satisfied for $S \subseteq (\Sigma^*)^n$ iff $(a_1, \ldots, a_n) \in S$. Then $\mathfrak{p}_w(F_n(L))$ if and only if $w \in L{\downarrow}$. It is easy to check that this is an unboundedness predicate. Hence, $R{\downarrow} = L{\downarrow}$.

However, in order to illustrate how to apply unboundedness predicates, we present an alternative approach. In [52], it was shown that if a language class $\mathcal{C}$ is closed under rational transductions (which is the case for VAS languages), then downward closures are computable for $\mathcal{C}$ if and only if, given a language L from $\mathcal{C}$ and letters $a_1, \ldots, a_n$, it is decidable whether $a_1^* \cdots a_n^* \subseteq L{\downarrow}$. Let us show how to decide the latter using unboundedness predicates.

For this, we use an n-dimensional predicate. For a subset $S \subseteq (\Sigma^*)^n$, let $S{\downarrow}$ be the set of all tuples $(u_1, \ldots, u_n) \in (\Sigma^*)^n$ such that there is some $(v_1, \ldots, v_n) \in S$ with $u_i \preceq v_i$ for $i \in [1, n]$. Our predicate $\mathfrak{p}_{\mathsf{sup}}$ is satisfied for $S \subseteq (\Sigma^*)^n$ if and only if $a_1^* \times \cdots \times a_n^* \subseteq S$. Then clearly $\mathfrak{p}_{\mathsf{sup}}(F_n(L))$ if and only if $a_1^* \cdots a_n^* \subseteq L{\downarrow}$. It is easy to check that $\mathfrak{p}_{\mathsf{sup}}$ fulfills axiom (i) and axiom (ii). For the latter, note that $a_1^* \times \cdots \times a_n^* \subseteq (S_1 \cup S_2){\downarrow}$ implies that for some $j \in \{1, 2\}$, there are infinitely many $\ell \in \mathbb{N}$, with $(a_1^\ell, \ldots, a_n^\ell) \in S_j$ and hence $a_1^* \times \cdots \times a_n^* \subseteq S_j{\downarrow}$. For axiom (iii), we need a simple combinatorial argument:

▶ **Lemma 4.7.** *If $a_1^* \times \cdots \times a_n^* \subseteq F_n(L_1 \cdots L_k){\downarrow}$, then $n = n_1 + \cdots + n_k$ with $a_1^* \times \cdots \times a_n^* \subseteq (F_{n_1}(L_1) \times \cdots \times F_{n_k}(L_k)){\downarrow}$.*

It remains to show that for a regular language R, it is decidable whether $a_1^* \cdots a_n^* \subseteq R{\downarrow}$. Since it is easy to construct an automaton for $R{\downarrow}$, this amounts to a simple inclusion check.

Non-overlapping factors. Our next example shows that under very mild assumptions on a language K, one can decide whether the words in a VAS language L contain arbitrarily many factors from K. For $w \in \Sigma^*$ and $K \subseteq \Sigma^+$, let $|w|_K$ be the largest number m such that there are $w_1, \ldots, w_m \in K$ with $(w_1, \ldots, w_m) \in F_m(w)$. Note that since $\varepsilon \notin K$, there is always a maximal such m. Consider the function $f_K \colon \Sigma^* \to \mathbb{N}$, $w \mapsto |w|_K$. A function $f \colon \Sigma^* \to \mathbb{N}$ is *unbounded* on $L \subseteq \Sigma^*$ if for every $k \in \mathbb{N}$, we have $f(w) \geq k$ for some $w \in L$.

▶ **Theorem 4.8.** *If $\mathcal{C}$ is a full trio with decidable emptiness problem, then given a VAS language L and a language $K \subseteq \Sigma^+$ from $\mathcal{C}$, it is decidable whether f_K is unbounded on L. If f_K is bounded on L, we can compute an upper bound.*

Theorem 4.8 is quite unexpected because very slight variations lead to undecidability. If we ask whether f_K is non-zero on a given VAS language (as opposed to unbounded), then this is in general undecidable. Indeed, suppose $\mathcal{C}$ is a full trio for which intersection with VAS languages is undecidable (such as languages of lossy channel systems[4] or higher-order pushdown languages [29, 52]). Then given a language $K \subseteq \Sigma^*$ from $\mathcal{C}$, a VAS language L and some $c \notin \Sigma$, the function f_{cKc} is non-zero on cLc if and only if $K \cap L \neq \emptyset$.

Furthermore, the same problem becomes undecidable in general if instead of VAS languages, we want to decide the problem for a language class as simple as one-counter languages (OCL). Indeed, suppose $\mathcal{C}$ is a full trio for which intersection with OCL is undecidable (such as the class of OCL). For a given $K \subseteq \Sigma^*$ from $\mathcal{C}$, an OCL $L \subseteq \Sigma^*$, and some $c \notin \Sigma$, the set $c(Lc)^*$ is effectively an OCL and f_{cKc} is unbounded on $c(Lc)^*$ if and only if $K \cap L \neq \emptyset$.

Let us prove Theorem 4.8. Fix a language $K \subseteq \Sigma^*$ from $\mathcal{C}$. Our predicate $\mathfrak{p}_{\mathsf{nof}}$ is one-dimensional and is satisfied on a set $L \subseteq \Sigma^*$ if and only if f_K is unbounded on L. Then clearly,

[4] It seems to be folklore that intersection between languages of lossy channel systems and languages of one-dimensional VAS is undecidable (the additional counter can be used to ensure that no letter is dropped). The only reference we could find is [46].

$\mathfrak{p}_{\mathsf{nof}}(F(L))$ if and only if f_K is unbounded on L. It is immediate that axioms (i*) and (ii*) are satisfied. Furthermore, axiom (iii*) follows by contraposition: If neither $\mathfrak{p}_{\mathsf{nof}}(F(L_0))$ nor $\mathfrak{p}_{\mathsf{nof}}(F(L_1))$, then there are $B_0, B_1 \in \mathbb{N}$ such that f_K is bounded by B_i on L_i for $i = 0, 1$. That implies that f_K is bounded by $B_0 + B_1 + 1$ on L_0L_1. This rules out $\mathfrak{p}_{\mathsf{nof}}(F(L_0L_1))$, which establishes axiom (iii*). The following uses standard arguments.

▶ **Lemma 4.9.** *Let $\mathcal{C}$ be a full trio with decidable emptiness problem. Given a language K from $\mathcal{C}$ and a regular language R, it is decidable whether f_K is unbounded on R. Moreover, if f_K is bounded on R, we can compute an upper bound.*

We can deduce Theorem 4.8 from Lemma 4.9 as follows. Using Theorem 3.1, we compute the language R. Then, f_K is unbounded on R iff it is unbounded on L. Moreover, an upper bound for f_K on R is also an upper bound for f_K on L because $L \subseteq R$.

Counting automata. To illustrate how these results can be used, we formulate an extension of Theorem 4.8 in terms of automata that can count. Let $\mathcal{C}$ be a full trio. Intuitively, a $\mathcal{C}$-counting automaton can read a word produced by a VAS and can use machines corresponding to $\mathcal{C}$ as oracles. Just like the intersection of two languages that describe threads in a concurrent system signals a safety violation [7, 8, 41], a successful oracle call would signal a particular undesirable event. In such a model, it would be undecidable whether any oracle call can be successful if, for example, $\mathcal{C}$ is the class of higher-order pushdown languages. However, we show that it *is* decidable whether such an automaton can make an unbounded number of successful oracle calls and if not, compute an upper bound. Hence, we can decide if the number of undesirable events is bounded and, if so, provide a bound.

A *$\mathcal{C}$-counting automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, C, q_0, E, Q_f)$, where Q is a finite set of *states*, Σ is its *input alphabet*, Γ is its *(oracle) tape alphabet*, C is a finite set of *counters*, $q_0 \in Q$ is its *initial state*, $Q_f \subseteq Q$ is its set of *final states*, and $E \subseteq Q \times \Sigma^* \times (\Omega \cup \{\varepsilon\}) \times Q$ is a finite set of *edges*, where Ω is a set of *operations* of the following form. First, we have an operation $\mathsf{push}(a)$ for each $a \in \Gamma$, which appends a to the oracle tape. Moreover, we have $\mathsf{check}(K, c)$ for each $K \subseteq \Gamma^*$ from $\mathcal{C}$ and each $c \in C$, which first checks whether the current tape content belongs to K and if so, increments the counter c. After the oracle query, it empties the oracle tape, regardless of whether the oracle answers positively or negatively.

A *configuration* of $\mathcal{A}$ is a triple (q, u, μ), where $q \in Q$ is the current state, $u \in \Gamma^*$ is the oracle tape content, and $\mu \in \mathbb{N}^C$ describes the counter values. For a label $x \in \Sigma \cup \{\varepsilon\}$, and configurations $(q, u, \mu), (q', u', \mu')$, we write $(q, u, \mu) \xrightarrow{x} (q', u', \mu')$ if (q', u', μ') results from (q, u, μ) as described above. In the general case $w \in \Sigma^*$, $(q, u, \mu) \xrightarrow{w} (q', u', \mu')$ has the obvious meaning. We extend the set of natural numbers $\mathbb{N}$ by setting $\overline{\mathbb{N}} = \mathbb{N} \cup \{\omega\}$, where ω is the first infinite ordinal number and represents the infinity. $\mathcal{A}$ defines a function $\Sigma^* \to \overline{\mathbb{N}}$ with $\mathcal{A}(w) = \sup \left\{ \inf_{c \in C} \mu(c) \mid \mu \in \mathbb{N}^C,\ (q_0, \varepsilon, 0) \xrightarrow{w} (q, u, \mu) \text{ for some } q \in Q_f,\ u \in \Gamma^* \right\}$, where 0 is the zero vector in $\mathbb{N}^C$. Hence, $\mathcal{A}$ is unbounded on L if for every $k \in \mathbb{N}$, there is a $w \in L$ and a run of $\mathcal{A}$ on w in which for each $c \in C$, at least k of the oracle queries for c are successful. The following can be shown similarly to Theorem 4.8, but using a multi-dimensional unboundedness predicate.

▶ **Theorem 4.10.** *Let $\mathcal{C}$ be a full trio with decidable emptiness. Given a VAS language L and a $\mathcal{C}$-counting automaton $\mathcal{A}$, it is decidable whether $\mathcal{A}$ is unbounded on L. Moreover, if $\mathcal{A}$ is bounded on L, then one can compute an upper bound $B \in \mathbb{N}$ for $\mathcal{A}$ on L.*

Factor inclusion. As a last example, we show how our results can be used to decide inclusion problems. Specifically, given a VAS language $L \subseteq \Sigma^*$, it is decidable whether $\Sigma^* \subseteq F(L)$. In fact, we show a more general result:

▶ **Theorem 4.11.** *If $\mathcal{C}$ is a full trio with decidable emptiness problem, then given a VAS language L and a language K from $\mathcal{C}$, it is decidable whether $K^* \subseteq F(L)$.*

Here, $\Sigma^* \subseteq F(L)$ is the special case where $K = \Sigma$. Recall that is is undecidable whether $L = \Sigma^*$ for VAS languages and for one-counter languages (OCL) (e.g. [15, Lemma 6.1]).

Similar to Theorem 4.8, deciding whether $\Sigma^* \subseteq F(L)$ is already undecidable for OCL L: For a given OCL $L \subseteq \Sigma^*$, pick a letter $c \notin \Sigma$ and note that $L' = c(Lc)^* \subseteq (\Sigma \cup \{c\})^*$ is effectively an OCL and $(\Sigma \cup \{c\})^* \subseteq F(L')$ if and only if $L = \Sigma^*$. Also, under the assumptions of the theorem, it is undecidable whether $K \subseteq F(L)$: If $L \subseteq \Sigma^*$ and $c \notin \Sigma$, then $c\Sigma^* c \subseteq F(cLc)$ if and only if $L = \Sigma^*$ (every full trio contains the regular set $c\Sigma^* c$).

Let us see how Theorem 4.11 follows from Theorem 3.1. Fix a language K from $\mathcal{C}$. We use the 1-dim. predicate $\mathfrak{p}_{\mathsf{fu}}$, which is satisfied on a set $L \subseteq \Sigma^*$ if and only if $K^* \subseteq F(L)$. Of course, axiom (i) holds by definition. Axiom (iii) follows by contraposition: Suppose that $K^* \subseteq F(L_1 L_2)$ and $K^* \not\subseteq F(L_1)$ with some $u \in K^* \setminus F(L_1)$. Let $v \in K^*$ be arbitrary. Then, since $K^* \subseteq F(L_1 L_2)$, we have $uv \in F(L_1 L_2)$. This means, there are $x, y \in \Sigma^*$ with $xuvy \in L_1 L_2$. Hence, we have $xuvy = w_1 w_2$ for some $w_i \in L_i$ for $i = 1, 2$. Then $|w_1| < |xu|$, because otherwise u would belong to $F(L_1)$. Therefore, v is a factor of w_2 and thus $v \in F(L_2)$. Hence, $K^* \subseteq F(L_2)$. Of course, a similar argument works if $K^* \subseteq F(L_1 L_2)$ and $K^* \not\subseteq F(L_2)$. This proves axiom (iii). Axiom (ii) can be shown the same way. Thus, by Theorem 3.1, it suffices to decide whether $K^* \subseteq F(R)$ for regular R, which follows from the fact that $\mathcal{C}$ is a full trio and has decidable emptiness.

5 Proof of the main result

We prove our decidability result using the KLMST decomposition. More specifically, we show a consequence that might be interesting in its own right.

▶ **Theorem 5.1.** *Given a VAS language $L \subseteq \Sigma^*$, one can compute $m, k \in \mathbb{N}$ and regular languages $R_{i,j} \subseteq \Sigma^*$, for $i \in [1, m]$, $j \in [1, k]$ so that*

$$L \subseteq \bigcup_{i=1}^{m} R_{i,1} \cdots R_{i,k} \qquad \text{and} \qquad R_{i,1} \times \cdots \times R_{i,k} \subseteq F_k(L) \text{ for every } i \in [1, m]. \tag{1}$$

We first show how to derive Theorem 3.1 from Theorem 5.1 and then proceed with the proof of Theorem 5.1, as it is much more technically complicated.

Proof of Theorem 3.1. Suppose Theorem 5.1 holds. Then, given a VAS language L, we compute $m, k \in \mathbb{N}$ and the regular languages $R_{i,j}$ for $i \in [1, m], j \in [1, k]$. We choose $R = \bigcup_{i=1}^{m} R_{i,1} \cdots R_{i,k}$. Then we have $L \subseteq R$. Let us show that $\mathfrak{p}(F_n(L))$ if and only if $\mathfrak{p}(F_n(R))$. If $\mathfrak{p}(F_n(L))$, then clearly $\mathfrak{p}(F_n(R))$, because $L \subseteq R$ implies $F_n(L) \subseteq F_n(R)$ and by axiom (i), this implies $\mathfrak{p}(F_n(R))$. Conversely, suppose $\mathfrak{p}(F_n(R))$. Then by axiom (ii), there is an $i \in [1, m]$ such that $\mathfrak{p}(F_n(R_i))$, where $R_i = R_{i,1} \cdots R_{i,k}$. According to axiom (iii), we can write $n = n_1 + \cdots + n_k$ such that $\mathfrak{p}$ holds for $S := F_{n_1}(R_{i,1}) \times \cdots \times F_{n_k}(R_{i,k})$. Note that by the choice of $R_{i,j}$, we have $R_{i,1} \times \cdots \times R_{i,k} \subseteq F_k(L)$ and therefore $S \subseteq F_n(L)$. This implies $\mathfrak{p}(F_n(L))$ by axiom (i).

Proof of Theorem 5.1. The remainder of this section is devoted to the proof of Theorem 5.1. Like the method for computing downward closures by Habermehl, Meyer, and Wimmel [28], the construction of the sets $R_{i,j}$ is based on Lambert's proof [38] of the decidability of the reachability problem for Petri nets. In order to be compatible with Lambert's exposition, we phrase our proof in terms of Petri nets instead of vector addition systems.

A *Petri net* $N = (P, T, \text{PRE}, \text{POST})$ consists of a finite set P of *places*, a finite set T of *transitions* and two mappings $\text{PRE}, \text{POST}\colon T \to \mathbb{N}^P$. Configurations of Petri net are elements of $\mathbb{N}^P$, called *markings*. For two markings M, M' we say that M' *dominates* M, denoted $M \leq M'$, if for every place $p \in P$, we have $M[p] \leq M'[p]$. The *effect* of a transition $t \in T$ is $\text{POST}(t) - \text{PRE}(t) \in \mathbb{Z}^P$, denoted $\Delta(t)$. If a marking M dominates $\text{PRE}(t)$ for a transition $t \in T$ then t is *fireable* in M and the result of firing t in marking M is $M' = M + \Delta(t)$, we write $M \xrightarrow{t} M'$. We extend notions of fireability and firing naturally to sequences of transitions, we also write $M \xrightarrow{w} M'$ for $w \in T^*$. The *effect* of $w \in T^*$ is sum of the effects of its letters, $\Delta(w) = \sum_{i=1}^{|w|} \Delta(w[i])$. For a Petri net $N = (P, T, \text{PRE}, \text{POST})$ and markings M_0, M_1, we define the language $L(N, M_0, M_1) = \{w \in T^* \mid M_0 \xrightarrow{w} M_1\}$. Hence, $L(N, M_0, M_1)$ is the set of transition sequences leading from M_0 to M_1. Moreover, let $L(N, M_0) = \bigcup_{M \in \mathbb{N}^P} L(N, M_0, M)$, i.e. the set of all the transition sequences fireable in M_0. A *labeled Petri net* is a Petri net $N = (P, T, \text{PRE}, \text{POST})$ together with an *initial marking* M_I, a *final marking* M_F, and a *labeling*, i.e. a homomorphism $h : T^* \to \Sigma^*$. The language *recognized by* the labeled Petri net is then defined as $L_h(N, M_I, M_F) = h(L(N, M_I, M_F))$.

It is folklore (and easy to see) that a language is a VAS language if and only if it is recognized by a labeled Petri net (and the translation is effective). Thus, it suffices to show Theorem 5.1 for languages of the form $L = h(L(N, M_I, M_F))$. Moreover, it is already enough to prove Theorem 5.1 for languages of the form $L(N, M_I, M_F)$. Indeed, observe that if we have constructed $R_{i,j}$ so that Eq. (1) is satisfied, then with $S_{i,j} = h(R_{i,j})$, we have $h(L) \subseteq \bigcup_{i=1}^{m} S_{i,1} \cdots S_{i,k}$ and $S_{i,1} \times \cdots \times S_{i,k} \subseteq F_k(h(L))$ for every $i \in [1, m]$. Thus from now on, we assume $L = L(N, M_I, M_F)$ for a fixed Petri net $N = (P, T, \text{PRE}, \text{POST})$.

The KLMST decomposition. Lambert's decision procedure [38] is a refinement of the previous ones by Mayr [43] and Kosaraju [37]. Leroux and Schmitz [39] recast it as an algorithm using WQO ideals and dubbed it *KLMST decomposition* after its inventors [37, 38, 43, 48].

The idea is the following. We disregard for a moment that a transition sequence has to keep all intermediate markings non-negative and only look for a sequence that may go negative on the way. It is standard technique to express the existence of such a sequence as a linear equation system $Ax = b$. As expected, solvability of this system is not sufficient for the existence of an actual run. However, if we are in the situation that we can find (a) runs that pump up all coordinates arbitrarily high and also (b) counterpart runs that remove those excess tokens again, then solvability of the equation system is also sufficient: We first increase all coordinates high enough, then we execute our positivity-ignoring sequence, and then we pump down again. Roughly speaking, the achievement of the KLMST decomposition is to put us in the latter situation, which we informally call *perfect circumstances*.

To this end, one uses a data structure, in Lambert's version called *marked graph-transition sequence (MGTS)*, which restricts the possible runs of the Petri net. If the MGTS satisfies a condition that realizes the above perfect circumstances, then it is called *perfect*. Unsurprisingly, not every MGTS is perfect. However, part of the procedure is a decomposition of an imperfect MGTS into finitely many MGTS that are less imperfect. Moreover, this decomposition terminates in a finite set of perfect MGTS. Thus, applied to an MGTS whose restriction is merely to start in M_I and end in M_F, then the decomposition yields finitely many perfect MGTS $\mathcal{N}_1, \ldots, \mathcal{N}_n$ such that the runs from M_I to M_F are precisely those conforming to at least one of the MGTS. Moreover, checking whether $\mathcal{N}_i$ admits a run amounts to solving a linear equation system.

Basic notions. Let us introduce some notions used in Lambert's proof. We extend the set of configurations $\mathbb{N}^d$ into $\overline{\mathbb{N}}^d$. We extend the notion of transition firing into $\overline{\mathbb{N}}^d$ naturally, by defining $\omega - k = \omega = \omega + k$ for every $k \in \mathbb{N}$. For $u, v \in \overline{\mathbb{N}}^d$ we write $u \leq_\omega v$ if $u[i] = v[i]$ or $v[i] = \omega$. Intuitively reaching a configuration with ω at some places means that it is possible to reach configurations with values ω substituted by arbitrarily high values.

A key notion in [38] is that of MGTS, which formulate restrictions on paths in Petri nets. A *marked graph-transition sequence (MGTS)* for our Petri net $N = (P, T, \text{Pre}, \text{Post})$ is a finite sequence $C_0, t_1, C_1 \dots C_{n-1}, t_n, C_n$, where t_i are transitions from T and C_i are precovering graphs, which are defined next. A *precovering graph* is a quadruple $C = (G, m, m^{\text{init}}, m^{\text{fin}})$, where $G = (V, E, h)$ is a finite, strongly connected, directed graph with $V \subseteq \overline{\mathbb{N}}^P$ and labeling $h\colon E \to T$, and three vectors: a *distinguished* vector $m \in V$, an *initial* vector $m^{\text{init}} \in \overline{\mathbb{N}}^P$, and a *final* vector $m^{\text{fin}} \in \overline{\mathbb{N}}^P$. A precovering graph has to meet two conditions: First, for every edge $e = (m_1, m_2) \in E$, there is an $m_3 \in \overline{\mathbb{N}}^P$ with $m_1 \xrightarrow{h(e)} m_3 \leq_\omega m_2$. Second, we have $m^{\text{init}}, m^{\text{fin}} \leq_\omega m$. Additionally we impose the restriction on MGTS that the initial vector of C_0 equals M_I and the final vector of C_n equals M_F.

Languages of MGTS. Each precovering graph can be treated as a finite automaton. For $m_1, m_2 \in V$, we denote by $L(C, m_1, m_2)$ the set of all $w \in T^*$ read on a path from m_1 to m_2. Moreover, let $L(C) = L(C, m, m)$. MGTS have associated languages as well. Let $\mathcal{N} = C_0, t_1, C_1 \dots C_{n-1}, t_n, C_n$ be an MGTS of a Petri net N, where $C_i = (G_i, m_i, m_i^{\text{init}}, m_i^{\text{fin}})$. Its language $L(\mathcal{N})$ is the set of all words of the form $w = w_0 t_1 w_1 \cdots w_{n-1} t_n w_n \in T^*$ where: $w_i \in L(C_i)$ for each $i \in [0, n]$ and (ii) there exist markings $\mu_0, \mu_0', \mu_1, \mu_1', \dots, \mu_n, \mu_n' \in \mathbb{N}^P$ such that $\mu_i \leq_\omega m_i^{\text{init}}$ and $\mu_i' \leq_\omega m_i^{\text{fin}}$ and $\mu_0 \xrightarrow{w_0} \mu_0' \xrightarrow{t_1} \mu_1 \xrightarrow{w_1} \dots \xrightarrow{w_{n-1}} \mu_{n-1}' \xrightarrow{t_n} \mu_n \xrightarrow{w_n} \mu_n'$. Notice that by (ii) and the restriction that $m_0^{\text{init}} = M_I$ and $m_n^{\text{fin}} = M_F$, we have $L(\mathcal{N}) \subseteq L(N, M_I, M_F)$ for any MGTS $\mathcal{N}$.

Hence roughly speaking, $L(\mathcal{N})$ is the set of runs that contain the transitions $t_1, \dots, t_n$ and additionally markings before and after firing these transitions are prescribed on some places: this is exactly what the restrictions $\mu_i \leq_\omega m_i^{\text{init}}$, $\mu_i' \leq_\omega m_i^{\text{fin}}$ impose.

Notice that at the moment we do not expect that values ω occurring at $m_i, m_i^{\text{init}}, m_i^{\text{fin}}$ impose any restriction on the form of accepted runs. Meaning of ω values is reflected in the notion of *perfect* MGTS described later. As an immediate consequence of the definition, we observe that for every MGTS $\mathcal{N} = C_0, t_1, C_1 \dots C_{n-1}, t_n, C_n$ we have

$$L(\mathcal{N}) \subseteq L(C_0) \cdot \{t_1\} \cdot L(C_1) \cdots L(C_{n-1}) \cdot \{t_n\} \cdot L(C_n). \tag{2}$$

Perfect MGTS. As announced above, Lambert calls MGTS with a paricular property *perfect* [38]. Since the precise definition is involved and we do not need all the details, it is enough for us to mention a selection of properties of perfect MGTS. Intuitively, in perfect MGTSes, the value ω on place p in m_i means that inside of the component C_i, the token count in place p can be made arbitrarily high. In [38] it is shown (Theorem 4.2 (page 94) together with the preceding definition) that

▶ **Theorem 5.2** ([38]). *For a Petri net N one can compute finitely many perfect MGTS $\mathcal{N}_1, \dots, \mathcal{N}_m$ such that $L(N, M_I, M_F) = \bigcup_{i=1}^m L(\mathcal{N}_i)$.*

Moreover, by Corollary 4.1 in [38] (page 93), given a perfect MGTS $\mathcal{N}$, it is decidable whether $L(\mathcal{N}) \neq \emptyset$. Therefore, our task reduces to the following. We have a perfect MGTS $\mathcal{N}$ with $L(\mathcal{N}) \neq \emptyset$ and want to compute regular languages $R_1, \dots, R_k$ such that $L(\mathcal{N}) \subseteq R_1 \cdots R_k$ and $R_1 \times \cdots \times R_k \subseteq F_k(L(\mathcal{N}))$. (Note that if the MGTS have

different lengths, we can always fill up with $\{\varepsilon\}$). We choose $R_1, \ldots, R_k$ to be the sequence $L(C_0), \{t_1\}, L(C_1), \ldots, L(C_{n-1}), \{t_n\}, L(C_n)$. Then Eq. (2) tells us that this achieves $L(\mathcal{N}) \subseteq R_1 \cdots R_k$ and all that remains to be shown is

$$L(C_0) \times \{t_1\} \times L(C_1) \times \cdots \times L(C_{n-1}) \times \{t_n\} \times L(C_n) \subseteq F_{2n+1}(L(\mathcal{N})). \tag{3}$$

Constructing runs. In order to show Eq. (3), we employ a simplified version of Lambert's iteration lemma, which involves covering sequences. Let C be a precovering graph for a Petri net $N = (P, T, \textsc{Pre}, \textsc{Post})$ with a distinguished vector $m \in \overline{\mathbb{N}}^P$ and initial vector $m^{\text{init}} \in \overline{\mathbb{N}}^P$. A sequence $u \in L(C) \cap L(N, m^{\text{init}})$ is called a *covering sequence for* C if for every place $p \in P$ we have either 1) $m^{\text{init}}[p] = \omega$, or 2) $m[p] = m^{\text{init}}[p]$ and $\Delta(u)[p] = 0$, or 3) $m[p] = \omega$ and $\Delta(u)[p] > 0$. This corresponds intuitively to the three possible cases for the set of runs in N crossing the component C in a place p: (i) runs that can have arbitrarily high value on p when entering C, (ii) runs where, when entering C, p has a fixed value, and the tokens in p cannot be pumped inside of C, or (iii) runs where, when entering C, p has a fixed value, but it can be pumped up inside of C.

Let $\mathcal{N} = C_0, t_1, C_1 \ldots C_{n-1}, t_n, C_n$ be an MGTS, where $C_i = (V_i, E_i, h_i)$ is a precovering graph, and let the distinguished vertex be m_i and initial vertex be m_i^{init}. If $\mathcal{N}$ is a perfect MGTS then according to the definition from [38] (page 92), for every $i \in [0, n]$ there exists a covering sequence $u_i \in L(C_i) \cap L(N, m_i^{\text{init}})$. This corresponds to the mentioned intuition that ω values imply arbitrarily high values. As a direct consequence of Lemma 4.1 in [38] (page 92), Lambert's iteration lemma, we obtain:

▶ **Lemma 5.3.** *Let $\mathcal{N} = C_0, t_1, C_1 \ldots C_{n-1}, t_n, C_n$ be a perfect MGTS and let x_i be a covering sequences for C_i for $i \in [0, n]$. Then there exist words $y_i \in T^*$ for $i \in [0, n]$ such that $x_0 y_0 \cdot t_1 \cdot x_1 y_1 \cdots x_{n-1} y_{n-1} \cdot t_n \cdot x_n y_n \in L(\mathcal{N})$.*

Lemma 5.3 is obtained from Lemma 4.1 in [38] as follows. The word u_i there is our x_i and v_i there is an arbitrary covering sequence of C_i reversed. Then, our y_i is set to $u_i^{k-1} \beta_i(w_i)^k (v_i)^k$ for some $k \geq k_0$. The only technical part of the proof of Theorem 5.1 is the following lemma.

▶ **Lemma 5.4.** *Let C be a precovering graph for a Petri net $N = (P, T, \textsc{Pre}, \textsc{Post})$ with a distinguished vector $m \in \overline{\mathbb{N}}^P$ and initial vector $m^{\text{init}} \in \overline{\mathbb{N}}^P$ such that $s \in L(C) \cap L(N, m^{\text{init}})$ is a covering sequence. Then for every $v \in L(C)$ there is a covering sequence for C of the form uv, for some $u \in T^*$.*

Proof. Intuitively, we do the following. The existence of a covering sequence means that one can obtain arbitrarily high values on places p where $m[p] = \omega$. Thus, in order to construct a covering sequence containing v as a suffix, we first go very high on the ω places, so high that adding v as a suffix later will still result in a sequence with positive effect.

Let us make this precise. Executing the sequence v might have a negative effect in a place $p \in P$ with $m[p] = \omega$. Let $k \in \mathbb{N}$ be the largest possible negative effect a prefix of v can have in any coordinate. Note that since s is a covering sequence, s^k is a covering sequence as well. We claim that $s^k v$ is also a covering sequence. It is contained in $L(C)$ and fireable at m^{init}. Moreover, by choice of k, the sequence $s^k v$ has a positive effect on each p with $m[p] = \omega$. If $m[p] < \omega$, then $\Delta(s)[p] = 0 = \Delta(v)[p]$ and hence $\Delta(s^k v)[p] = 0$. ◀

Using Lemma 5.3 and Lemma 5.4, it is now easy to show Eq. (3). Given words $v_i \in T^*$ with $v_i \in L(C_i)$ for $i \in [0, n]$, we use Lemma 5.4 to choose $x_i \in T^*$ such that $x_i v_i$ is a covering sequence of C_i for $i \in [0, n]$. By Lemma 5.3, we can find $w_1, \ldots, w_n$ so that the word $x_0 v_0 w_0 \cdot t_1 \cdot x_1 v_1 w_1 \cdots x_{n-1} v_{n-1} w_{n-1} \cdot t_n \cdot x_n v_n w_n$ belongs to $L(\mathcal{N})$, and thus $(v_0, t_1, v_1, \ldots, v_{n-1}, t_n, v_n) \in F_{2n+1}(L(\mathcal{N}))$, which proves Eq. (3).

References

1 Mohamed Faouzi Atig and Pierre Ganty. Approximating Petri net reachability along context-free traces. In *FSTTCS 2011*, volume 13, pages 152–163, Dagstuhl, Germany, 2011.

2 Mohamed Faouzi Atig and Peter Habermehl. On Yen's path logic for Petri nets. *Int. J. Found. Comput. Sci.*, 22(4):783–799, 2011.

3 Jean Berstel. *Transductions and Context-Free Languages*. Teubner, 1979.

4 Michel Blockelet and Sylvain Schmitz. Model checking coverability graphs of vector addition systems. In *MFCS 2011*, pages 108–119, Berlin, Heidelberg, 2011. Springer.

5 Mikołaj Bojańczyk. It is undecidable if two regular tree languages can be separated by a deterministic tree-walking automaton. *Fundam. Inform.*, 154(1-4):37–46, 2017.

6 Rémi Bonnet, Alain Finkel, Jérôme Leroux, and Marc Zeitoun. Place-Boundedness for Vector Addition Systems with one zero-test. In *FSTTCS 2010*, pages 192–203, 2010.

7 Ahmed Bouajjani, Javier Esparza, and Tayssir Touili. A generic approach to the static analysis of concurrent programs with procedures. *International Journal of Foundations of Computer Science*, 14(04):551–582, 2003.

8 Sagar Chaki, Edmund M. Clarke, Nicholas A. Kidd, Thomas W. Reps, and Tayssir Touili. *Verifying Concurrent Message-Passing C Programs with Recursive Calls*, pages 334–349. Springer-Verlag, Berlin Heidelberg, 2006. doi:10.1007/11691372_22.

9 Pierre Chambart, Alain Finkel, and Sylvain Schmitz. Forward analysis and model checking for trace bounded WSTS. *Theoretical Computer Science*, 637:1–29, 2016.

10 Lorenzo Clemente, Wojciech Czerwiński, Sławomir Lasota, and Charles Paperman. Regular separability of Parikh automata. In *ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 117:1–117:13, 2017.

11 Lorenzo Clemente, Wojciech Czerwiński, Sławomir Lasota, and Charles Paperman. Separability of reachability sets of vector addition systems. In *STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 24:1–24:14, 2017.

12 Thomas Colcombet. Regular cost functions, part I: logic and algebra over words. *Logical Methods in Computer Science*, 9(3), 2013.

13 Wojciech Czerwiński and Sławomir Lasota. Regular separability of one counter automata. In *LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017.

14 Wojciech Czerwiński, Sławomir Lasota, Roland Meyer, Sebastian Muskalla, K Narayan Kumar, and Prakash Saivasan. Regular separability of well structured transition systems. *CoRR*, abs/1702.05334, 2018.

15 Wojciech Czerwiński, Wim Martens, Lorijn van Rooijen, Marc Zeitoun, and Georg Zetzsche. A characterization for decidable separability by piecewise testable languages. *Discrete Mathematics & Theoretical Computer Science*, 19(4), 2017.

16 Stéphane Demri. On selective unboundedness of VASS. In *Proceedings of the 12th International Workshops on Verification of Infinite State Systems (INFINITY'10)*, volume 39 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–15, Singapore, 2010. doi:10.4204/EPTCS.39.1.

17 Stéphane Demri, Alain Finkel, Valentin Goranko, and Govert van Drimmelen. Model-checking CTL* over flat Presburger counter systems. *Journal of Applied Non-Classical Logics*, 20(4):313–344, 2010.

18 Javier Esparza and Pierre Ganty. Complexity of pattern-based verification for multithreaded programs. In *POPL 2011*, pages 499–510, 2011.

19 Javier Esparza, Pierre Ganty, and Rupak Majumdar. A perfect model for bounded verification. In *LICS 2012*, pages 285–294, 2012.

20 Javier Esparza, Pierre Ganty, and Tomás Poch. Pattern-based verification for multithreaded programs. *ACM Trans. Program. Lang. Syst.*, 36(3):9:1–9:29, 2014.

21 Pierre Ganty, Rupak Majumdar, and Benjamin Monmege. Bounded underapproximations. *Formal Methods in System Design*, 40(2):206–231, 2012.

22 Paweł Gawrychowski, Dalia Krieger, Narad Rampersad, and Jeffrey Shallit. Finding the growth rate of a regular or context-free language in polynomial time. *International Journal of Foundations of Computer Science*, 21(04):597–618, 2010.

23 Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. Well-structured languages. *Acta Informatica*, 44(3–4):249–288, 2007.

24 Seymour Ginsburg and Edwin H Spanier. Bounded algol-like languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964.

25 Seymour Ginsburg and Edwin H. Spanier. Bounded regular sets. *Proceedings of the American Mathematical Society*, 17(5):1043–1049, 1966. doi:10.2307/2036087.

26 Jean Goubault-Larrecq and Sylvain Schmitz. Deciding piecewise testable separability for regular tree languages. In *ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 97:1–97:15, 2016.

27 Sheila A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7(3):311–324, 1978. doi:10.1016/0304-3975(78)90020-8.

28 Peter Habermehl, Roland Meyer, and Harro Wimmel. The downward-closure of Petri net languages. In *ICALP 2010*, volume 6199 of *Lecture Notes in Computer Science*, pages 466–477. Springer-Verlag, 2010.

29 Matthew Hague and Anthony Widjaja Lin. Model checking recursive programs with numeric data types. In *CAV 2011*, volume 6806 of *Lecture Notes in Computer Science*, pages 743–759. Springer-Verlag, 2011.

30 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *LICS 2008*, pages 452–461, 2008.

31 Leonard H. Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1):94–98, 1969. doi:10.1016/S0021-9800(69)80111-0.

32 Dirk Hauschildt. *Semilinearity of the reachability set is decidable for Petri nets*. PhD thesis, Fachbereich Informatik, Universität Hamburg, 1990. Also available as Technical Report No. FBI-HH-B-146/90, Fachbereich Informatik, University of Hamburg.

33 Dirk Hauschildt and Matthias Jantzen. Petri net algorithms in the theory of matrix grammars. *Acta Informatica*, 31(8):719–728, Aug 1994.

34 Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society. Third Series*, 2:326–336, 1952. doi:10.1112/plms/s3-2.1.326.

35 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

36 Matthias Jantzen. On the hierarchy of petri net languages. *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications*, 13(1):19–30, 1979. URL: http://www.numdam.org/item?id=ITA_1979__13_1_19_0.

37 S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *STOC 1982, May 5-7, 1982, San Francisco, California, USA*, pages 267–281, 1982.

38 Jean-Luc Lambert. A structure to decide reachability in Petri nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992.

39 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems demystified. In *LICS 2015*, 2015.

40 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the coverability problem for pushdown vector addition systems in one dimension. In *ICALP 2015*, pages 324–336, Berlin Heidelberg, 2015. Springer-Verlag. doi:10.1007/978-3-662-47666-6_26.

41 Zhenyue Long, Georgel Calin, Rupak Majumdar, and Roland Meyer. Language-theoretic abstraction refinement. In *FASE 2012*, pages 362–376. Springer, 2012.

42 A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 12(1):38–42, 1976.

43 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *STOC 1981, May 11-13, 1981, Milwaukee, Wisconsin, USA*, pages 238–246, 1981.

44 Thomas Place and Marc Zeitoun. Concatenation hierarchies: New bottle, old wine. In *CSR 2017, Kazan, Russia, June 8-12, 2017, Proceedings*, pages 25–37, 2017.

45 Thomas Place and Marc Zeitoun. Separation for dot-depth two. In *LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017.

46 Klaus Reinhardt. The "trio-zoo"—classes of formal languages generated from one language by rational transduction. Unpublished manuscript.

47 Klaus Reinhardt. Reachability in Petri nets with inhibitor arcs. *Electronic Notes in Theoretical Computer Science*, 223:239–264, 2008. Proceedings of the Second Workshop on Reachability Problems in Computational Models (RP 2008). doi:10.1016/j.entcs.2008.12.042.

48 George S. Sacerdote and Richard L. Tenney. The decidability of the reachability problem for vector addition systems (preliminary version). In *STOC 1977*, pages 61–76. ACM, 1977.

49 Rani Siromoney. A characterization of semilinear sets. *Proceedings of the American Mathematical Society*, 21(3):689–694, 1969.

50 Hsu-Chun Yen. A unified approach for deciding the existence of certain Petri net paths. *Information and Computation*, 96(1):119–137, 1992.

51 Georg Zetzsche. The emptiness problem for valence automata over graph monoids. To appear in Information and Computation.

52 Georg Zetzsche. An approach to computing downward closures. In *ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 440–451, 2015.

Reachability and Distances under Multiple Changes

Samir Datta
Chennai Mathematical Institute & UMI ReLaX, Chennai, India
sdatta@cmi.ac.in

Anish Mukherjee
Chennai Mathematical Institute, Chennai, India
anish@cmi.ac.in

Nils Vortmeier
TU Dortmund University, Dortmund, Germany
nils.vortmeier@tu-dortmund.de

Thomas Zeume
TU Dortmund University, Dortmund, Germany
thomas.zeume@tu-dortmund.de

Abstract

Recently it was shown that the transitive closure of a directed graph can be updated using first-order formulas after insertions and deletions of single edges in the dynamic descriptive complexity framework by Dong, Su, and Topor, and Patnaik and Immerman. In other words, Reachability is in DynFO.

In this article we extend the framework to changes of multiple edges at a time, and study the Reachability and Distance queries under these changes. We show that the former problem can be maintained in DynFO$(+,\times)$ under changes affecting $\mathcal{O}(\frac{\log n}{\log\log n})$ nodes, for graphs with n nodes. If the update formulas may use a majority quantifier then both Reachability and Distance can be maintained under changes that affect $\mathcal{O}(\log^c n)$ nodes, for fixed $c \in \mathbb{N}$. Some preliminary results towards showing that distances are in DynFO are discussed.

2012 ACM Subject Classification Theory of computation → Models of computation, Theory of computation → Finite Model Theory

Keywords and phrases dynamic complexity, reachability, distances, complex changes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.120

Related Version A full version is available at [6], `https://arxiv.org/abs/1804.08555`.

Funding The authors acknowledge the financial support by the DAAD-DST grant "Exploration of New Frontiers in Dynamic Complexity". The first and the second authors were partially funded by a grant from Infosys foundation. The second author was partially supported by a TCS PhD fellowship. The last two authors acknowledge the financial support by DFG grant SCHW 678/6-2 on "Dynamic Expressiveness of Logics".

Acknowledgements We thank Rajit Datta and Partha Mukhopadhyay for pointing us to the Agrawal-Vinay technique for computing determinants. We also thank Thomas Schwentick for many illuminating discussions on dynamic complexity.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 120; pp. 120:1–120:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In today's databases, data sets are often large and subject to frequent changes. In use cases where only a fixed set of queries has to be evaluated on such data, it is not efficient to re-evaluate queries after each change, and therefore dynamic approaches have been considered. The idea is that when a database $\mathcal{D}$ is modified by changing a set $\Delta\mathcal{D}$ of tuples then the result of a query is recomputed by using its result on $\mathcal{D}$, the set $\Delta\mathcal{D}$, and possibly other previously computed auxiliary data.

One such dynamic approach is the *dynamic descriptive complexity* approach, formulated independently by Dong, Su, and Topor [8], as well as Patnaik and Immerman [19]. In their framework the query result and the auxiliary data are represented by relations, and updates of the auxiliary relations are performed by evaluating first-order formulas. The class of queries that can be maintained in this fashion constitutes the class DynFO. The motivation to use first-order logic as the vehicle for updates is that its evaluation is highly parallelizable and, in addition, that it corresponds to the relational algebra which is the core of SQL. Hence, if a query result can be maintained using a first-order update program, this program can be translated into equivalent SQL queries.

While it is desirable to understand how to update query results under complex changes $\Delta\mathcal{D}$, the focus of dynamic descriptive complexity so far has been on single tuple changes. The reason is that for many queries our techniques did not even suffice to tackle this case.

In recent years, however, we have seen several new techniques for maintaining queries. The Reachability query – one of the main objects of study in dynamic descriptive complexity – has been shown to be in DynFO using a linear algebraic method and a simulation technique [4]. The latter has been advanced into a very powerful tool: for showing that a query can be maintained in DynFO, it essentially suffices to show that it can be maintained for $\log n$ many change steps after initializing the auxiliary data by an AC^1 pre-computation[1] [5], where n is the size of the database's (active) domain. This tool has been successfully applied to show that all queries expressible in monadic second order logic can be maintained in DynFO on structures of bounded treewidth.

Those new techniques motivate a new attack on more complex changes $\Delta\mathcal{D}$. But what are reasonable changes to look at? Updating a query after a change $\Delta\mathcal{D}$ that replaces the whole database by a new database is essentially equivalent to the static evaluation problem with built-in relations: the stored auxiliary data has to be helpful for every possible new database, and therefore plays the role of built-in relations. Thus changes should be restricted in some way. Three approaches come to mind immediately: to only allow changes of restricted size; to restrict changes structurally; or to define changes in a declarative way.

In this article we focus on the first approach. Before discussing our results we shortly outline the other two approaches.

There is a wide variety of structural restrictions. For example, the change set $\Delta\mathcal{D}$ could only change the database locally or in such a way that the changes affect auxiliary relations only locally, e.g., if edges are inserted into distinct connected components it should be easier to maintain reachability. Another option is to restrict $\Delta\mathcal{D}$ to be of a certain shape, examples studied in the literature are cartesian-closed changes [8] and deletions of anti-chains [7].

A declarative mechanism for changing a database is to provide a set of parameterised rules that state which tuples should be changed depending on a parameter provided by a user. For example, a rule $\rho(x, y; z)$ could state that all edges (x, y) shall be inserted into a

[1] Readers not familiar with the circuit class AC^1 may safely think of LOGSPACE pre-computations.

graph such that x and y are connected to the parameter z. First-order logic as a declarative mean to change databases has been studied in [20], where it was shown that undirected reachability can be maintained under insertions defined by first-order formulas, and single tuple deletions.

In this article we study changes of small size with a focus on the Reachability and Distance queries. As can be seen from the discussion above, the former query has been well-studied in diverse settings of dynamic descriptive complexity, and therefore results on its maintainability under small changes serve as an important reference point.

There is another reason to study Reachability under non-constant size changes. Recall that Reachability is complete for the static complexity class NL. The result that Reachability is in DynFO does not imply NL $\subseteq$ DynFO, as DynFO is only known to be closed under very weak reductions, called bounded first-order reductions, under which Reachability is not NL-complete [19]. In short, these reductions demand that whenever a bit of an instance is changed, then only constantly many bits change in the image of the instance under the reduction. When a query such as Reachability is maintainable under larger changes, then this restriction may be relaxed and might yield new maintainability results for other queries under single edge changes.

In this work we show that Reachability can be maintained under changes of non-constant size. Since our main interest is the study of changes of non-constant size, we assume throughout the article that all classes come with built-in arithmetic and denote, e.g., by DynFO$(+,\times)$ the class of queries that can be maintained with first-order updates in the presence of a built-in linear addition and multiplication relations. How our results can be adapted to classes without built-in arithmetic is discussed towards the end of Section 3.

▶ **Theorem 1.** *Reachability can be maintained in* DynFO$(+,\times)$ *under changes that affect* $\mathcal{O}(\frac{\log n}{\log\log n})$ *nodes of a graph, where n is the number of nodes of the graph.*

The distance query was shown to be in DynFO+Maj by Hesse [13], where the class DynFO+Maj allows to specify updates with first-order formulas that may include majority quantifiers (equivalently, updates can be specified by uniform TC^0 computations). We generalize Hesse's result to changes of size polylogarithmic in the size of the domain.

▶ **Theorem 2.** *Reachability and Distance can be maintained in* DynFO+Maj$(+,\times)$ *under changes that affect* $\mathcal{O}(\log^c n)$ *nodes of a graph, where $c \in \mathbb{N}$ is fixed and n is the number of nodes of the graph.*

One of the important open questions of dynamic descriptive complexity is whether distances can be maintained in DynFO, even under single edge changes. We contribute to the solution of this question by discussing how distances can be maintained in a subclass of DynFO+Maj$(+,\times)$ that is only slightly stronger than DynFO$(+,\times)$.

Organization

After recapitulating notations in Section 2, we adapt the dynamic complexity framework to bulk changes in Section 3. Our main results, maintainability of reachability and distances under multiple changes, are proved in Section 4 and Section 5. We conclude with a discussion in Section 6.

2 Preliminaries

In this section we review basic definitions and results from finite model theory and databases.

We consider finite relational structures over relational signatures $\tau = \{R_1, \ldots, R_\ell\}$, where each R_i is a relational symbol of arity $\mathrm{Ar}(R_i)$. A τ-structure $\mathcal{D}$ consists of a finite domain D and relations $R_i^{\mathcal{D}}$ over D of arity $\mathrm{Ar}(R_i)$, for each $i \in \{1, \ldots, \ell\}$. The *active domain* $\mathrm{adom}(\mathcal{D})$ of a structure $\mathcal{D}$ contains all elements used in some tuple of $\mathcal{D}$. Since the motivation to study dynamic complexity originates from database theory, we use terminology from this area. In particular we use the terms "relational structure" and "relational database" synonymously.

We study the queries Reachability and Distance. Reachability asks, given a directed graph G, for all pairs s, t of nodes such that there is a path from s to t in G. Distance asks for the length of the shortest path between any pair of reachable nodes.

We assume familiarity with first-order logic FO and refer to [16] for an introduction. The logic FO+Maj extends FO by allowing majority quantifiers. Such quantifiers can ask whether more than half of all elements satisfy a given formula. We write FO(+,×) and FO+Maj(+,×) to denote that formulas have access to built-in relations $\leq, +, \times$ which are interpreted as linear order, addition and multiplication on the domain of the underlying structure. We note that FO(+,×) and FO+Maj(+,×) are equal to the circuit classes (DLOGTIME-)uniform AC^0 and TC^0, respectively [2].

In FO(+, ×), each tuple $(a_1, \ldots, a_c)$ encodes a number from $[n^c - 1]_0 \stackrel{\text{def}}{=} \{0, \ldots n^c - 1\}$. We will henceforth identify tuples over the domain and numbers.

It is well-known that FO(+, ×) supports arithmetic on numbers with polylog bits. Furthermore, iterated addition and multiplication for polylog many numbers with polylog bits can be expressed in FO(+, ×). More precisely:

▶ **Lemma 3** (cf. [14, Theorem 5.1]). *Suppose φ is a* FO(+, ×) *formula that defines $r \in \mathcal{O}(\log^c n)$ polylog bit numbers $a_1, \ldots, a_r$, then there are formulas ψ_+ and $\psi_\times$ that define the sum and product of $a_1, \ldots, a_r$, respectively.*

Due to these facts, many calculations can be defined in FO(+, ×). In particular, primes can be identified, and $\frac{\log n}{\log \log n}$ numbers of $\log \log n$ bits each can be encoded and decoded in $\log n$ bit numbers.

Suppose $p_1, \ldots, p_m$ are primes whose product is N. Then each number $A < N$ can be uniquely represented as a tuple $\bar{a} = (a_1, \ldots, a_m)$ where $a_i = A \mod p_i$. The tuple $\bar{a}$ is called *Chinese remainder representation* (CRR) of A. The number A can recovered from $\bar{a}$ via $A = \sum_i a_i h_i C_i - rN$, where $C_i = \frac{N}{m_i}$, h_i is the inverse of C_i modulo m_i, and $r = \sum_{i=1}^{m} \lfloor \frac{x_i h_i}{m_i} \rfloor$ [14, p. 702]. Due to Lemma 3, in FO(+, ×) one can encode and decode $\mathcal{O}(\log n)$ bit numbers into their CRR defined by $\mathcal{O}(\log n)$ primes with $\mathcal{O}(\log \log n)$ bits.

In this article we use basic notions and results from linear algebra which are introduced when they are needed. Throughout the article, a matrix with $\mathcal{O}(n^d)$ rows and columns and entries in $[n^c]_0$ will be represented by a relation R that contains a tuple $(\bar{r}, \bar{c}, \bar{v})$ if and only if the value at row $\bar{r}$ and column $\bar{c}$ is $\bar{v}$.

3 Dynamic Framework for Multiple Changes

We briefly repeat the essentials of dynamic complexity, closely following [21], and discuss generalisations due to changes of non-constant size.

The goal of a dynamic program is to answer a given query on an *input database* subjected to changes that insert or delete tuples. The program may use an auxiliary data structure represented by an *auxiliary database* over the same domain. Initially, both input and auxiliary database are empty; and the domain is fixed during each run of the program.

Changes. In previous work, changes of single tuples have been represented as explicit parameters for the formulas used to update the auxiliary relations. Non-constant size changes cannot be represented in this fashion. An alternative is to represent changes implicitly by giving update formulas access to the old input database as well as to the changed input database [11]. Here, we opt for this approach.

For a database $\mathcal{D}$ over domain D and schema τ, a change $\Delta\mathcal{D}$ consists of sets R^+ and R^- of tuples for each relation symbol $R \in \tau$. The result $\mathcal{D} + \Delta\mathcal{D}$ of an application of the change $\Delta\mathcal{D}$ to $\mathcal{D}$ is the input database where $R^{\mathcal{D}}$ is changed to $(R^{\mathcal{D}} \cup R^+) \setminus R^-$. The *size* of $\Delta\mathcal{D}$ is the total number of tuples in relations R^+ and R^- and the set of *affected elements* is the (active) domain of tuples in $\Delta\mathcal{D}$.

Dynamic Programs and Maintenance of Queries. A dynamic program consists of a set of update rules that specify how auxiliary relations are updated after changing the input database. An *update rule* for updating an ℓ-ary auxiliary relation T after a change is a first-order formula φ over schema $\tau \cup \tau_{\text{aux}}$ with ℓ free variables, where τ_{aux} is the schema of the auxiliary database. After a change $\Delta\mathcal{D}$, the new version of T is $T \stackrel{\text{def}}{=} \{\vec{a} \mid (\mathcal{D} + \Delta\mathcal{D}, \mathcal{A}) \models \varphi(\vec{a})\}$ where $\mathcal{D}$ is the old input database and $\mathcal{A}$ is the current auxiliary database. Note that a dynamic program can choose to have access to the old input database by storing it in its auxiliary relations.

For a state $\mathcal{S} = (\mathcal{D}, \mathcal{A})$ of the dynamic program $\mathcal{P}$ with input database $\mathcal{D}$ and auxiliary database $\mathcal{A}$ we denote the state of the program after applying a change sequence α and updating the auxiliary relations accordingly by $\mathcal{P}_\alpha(\mathcal{S})$.

The dynamic program *maintains* a q-ary query $\mathcal{Q}$ under changes that affect k elements (under changes of size k, respectively) if it has a q-ary auxiliary relation Q that at each point stores the result of $\mathcal{Q}$ applied to the current input database. More precisely, for each non-empty sequence α of changes that affect k elements (changes of size k, respectively), the relation Q in $\mathcal{P}_\alpha(\mathcal{S}_\emptyset)$ and $\mathcal{Q}(\alpha(\mathcal{D}_\emptyset))$ coincide, where $\mathcal{D}_\emptyset$ is an empty input structure, $\mathcal{S}_\emptyset$ is the auxiliary database with empty auxiliary relations over the domain of $\mathcal{D}_\emptyset$, and $\alpha(\mathcal{D}_\emptyset)$ is the input database after applying α.

If a dynamic program maintains a query, we say that the query is in DynFO. Similarly to DynFO one can define the class of queries DynFO$(+, \times)$ that allows for three particular auxiliary relations that are initialised as a linear order and the corresponding addition and multiplication relations. Other classes are defined accordingly.

For many natural queries $\mathcal{Q}$, in order to show that $\mathcal{Q}$ can be maintained, it is enough to show that the query can be maintained for a bounded number of steps. Intuitively, this is possible for queries for which isolated elements do not influence the query result, if there are many such elements. Formally, a query $\mathcal{Q}$ is *almost domain-independent* if there is a $c \in \mathbb{N}$ such that $\mathcal{Q}(\mathcal{A}) \restriction (\text{adom}(\mathcal{A}) \cup B) = \mathcal{Q}(\mathcal{A} \restriction (\text{adom}(\mathcal{A}) \cup B))$ for all structures $\mathcal{A}$ and sets $B \subseteq A \setminus \text{adom}(\mathcal{A})$ with $|B| \geq c$.

A query $\mathcal{Q}$ is $(\mathcal{C}, f)$-*maintainable*, for some complexity class $\mathcal{C}$ and some function $f : \mathbb{N} \to \mathbb{R}$, if there is a dynamic program $\mathcal{P}$ and a $\mathcal{C}$-algorithm $\mathbb{A}$ such that for each input database $\mathcal{D}$ over a domain of size n, each linear order $\leq$ on the domain, and each change sequence α of length $|\alpha| \leq f(n)$, the relation Q in $\mathcal{P}_\alpha(\mathcal{S})$ and $\mathcal{Q}(\alpha(\mathcal{D}))$ coincide, where $\mathcal{S} = (\mathcal{I}, \mathbb{A}(\mathcal{I}, \leq))$.

The following theorem is a slight adaption of Theorem 3 from [5] and can be proved analogously.

▶ **Theorem 4.** *Every* $(\text{AC}^i, \log^i n)$*-maintainable, almost domain-independent query is in* DynFO$(+, \times)$.

The Role of the Domain and Arithmetic. In order to focus on the study of changes of non-constant size, we choose a simplified approach and include arithmetic in our setting. We state our results for DynFO(+, ×) and according classes to make it clear that we assume the presence of a linear order, addition and multiplication relation on the whole domain at all times.[2]

We shortly discuss the consequences of not assuming built-in arithmetic on our results. For single tuple changes, the presence of built-in arithmetic essentially gives no advantage.

▶ **Proposition 5** ([4, Theorem 4], formulation from [5, Proposition 2]). *If a query* $\mathcal{Q} \in$ DynFO(+, ×) *under single-tuple changes is almost domain-independent, then also* $\mathcal{Q} \in$ DynFO.

This result relies on the fact that one can maintain a linear order and arithmetic on the activated domain in DynFO under single-tuple changes [9], that is, on all elements that were in the active domain at some point of time. Under larger changes this is a priori not possible, as then one has to express in FO a linear order and arithmetic on the elements that enter the active domain.

An alternate approach to assuming the presence of built-in arithmetic is to demand that changes *provide* additional information on the changed elements, for example, that they provide a linear order and arithmetic on the domain of the change. Using this approach, our results can be stated in terms of DynFO and DynFO+Maj with the sole modification that sizes of changes are given relative to the size of the activated domain instead of with respect to the size of the whole domain. In this fashion our results also translate to the setting of first-order incremental evaluation systems of Dong, Su, and Topor [8], where the domain can grow and shrink.

4 Reachability under Multiple Changes

In this section we prove that Reachability can be maintained under multiple changes.

▶ **Theorem 1.** *Reachability can be maintained in* DynFO(+, ×) *under changes that affect* $\mathcal{O}(\frac{\log n}{\log \log n})$ *nodes of a graph, where* n *is the number of nodes of the graph.*

The approach is to use the well-known fact that Reachability can be reduced to the computation of the inverse of a matrix, and to invoke the Sherman-Morrison-Woodbury identity (cf. [12]) to update the inverse. This identity essentially reduces the update of inverses after a change affecting k nodes to the computation of an inverse of a $k \times k$ matrix.

The challenge is to define the updates in FO(+, ×). The key ingredients here are to compute inverses with respect to many primes, and throw away primes for which the inverse does not exist. As, by Theorem 4, it suffices to maintain the inverse for $\log^c n$ many steps for some c to be fixed later (see proof of Theorem 6), some primes remain valid if one starts from sufficiently – but polynomially – many primes. We show that the inverse of $k \times k$ matrices over $\mathbb{Z}_p$ can be defined in FO(+, ×) for $k = \frac{\log n}{\log \log n}$.

Theorem 1 in particular generalizes the result that Reachability can be maintained under single edge changes [4]; our proof is an alternative to the proof presented in the latter work.

[2] Different assumptions have been made in the literature. In [18], Patnaik and Immerman assume only a linear order to be present, while full arithmetic is assumed in [19]. Etessami observed that arithmetic can be built up dynamically, and therefore subsequent work usually assumed initially empty auxiliary relations, see e.g. [4, 5]. In the setting of first-order incremental evaluation systems usually no arithmetic is assumed to be present [8].

In [4], maintenance of Reachability is reduced to the question whether a matrix A has full rank, and it was shown that the rank can be maintained by storing and updating an invertible matrix B and a matrix D from which the rank can be easily extracted, such that $B \cdot A = D$.

4.1 Reachability and Matrix Inverses

There is a path from s to t in a graph G of size n with adjacency matrix A_G if and only if the s-t-entry of the matrix $(nI - A_G)^{-1}$ is non-zero. This follows from the equation $(nI - A_G)^{-1} = \frac{1}{n}\sum_{i=0}^{\infty}(\frac{1}{n}A_G)^i$ and the fact that A_G^i counts the number of paths from s to t of length i. Notice that $A \stackrel{\text{def}}{=} nI - A_G$ is invertible as matrix over $\mathbb{Q}$ for every adjacency matrix A_G since it is strictly diagonally dominant [15, Theorem 6.1.10].

When applying a change ΔG to G that affects k nodes, the adjacency matrix of G is updated by adding a suitable change matrix ΔA with at most k non-zero rows and columns to A. Thus Theorem 1 follows from the following proposition[3].

▶ **Theorem 6.** *When $A \in \mathbb{Z}^{n\times n}$ takes values polynomial in n and is assumed to stay invertible over $\mathbb{Q}$, then non-zeroness of entries of $A^{-1} \in \mathbb{Q}^{n\times n}$ can be maintained in* DynFO$(+,\times)$ *under changes that affect $\mathcal{O}(\frac{\log n}{\log\log n})$ rows and columns.*

Each change affecting $\mathcal{O}(\frac{\log n}{\log\log n})$ rows and columns can be partitioned into constantly many changes that affect $k \stackrel{\text{def}}{=} \frac{\log n}{\log\log n}$ rows and columns. We therefore concentrate on such changes in the following.

The change matrix ΔA for a change affecting k rows and columns has at most k non-zero rows and columns and can therefore be decomposed into a product UBV of suitable matrices U, B, and V, where U, B, and V have dimensions $n \times k$, $k \times k$, and $k \times n$, respectively.

▶ **Lemma 7.** *Fix a ring R. Suppose $M \in R^{n\times n}$ with non-zero rows $r_{i_1}, \dots, r_{i_k}$ and columns $c_{j_1}, \dots, c_{j_k}$. Then $M = UBV$ with $U \in R^{n\times k}, B \in R^{k\times k}$, and $V \in R^{k\times n}$ where*

1. *B is obtained from M by removing all-zero rows and columns.*
2. $U = \begin{pmatrix} \bar{u}_1 \\ \vdots \\ \bar{u}_n \end{pmatrix}$ *where* $\bar{u}_i = \begin{cases} \bar{0}^T & \text{if } i \notin \{i_1, \dots, i_k\} \\ \bar{e}_m^T & \text{if } i = i_m \end{cases}$
3. $V = \begin{pmatrix} \bar{v}_1, \dots, v_n \end{pmatrix}$ *where* $\bar{v}_j = \begin{cases} \bar{0} & \text{if } j \notin \{j_1, \dots, j_k\} \\ \bar{e}_m & \text{if } j = j_m \end{cases}$

Here, $\bar{e}_m$ denotes the m-th unit vector.

By the Sherman-Morrison-Woodbury identity (cf. [12]), the updated inverse can therefore be written as

$$(A + \Delta A)^{-1} = (A + UBV)^{-1} = A^{-1} - A^{-1}U(I + BVA^{-1}U)^{-1}BVA^{-1} \qquad (\star)$$

The inverse of a matrix in $\mathbb{Z}^{n\times n}$ with entries that are polynomial in n is a matrix in $\mathbb{Q}^{n\times n}$ with entries $\frac{a}{b}$ that may involve numbers exponential in n. In particular computations cannot be performed in FO$(+,\times)$ directly. For this reason all computations will be done modulo many primes, and non-zeroness of entries of A^{-1} is extracted from these values.

[3] Due to lack of space some details are hidden here. The described reduction maps the empty graph to the matrix whose diagonal entries are n. Values of the inverse for this matrix cannot be determined in FO, and thus one does not immediately get the desired result for Reachability. This issue can be circumvented by mapping to matrices with only some non-zero entries on the diagonal, and studying the inverse of the matrices induced by non-zero diagonal entries.

Let us first see how to update $(A+\Delta A)^{-1}$ modulo a prime p under the assumption that both $A \pmod p$ and $A+\Delta A \pmod p$ are invertible. Observe that $(I+BVA^{-1}U)^{-1}$ is a $k \times k$ matrix and therefore an essential prerequisite to compute $(A+\Delta A)^{-1} \pmod p$ is to be able to define the inverse of such small matrices. That this is possible follows from the following lemma and the fact that $[D^{-1}]_{ij} = (-1)^{i+j}\frac{\det D_{ji}}{\det D}$ for invertible $D \in \mathbb{Z}_p^{k\times k}$. Here $[C]_{ij}$ denotes the ij-th entry of a matrix C and C_{ji} denotes the submatrix obtained by removing the j-th row and the i-th column.

▶ **Theorem 8.** *Fix a domain of size n and a prime $p \in \mathcal{O}(n^c)$. The value of the determinant of a matrix $A \in \mathbb{Z}_p^{k\times k}$ for $k = \frac{\log n}{\log\log n}$ can be defined in* $\mathrm{FO}(+,\times)$.

The technical proof of this theorem is deferred until the next Subsection 4.2.

That $(A+\Delta A)^{-1} \pmod p$ can defined in $\mathrm{FO}(+,\times)$ using Equation $(\star)$ now is a consequence of a straightforward analysis of the involved matrix operations.

▶ **Proposition 9.** *Fix a domain of size n and a prime $p \in \mathcal{O}(n^c)$. Given the inverse of a matrix $A \in \mathbb{Z}_p^{n\times n}$ and a matrix $\Delta A \in \mathbb{Z}_p^{n\times n}$ with at most $k = \frac{\log n}{\log\log n}$ non-zero rows and columns, one can determine whether $A+\Delta A$ is invertible in* $\mathrm{FO}(+,\times)$ *and, if so, the inverse can be defined.*

Proof. A decomposition of the matrix ΔA into UBV with $U \in \mathbb{Z}_p^{n\times k}, B \in \mathbb{Z}_p^{k\times k}$, and $V \in \mathbb{Z}_p^{k\times n}$ can be defined in $\mathrm{FO}(+,\times)$ using the characterization from Lemma 7. A simple analysis of the right hand side of Equation $(\star)$ – taking the dimensions of U, V, and B into account – yields that $VA^{-1}U$ and therefore $(I+BVA^{-1}U)^{-1}B$ are $k \times k$ matrices. Furthermore, $U(I+BVA^{-1}U)^{-1}BV$ is an $n \times n$ matrix that has at most k non-zero rows and columns.

The only obstacle to invertibility is that the inverse of $D \stackrel{\text{def}}{=} I+BVA^{-1}U$ may not exist in $\mathbb{Z}_p$. This is the case if and only if $\det(D) \equiv 0 \pmod p$ which can be tested using Theorem 8. If D is invertible, then its inverse can be defined by invoking Theorem 8 twice and using $[D]_{ij} = (-1)^{i+j}\frac{\det D_{ji}}{\det D}$.

Finally, if one knows how to compute $(I+BVA^{-1}U)^{-1}$, each entry in $A^{-1}U(I+BVA^{-1}U)^{-1}BV$ can be defined by adding k products of two numbers, and similarly for $(A^{-1}U(I+BVA^{-1}U)^{-1}BV)A^{-1}$. This can be done in $\mathrm{FO}(+,\times)$ due to Lemma 3. ◀

It remains to show how to maintain non-zeroness of entries of $(A+\Delta A)^{-1} \in \mathbb{Q}^{n\times n}$. Essentially a dynamic program can maintain a Chinese remainder representation of $(A+\Delta A)^{-1}$ and extract whether an entry is non-zero from this representation. An obstacle is that whenever $(I+BVA^{-1}U)^{-1} \pmod p$ does not exist for a prime p during the update process, then this prime p becomes *invalid* for the rest of the computation. The idea to circumvent this is simple: with each change, only a small number of primes become invalid. However, since the determinant can be computed in NC^2 (cf. [3]), using Theorem 4 we only need to be able to maintain a correct result for $\log^2 n$ many steps. Thus starting from sufficiently many primes will guarantee that enough primes are still valid after $\log^2 n$ steps.

We make these numbers more precise in the following.

Proof (of Theorem 6). By Theorem 4 and since values of the inverse of a matrix are almost domain-independent, it suffices to exhibit a dynamic program[4] that maintains non-zeroness of entries of A^{-1} for $\log^2 n$ changes of size $\frac{\log n}{\log\log n}$. The dynamic program maintains A^{-1}

[4] Actually we only describe a program that works correctly for sufficiently large n. However, small n can be easily dealt with separately.

(mod p) for each of the first $2n^3$ many primes p, which, by the Prime Number Theorem, can be found among the first n^4 numbers. Denote by P the set of the first $2n^3$ primes. The NC^2 initialization procedure computes $A^{-1} \pmod p$ for each prime in P. The update procedure for a change ΔA is simple:

(1) For each prime $p \in P$:
  (a) If $(A + \Delta A)^{-1} \pmod p$ is not invertible then remove p from P.
  (b) If $(A + \Delta A)^{-1} \pmod p$ is invertible then update $(A + \Delta A)^{-1} \pmod p$.

(2) Declare $[(A + \Delta A)^{-1}]_{st} \neq 0$ if there is a prime $p \in P$ with $[(A + \Delta A)^{-1}]_{st} \not\equiv 0 \pmod p$.

The Steps 1a and 1b can be performed in $\mathrm{FO}(+, \times)$ due to Proposition 9.

It remains to argue that the result from Step 2 is correct. Observe that the values of entries of A are at most n at all times, and therefore $\det(A) \leq n! n^n \leq 2^{n^2}$ for large enough n. Thus, since $\det(A) \neq 0$ over $\mathbb{Z}$ by assumption, there are at most n^2 primes p such that $\det(A) \equiv 0 \pmod p$, for all A reached after a sequence of changes.

In particular, $(A + \Delta A)^{-1} \pmod p$ is not invertible – equivalently, $(I + BVA^{-1}U)^{-1} \pmod p$ does not exist – for at most n^2 primes p. Hence, each time Step 1 is executed, at most n^2 primes are declared invalid and removed from P. All in all this step is executed at most $\log^2 n$ times, and therefore not more than n^3 primes are removed from P. Thus for the remaining n^3 valid primes, the inverses $(A + \Delta A)^{-1} \pmod p$ are computed correctly.

Each entry of $(A + \Delta A)^{-1}$ is, again, bounded by 2^{n^2}, so if $[(A + \Delta A)^{-1}]_{st} \neq 0$ there are at most n^2 primes $p \in P$ with $[(A + \Delta A)^{-1}]_{st} \equiv 0 \pmod p$. So, the result declared in Step 2 is correct. ◀

4.2 Defining the Determinant of Small Matrices

In this subsection we prove Theorem 8. The symbolic determinant of a $k \in \mathcal{O}(\frac{\log}{\log \log n})$ sized matrix is a sum of $k! \in n^{\mathcal{O}(1)}$ monomials and therefore cannot be naïvely defined in $\mathrm{FO}(+, \times)$. Here we use the fact that $\mathrm{FO}(+, \times)$ can easily convert $\log n$ bit numbers into their Chinese remainder presentation and back, and show how the determinant can be computed modulo $\log \log n$ bit primes.

It is easy to verify whether the value of a determinant modulo a $\mathcal{O}(\log \log n)$ bit prime is zero in $\mathrm{FO}(+, \times)$ by guessing a linear combination witnessing that the rank is less than full. We aim for a characterization that allows to reduce the verification of determinant values to such zeroness tests. To this end we use the self-reducibility and multilinearity of determinants. Assume $[A]_{11} \neq 0$ and that the determinant of A_{11} is also non-zero. Then the determinant can be written as $[A]_{11} \cdot d + r$ for some d and r. By finding an a such that the determinant is zero when $[A]_{11}$ is replaced by a in A we gain $r = -ad$. Repeating this step recursively for d – which is the determinant of a smaller matrix – one obtains a procedure for determining the value of the determinant that can be parallelized.

The following lemma is a preparation for deriving the characterization. We denote by A_i the matrix obtained from a matrix A by removing all rows and columns larger than i.

▶ **Lemma 10.** *Suppose $B = (\bar{b}_1, \ldots, \bar{b}_k) \in \mathbb{F}^{k \times k}$ is a non-singular matrix over a field $\mathbb{F}$. Then there is a permutation $\pi : [k] \to [k]$ such that for $A \stackrel{def}{=} (b_{\pi(1)}, \ldots, b_{\pi(k)})$:*

$$[A]_{ii} \neq 0 \text{ and } \det(A_i) \neq 0 \text{ for all } i \in [k]$$

The following proposition characterizes the determinant of a matrix. We will see that this characterization allows for parallel computation of the determinant of small matrices.

▶ **Proposition 11.** *Suppose* $A = (a_{ij})_{1 \leq i,j \leq k} \in \mathbb{F}^{k \times k}$ *is a matrix over a field* $\mathbb{F}$ *such that* $a_{ii} \neq 0$ *and* $\det(A_i) \neq 0$ *for all* $i \in [k]$*. Let* A_i^b *be the matrix obtained from* A_i *by replacing* a_{ii} *by* b *for some* $b \in \mathbb{F}$*. Then there are unique* $b_2, \ldots, b_k \in \mathbb{F}$ *and* $d_1, \ldots, d_k \in \mathbb{F}$ *such that*

1. $d_1 = a_{11}$,
2. $d_i = (a_{ii} - b_i)d_{i-1}$, *and*
3. $\det(A_i^b) = 0$

for $2 \leq i \leq k$*. Furthermore, it holds that* $d_i = \det(A_i)$.

Finally we show that the characterization from the previous proposition can be used to define the determinant of small matrices in $\mathrm{FO}(+, \times)$.

Proof (of Theorem 8). Suppose $A \in \mathbb{Z}_p^{k \times k}$ is a matrix with $p \in \mathcal{O}(n^c)$ and $k = \frac{\log n}{\log \log n}$. The idea is to define $\det(A) \pmod p$ in Chinese remainder representation for primes $q_1, \ldots, q_m$. A simple calculation shows that $m \in \mathcal{O}(\log n)$ primes each of $\mathcal{O}(\log \log n)$ bits suffice. The Chinese remainder representation can be defined from A and the value $\det(A) \pmod p$ can be recovered from the values $\det(A) \pmod{q_1}, \ldots, \det(A) \pmod{q_m}$ in $\mathrm{FO}(+, \times)$ due to Lemma 3. Thus let us show how to define $\det(A) \pmod q$ for a prime q of $\mathcal{O}(\log \log n)$ bits.

The idea is to first test whether the determinant is zero. If not, the fact that it is not zero is used to define the determinant using Proposition 11.

If $A \pmod q$ is singular then there exists a non-trivial linear combination of the columns that yields the all zero vector. Such a linear combination is determined by specifying one $\mathcal{O}(\log \log n)$ bit number for each of the k columns. It can thus be encoded in $\mathcal{O}(\log n)$ bits, and therefore existentially quantified by a first-order formula. Such a "guess" can be decoded (i.e., the k numbers of $\mathcal{O}(\log \log n)$ length can be extracted) in $\mathrm{FO}(+, \times)$, see Section 2. Checking if a guessed linear combination is zero requires to sum k small numbers and is hence in $\mathrm{FO}(+, \times)$ due to Lemma 3.

Now, for defining the determinant $\det(A) \pmod q$ when $A \pmod q$ is non-singular, a formula can guess a permutation π of $[k]$ and verify that it satisfies the conditions from Lemma 10. Note that such a permutation can be represented as a sequence of k pairs of numbers of $\log \log n$ bits each, and hence be stored in $\mathcal{O}(\log n)$ bits. The verification of the conditions from Lemma 10 requires the zero-test for determinants explained above. After fixing π, the values $b_2, \ldots, b_k$ as well as $d_1, \ldots, d_k$ from Proposition 11 can be guessed and verified. Again, these numbers can be stored in $\mathcal{O}(\log n)$ bits. For verifying the conditions from Proposition 11 on the determinants of A_i^b, the zero-test for determinants is used. ◀

5 Distances under Multiple Changes

In this section we extend the techniques from the previous section to show how distances can be maintained under changes that affect polylogarithmically many nodes with first-order updates that may use majority quantifiers. Afterwards we discuss how the techniques extend to other dynamic complexity classes.

▶ **Theorem 2.** *Reachability and Distance can be maintained in* DynFO+Maj$(+, \times)$ *under changes that affect* $\mathcal{O}(\log^c n)$ *nodes of a graph, where* $c \in \mathbb{N}$ *is fixed and* n *is the number of nodes of the graph.*

The idea is to use generating functions for counting the number of paths of each length, following Hesse [13]. Fix a graph G with adjacency matrix $A_G \in \mathbb{Z}^{n \times n}$ and a formal variable x. Then $D \stackrel{\text{def}}{=} \sum_{i=0}^{\infty} (xA_G)^i$ is a matrix of formal power series from $\mathbb{Z}[[x]]$ such that if $[D]_{st} = \sum_{i=0}^{\infty} c_i x^i$ then c_i is the number of paths from s to t of length i. In particular, the distance between s and t is the smallest i such that c_i is non-zero. Note that if such an i exists, then $i < n$.

Similarly to the corresponding matrix from the previous section, the matrix D is invertible over $\mathbb{Z}[[x]]$ and can be written as $(I - xA_G)^{-1}$ (cf. [10, Example 3.6.1]). The maintenance of distances thus reduces to maintaining for a matrix $A \in \mathbb{Z}[[x]]$, for each entry (s,t), the smallest $i < n$ such that the ith coefficient is non-zero.

▶ **Theorem 12.** *Suppose $A \in \mathbb{Z}[[x]]^{n\times n}$ stays invertible over $\mathbb{Z}[[x]]$. For all $s,t \in [n]$ one can maintain the smallest $i < n$ such that the ith coefficient of the st-entry of A^{-1} is non-zero in* DynFO+Maj$(+,\times)$ *under changes that affect $\mathcal{O}(\log^c n)$ nodes, for fixed $c \in \mathbb{N}$.*

The idea is the same as for Reachability. When updating A to $A + \Delta A$ then one can decompose the change matrix ΔA into UBV for suitable matrices U, B, and V, and apply the Sherman-Morrison-Woodbury identity ($\star$), this time over the field of fractions $\mathbb{Z}((x))$.

Of course computing with inherently infinite formal power series is not possible in DynFO+Maj$(+,\times)$. However, as stated in Theorem 12, in the end we are only interested in the first $i < n$ coefficients of power series. We therefore show that it suffices to truncate all occurring power series at the n-th term and use FO+Maj$(+,\times)$'s ability to define iterated sums and products of polynomials [14].

Formally, we have to show that no precision for the first $i < n$ coefficients is lost when computing with truncated power series. This motivates the following definition. A formal power series $g(x) = \sum_i c_i x^i \in \mathbb{Z}[[x]]$ is an *m-approximation* of a formal power series $h(x) = \sum_i d_i x^i \in \mathbb{Z}[[x]]$, denoted by $g(x) \approx_m h(x)$, if $c_i = d_i$ for all $i \leq m$. This notion naturally extends to matrices over $\mathbb{Z}[[x]]$: a matrix $A \in \mathbb{Z}[[x]]^{\ell\times k}$ is an m-approximation of a matrix $B \in \mathbb{Z}[[x]]^{\ell\times k}$ if each entry of A is an m-approximation of the corresponding entry of B. The notion of m-approximation is preserved under all arithmetic operations that will be relevant.

▶ **Lemma 13.** *Fix an $m \in \mathbb{N}$.*

1. *Suppose $g(x), g'(x), h(x), h'(x) \in \mathbb{Z}[[x]]$ with $g(x) \approx_m g'(x)$ and $h(x) \approx_m h'(x)$. Then*
   - **(i)** *$g(x) + h(x) \approx_m g'(x) + h'(x)$,*
   - **(ii)** *$g(x)h(x) \approx_m g'(x)h'(x)$, and*
   - **(iii)** *$\frac{1}{g(x)} \approx_m \frac{1}{g'(x)}$ whenever $g(x)$ and $g'(x)$ are normalized.*
2. *Suppose $A, A', B, B' \in \mathbb{Z}[[x]]^{n\times n}$ with $A \approx_m A'$ and $B \approx_m B'$. Then*
   - **(i)** *$A + B \approx_m A' + B'$,*
   - **(ii)** *$AB \approx_m A'B'$,*
   - **(iii)** *If A is invertible over $\mathbb{Z}[[x]]$ then so is A', and $A^{-1} \approx_m A'^{-1}$.*

Here, a formal power series $\sum_i c_i x^i \in \mathbb{Z}[[x]]$ is *normalized* if $c_0 = 1$.

An approximation of the inverse of a matrix $A \in \mathbb{Z}[[x]]^{n\times n}$ can be updated using the Sherman-Morrison-Woodbury identity.

▶ **Proposition 14.** *Suppose $A \in \mathbb{Z}[[x]]^{n\times n}$ is invertible over $\mathbb{Z}[[x]]$, and $C \in \mathbb{Z}[[x]]^{n\times n}$ is an m-approximation of A^{-1}. If $A + \Delta A$ is invertible over $\mathbb{Z}[[x]]$ and ΔA can be written as UBV with $U \in \mathbb{Z}[[x]]^{n\times k}, B \in \mathbb{Z}[[x]]^{k\times k}$, and $V \in \mathbb{Z}[[x]]^{k\times n}$, then*

$$(A + \Delta A)^{-1} \approx_m C - CU(I + BVCU)^{-1}BVC$$

Proof. This follows immediately from the Sherman-Morrison-Woodbury identity $(A + UBV)^{-1} = A^{-1} - A^{-1}U(I + BVA^{-1}U)^{-1}BVA^{-1}$ and Lemma 13. ◀

As already discussed in Section 4, the Sherman-Morrison-Woodbury identity involves inverting $k \times k$ matrices, which reduces to computing the determinant of such matrices. We show that this is possible in FO+Maj for $k \times k$ matrices of polynomials for $k \in \mathcal{O}(\log^c n)$.

▶ **Lemma 15.** *Fix a domain of size n and $c \in \mathbb{N}$. The determinant of a matrix $A \in \mathbb{Z}[x]^{k\times k}$, with entries of degree polynomial in n, can be defined in* FO+Maj$(+,\times)$ *for $k \in \mathcal{O}(\log^c n)$.*

Proof. We show that the value can be computed in uniform TC^0, which is as powerful as FO+Maj$(+,\times)$ [2].

Computing the determinant of an $k \times k$ matrix is equivalent to computing the iterated matrix product of k matrices of dimension at most $(k+1)\times(k+1)$ [3], and this reduction is a uniform TC^0-reduction as can be seen implicitly in [17, p. 482]. Thus the lemma statement follows from the fact that iterated products of matrices $A_1, \ldots, A_k \in \mathbb{Z}[x]^{k\times k}$ with $k \in \mathcal{O}(\log^c n)$ can be computed in uniform TC^0, which can be proven like in [1, p. 69]. ◀

Proof (of Theorem 12). The dynamic program maintains an n-approximation $C \in \mathbb{Z}[x]^{n\times n}$ of A^{-1} that truncates A^{-1} at degree n. When A is updated to $A + \Delta A$ then:

1. ΔA is decomposed into suitable $U \in \mathbb{Z}[x]^{n\times k}$, $B \in \mathbb{Z}[x]^{k\times k}$, and $V \in \mathbb{Z}[x]^{k\times n}$;
2. C is updated via $C' \stackrel{\text{def}}{=} C - CU(I + BVCU)^{-1}BVC$;
3. All entries of C' are truncated at degree n.

The steps can be defined in FO+Maj$(+,\times)$ due to Lemma 7, Lemma 15, and the fact that iterated addition and multiplication of polynomials can be defined in FO+Maj$(+,\times)$, see [14]. The maintained matrix C is indeed an n-approximation of A^{-1} due to Proposition 14. ◀

From the proof of Theorem 12 it is clear that the main obstacle towards maintaining distances for changes that affect a larger set of nodes is to compute determinants of larger matrices. Since distances can be computed in NL, only classes below NL are interesting from a dynamic perspective. As an example we state a result for the circuit class NC^1.

▶ **Corollary 16.** *Reachability and Distance can be maintained in* DynNC^1 *under changes that affect $\mathcal{O}(2^{\sqrt{\log n/\log^* n}})$ nodes.*

Here $\log^* n$ denotes the smallest number i such that i-fold application of log yields a number smaller than 1.

6 Conclusion

For us it came as a surprise that Reachability can be maintained under changes of non-constant size, without any structural restrictions. In contrast, the dynamic program for Reachability from [4] can only deal with changing $\log n$ many outgoing edges of single nodes (or, symmetrically, $\log n$ many incoming edges; a combination is not possible).

It would be interesting to improve our results for Dyn FO$(+,\times)$ to changes of size $\mathcal{O}(\log n)$. The obstacle is the computation of determinants of matrices of this size, which we can only do for $\mathcal{O}(\frac{\log n}{\log\log n})$ size matrices. Yet in principle our approach can deal with certain changes that affect more nodes: the matrices U and V in the Sherman-Morrison-Woodbury identity can be chosen differently, as long as all computations involve only adding $\mathcal{O}(\log n)$ numbers.

One of the big remaining open questions in dynamic complexity is whether distances are in Dyn FO. Our approach sheds some light. It can be adapted so as to maintain information within Dyn FO$(+,\times)$ from which shortest distances can be extracted in FO+Maj$(+,\times)$.

▶ **Theorem 17.** *Distances can be defined by a* FO+Maj$(+,\times)$ *query from auxiliary relations that can be maintained in* Dyn FO$(+,\times)$ *under changes that affect $\mathcal{O}(\frac{\log n}{\log\log n})$ nodes.*

References

1 Manindra Agrawal and V Vinay. Arithmetic circuits: A chasm at depth four. In *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 67–75. IEEE, 2008.

2 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *J. Comput. Syst. Sci.*, 41(3):274–306, 1990. doi:10.1016/0022-0000(90)90022-D.

3 Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1-3):2–21, 1985. doi:10.1016/S0019-9958(85)80041-3.

4 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2015. doi:10.1007/978-3-662-47666-6.

5 Samir Datta, Anish Mukherjee, Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. A strategy for dynamic programs: Start over and muddle through. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 98:1–98:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. URL: http://www.dagstuhl.de/dagpub/978-3-95977-041-5, doi:10.4230/LIPIcs.ICALP.2017.98.

6 Samir Datta, Anish Mukherjee, Nils Vortmeier, and Thomas Zeume. Reachability and distances under multiple changes. *CoRR*, abs/1804.08555, 2018. arXiv:1804.08555.

7 Guozhu Dong and Chaoyi Pang. Maintaining transitive closure in first order after node-set and edge-set deletions. *Inf. Process. Lett.*, 62(4):193–199, 1997. doi:10.1016/S0020-0190(97)00066-5.

8 Guozhu Dong, Jianwen Su, and Rodney W. Topor. Nonrecursive incremental evaluation of datalog queries. *Ann. Math. Artif. Intell.*, 14(2-4):187–223, 1995. doi:10.1007/BF01530820.

9 Kousha Etessami. Dynamic tree isomorphism via first-order updates. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 235–243, 1998. doi:10.1145/275487.275514.

10 Chris D. Godsil. *Algebraic combinatorics*. Chapman and Hall mathematics series. Chapman and Hall, 1993.

11 Erich Grädel and Sebastian Siebertz. Dynamic definability. In Alin Deutsch, editor, *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*, pages 236–248. ACM, 2012. URL: http://dl.acm.org/citation.cfm?id=2274576, doi:10.1145/2274576.2274601.

12 Harold V Henderson and Shayle R Searle. On deriving the inverse of a sum of matrices. *Siam Review*, 23(1):53–60, 1981.

13 William Hesse. The dynamic complexity of transitive closure is in $DynTC^0$. *Theor. Comput. Sci.*, 296(3):473–485, 2003. doi:10.1016/S0304-3975(02)00740-5.

14 William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65(4):695–716, 2002. doi:10.1016/S0022-0000(02)00025-9.

15 Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

16 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.

17 Meena Mahajan and V. Vinay. Determinant: Old algorithms, new insights. *SIAM J. Discrete Math.*, 12(4):474–490, 1999. doi:10.1137/S0895480198338827.

18 Sushant Patnaik and Neil Immerman. Dyn-fo: A parallel, dynamic complexity class. In Victor Vianu, editor, *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 24-26, 1994, Minneapolis, Minnesota, USA*, pages 210–221. ACM Press, 1994. URL: http://dl.acm.org/citation.cfm?id=182591, doi:10.1145/182591.182614.

19 Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997. doi:10.1006/jcss.1997.1520.

20 Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Dynamic complexity under definable changes. In Michael Benedikt and Giorgio Orsi, editors, *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, volume 68 of *LIPIcs*, pages 19:1–19:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. URL: http://www.dagstuhl.de/dagpub/978-3-95977-024-8, doi:10.4230/LIPIcs.ICDT.2017.19.

21 Thomas Schwentick and Thomas Zeume. Dynamic complexity: recent updates. *SIGLOG News*, 3(2):30–52, 2016. doi:10.1145/2948896.2948899.

When is Containment Decidable for Probabilistic Automata?

Laure Daviaud
University of Warwick, Coventry, UK
L.Daviaud@warwick.ac.uk

Marcin Jurdziński
University of Warwick, Coventry, UK
Marcin.Jurdzinski@warwick.ac.uk

Ranko Lazić[1]
University of Warwick, Coventry, UK
R.S.Lazic@warwick.ac.uk

Filip Mazowiecki[2]
Université de Bordeaux, Bordeaux, France
filip.mazowiecki@u-bordeaux.fr

Guillermo A. Pérez[3]
Université libre de Bruxelles, Brussels, Belgium
gperezme@ulb.ac.be
https://orcid.org/0000-0002-1200-4952

James Worrell[4]
University of Oxford, Oxford, UK
James.Worrell@cs.ox.ac.uk
https://orcid.org/0000-0001-8151-2443

Abstract

The containment problem for quantitative automata is the natural quantitative generalisation of the classical language inclusion problem for Boolean automata. We study it for probabilistic automata, where it is known to be undecidable in general. We restrict our study to the class of probabilistic automata with bounded ambiguity. There, we show decidability (subject to Schanuel's conjecture) when one of the automata is assumed to be unambiguous while the other one is allowed to be finitely ambiguous. Furthermore, we show that this is close to the most general decidable fragment of this problem by proving that it is already undecidable if one of the automata is allowed to be linearly ambiguous.

2012 ACM Subject Classification Theory of computation → Quantitative automata, Theory of computation → Probabilistic computation

Keywords and phrases Probabilistic automata, Containment, Emptiness, Ambiguity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.121

[1] R. Lazić has been supported by a Leverhulme Trust Research Fellowship RF-2017-579.
[2] F. Mazowiecki has been supported by the French National Research Agency (ANR) in the frame of the "Investments for the future" Programme IdEx Bordeaux (ANR-10-IDEX-03-02).
[3] G. A. Pérez has been supported by an F.R.S.-FNRS Aspirant fellowship and an FWA postdoc fellowship.
[4] J. Worrell has been supported by the EPSRC Fellowship EP/N008197/1.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 121; pp. 121:1–121:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version A full version of the paper is available at `https://arxiv.org/abs/1804.09077`.

Funding L. Daviaud, M. Jurdziński, and R. Lazić have been supported by the EPSRC grant EP/P020992/1.

Acknowledgements We thank Shaull Almagor for some helpful remarks.

1 Introduction

Probabilistic automata (PA) are a quantitative extension of classical Boolean automata that were first introduced by Rabin [20]. Non-deterministic choices are replaced by probabilities: each transition carries a rational number which gives its probability to be chosen amongst all the other transitions going out of the same state and labelled by the same letter. Then, instead of simply accepting or rejecting a word, such an automaton measures the probability of it being accepted.

PA can be seen as (blind) partially observable Markov decision processes [19]. The latter have numerous applications in the field of artificial intelligence [22, 11]. Further applications for PA include, amongst others, verification of probabilistic systems [23, 14, 5], reasoning about inexact hardware [18], quantum complexity theory [25], uncertainty in runtime modelling [9], as well as text and speech processing [17]. PA are very expressive, as witnessed by the mentioned applications, most natural verification-related decision problems for them are consequently undecidable. However, equivalence and minimisation do admit efficient algorithms [13].

Due to the aforementioned negative results, many sub-classes of probabilistic automata have been studied. These include hierarchical [7] and leaktight [2] automata; and more recently, bounded-ambiguity automata [8] (see [6] for a survey).

In this paper, we continue the study of the class of PA with bounded ambiguity. We focus on the containment problem: *given two automata $\mathcal{A}$ and $\mathcal{B}$, determine whether for all words w, the probability of it being accepted by $\mathcal{A}$ is at most the probability of it being accepted by $\mathcal{B}$.* The problem is known to be undecidable even for the subclass of automata with polynomial ambiguity, more specifically, already for automata with quadratic ambiguity [8].

Contributions. In this paper, we refine the undecidability result by extending it to the class of linearly ambiguous automata.

▶ **Theorem 1.** *The containment problem is undecidable for the class of linearly ambiguous probabilistic automata.*

The proof we provide gives in fact two stronger results. Firstly, the containment problem for linearly ambiguous PA is already undecidable if one of the two input automata is unambiguous. Secondly, and perhaps more importantly, the better-known emptiness problem (*given a probabilistic automaton, does there exist a word accepted with probability at least* $1/2$*?*) is also undecidable for the class of linearly ambiguous PA. This strictly refines the previous best known result [8].

This negative result motivates us to turn our attention to the class of finitely ambiguous PA. For this class, we prove that the containment problem is decidable, provided that one of the two input automata is unambiguous (and conditional on Schanuel's conjecture).

▶ **Theorem 2.** *If Schanuel's conjecture holds then the containment problem is decidable for the class of finitely ambiguous probabilistic automata, provided that at least one of the input automata is unambiguous.*

The intermediate problem, i.e., when both input PA are finitely ambiguous, remains open.

Organisation of the paper. In Section 2, we give the formal definition of probabilistic automata, the notion of ambiguity, and the problems under consideration. We also recall classical results that will be useful in the paper. In Section 3, we explain how to translate the containment problem into a problem about the existence of integral exponents for certain exponential inequalities. Using this formalism, we prove that the containment problem for $\mathcal{A}$ and $\mathcal{B}$, as stated above, is decidable if $\mathcal{A}$ is finitely ambiguous and $\mathcal{B}$ is unambiguous. In Section 4, we tackle the more challenging direction and prove that the containment problem is also decidable if $\mathcal{A}$ is unambiguous and $\mathcal{B}$ is finitely ambiguous. Finally, in Section 5, we prove that the containment problem is undecidable provided that one of the automata is linearly ambiguous.

2 Preliminaries

In this section, we define probabilistic automata and recall some classical results.

Notation. We use boldface lower-case letters, e.g., $\boldsymbol{a}, \boldsymbol{b}, \ldots$, to denote vectors and upper-case letters, e.g., $M, N, \ldots$, for matrices. For a vector $\boldsymbol{a}$, we write a_i for its i-th component, and $\boldsymbol{a}^\top$ for its transpose.

2.1 Probabilistic automata and ambiguity

For a finite set S, we say that a function $f : S \to \mathbb{Q}_{\geq 0}$ is a *distribution over* S if $\sum_{s \in S} f(s) \leq 1$. We write $\mathcal{D}(S)$ for the set of all distributions over S. We also say that a vector $\boldsymbol{d} = (d_1, d_2, \ldots, d_n) \in \mathbb{Q}^n_{\geq 0}$ of non-negative rationals is a distribution if $\sum_{i=1}^n d_i \leq 1$.

A *probabilistic automaton (PA)* $\mathcal{A}$ is a tuple $(\Sigma, Q, \delta, \iota, F)$, where:

- Σ is the finite alphabet,
- Q is the finite set of states,
- $\delta : Q \times \Sigma \to \mathcal{D}(Q)$ is the (probabilistic) transition function,
- $\iota \in \mathcal{D}(Q)$ is the initial distribution, and
- $F \subseteq Q$ is the set of final states.

We write $\delta(q, a, p)$ instead of $\delta(q, a)(p)$ for the *probability of moving from q to p reading a*. Consider the word $w = a_1 \ldots a_n \in \Sigma^*$. A *run ρ of $\mathcal{A}$ over $w = a_1 \ldots a_n$* is a sequence of transitions $(q_0, a_1, q_1), (q_1, a_2, q_2), \ldots, (q_{n-1}, a_n, q_n)$ where $\delta(q_{i-1}, a_i, q_i) > 0$ for all $1 \leq i \leq n$. It is an *accepting run* if $\iota(q_0) > 0$ and $q_n \in F$. The *probability of the run ρ* is $\Pr_{\mathcal{A}}(\rho) \stackrel{\text{def}}{=} \iota(q_0) \cdot \prod_{i=1}^n \delta(q_{i-1}, a_i, q_i)$.

The automaton $\mathcal{A}$ realizes a function $[\![\mathcal{A}]\!]$ mapping words over the alphabet Σ to values in $[0, 1]$. Formally, for all $w \in \Sigma^*$, we set: $[\![\mathcal{A}]\!](w) \stackrel{\text{def}}{=} \sum_{\rho \in \mathrm{Acc}_{\mathcal{A}}(w)} \Pr_{\mathcal{A}}(\rho)$ where $\mathrm{Acc}_{\mathcal{A}}(w)$ is the set of all accepting runs of $\mathcal{A}$ over w.

Ambiguity. The notion of ambiguity depends only on the structure of the underlying automaton (i.e., whether a probability is null or not, but not on its actual value). An automaton $\mathcal{A}$ is said to be *unambiguous* (resp. *k-ambiguous*) if for all words w, there is at most one accepting run (resp. k accepting runs) over w in $\mathcal{A}$. If an automaton is k-ambiguous for some k, then it is said to be *finitely ambiguous*. If there exists a polynomial P, such that

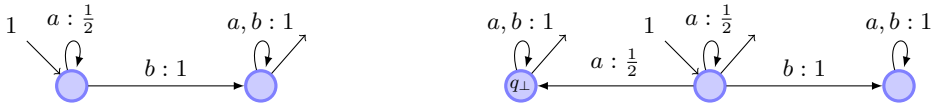


Figure 1 Two PA over the alphabet $\Sigma = \{a, b\}$ are depicted. On the left hand side, automaton $\mathcal{A}$ induces the function $a^n b\Sigma^* \mapsto \frac{1}{2^n}$ and $a^* \mapsto 0$. On the right hand side, the automaton $\overline{\mathcal{A}}$ induces the function $a^n b\Sigma^* \mapsto 1 - \frac{1}{2^n}$ and $a^* \mapsto 1$. Observe that $\mathcal{A}$ is unambiguous and $\overline{\mathcal{A}}$ is linearly ambiguous.

for every word w, the number of accepting runs of $\mathcal{A}$ on w is bounded by $P(|w|)$ (where $|w|$ is the length of w), then $\mathcal{A}$ is said to be *polynomially ambiguous*, and *linearly ambiguous* whenever the degree of P is at most 1.

It is well-known that if an automaton is not finitely ambiguous then it is at least linearly ambiguous (see, for example, the criterion in [24, Section 3]). The same paper shows that if an automaton is finitely ambiguous then it is k-ambiguous for k bounded exponentially in the number of states of that automaton.

We give two examples of PA and discuss their ambiguity in Figure 1. As usual, they are depicted as graphs. The initial distribution is denoted by ingoing arrows associated with their probability (when there is no such arrow, the initial probability is 0) and the final states are denoted by outgoing arrows.

2.2 Decision problems

In this work, we are interested in comparing the functions computed by PA. We write $[\![\mathcal{A}]\!] \leq [\![\mathcal{B}]\!]$ if "$\mathcal{A}$ is contained in $\mathcal{B}$", that is if $[\![\mathcal{A}]\!](w) \leq [\![\mathcal{B}]\!](w)$ for all $w \in \Sigma^*$; and we write $[\![\mathcal{A}]\!] < \frac{1}{2}$ if $[\![\mathcal{A}]\!](w) < \frac{1}{2}$ for all $w \in \Sigma^*$. We are interested in the following decision problems for PA.

- *Containment problem:* Given probabilistic automata $\mathcal{A}$ and $\mathcal{B}$, does $[\![\mathcal{A}]\!] \leq [\![\mathcal{B}]\!]$ hold?
- *Emptiness problem:* Given a probabilistic automaton $\mathcal{A}$, does $[\![\mathcal{A}]\!] < \frac{1}{2}$ hold?

We will argue that the containment and emptiness problems are both undecidable when considered for the class of linearly ambiguous automata (Section 5). The emptiness problem is known to be decidable for the class of finitely ambiguous automata [8]. We tackle here the more difficult containment problem (Sections 3 and 4).

2.3 Classical results

Weighted-sum automaton. For PA $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ over the same alphabet, and for a discrete distribution $\boldsymbol{d} = (d_1, d_2, \ldots, d_n)$, the *weighted sum* (of $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ with weights $\boldsymbol{d}$) is defined to be the disjoint union of the n automata with the initial distribution $\iota(q) \stackrel{\text{def}}{=} d_i \cdot \iota_i(q)$ if q is a state of $\mathcal{A}_i$, where ι_i is the initial distribution of $\mathcal{A}_i$. Note that if $\mathcal{B}$ is the weighted sum of $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ with weights $\boldsymbol{d}$ then it is also a probabilistic automaton and $[\![\mathcal{B}]\!] = \sum_{i=1}^n d_i \cdot [\![\mathcal{A}_i]\!]$.

Complement automaton. For a PA $\mathcal{A}$, we define its *complement automaton* $\overline{\mathcal{A}}$ in the following way. First, define the PA $\mathcal{A}'$ by modifying $\mathcal{A}$ as follows:

- add a new sink state $q_\perp$;
- obtain the transition function δ' from δ by adding transitions:
  - $\delta'(q_\perp, a, q_\perp) = 1$ for all $a \in \Sigma$,
  - $\delta'(q, a, q_\perp) = 1 - \sum_{r \in Q} \delta(q, a, r)$ for all $(q, a) \in Q \times \Sigma$;
- obtain the initial distribution ι' from ι by adding $\iota'(q_\perp) = 1 - \sum_{q \in Q} \iota(q)$.

Observe that $[\![\mathcal{A}']\!] = [\![\mathcal{A}]\!]$, that $\sum_{r\in Q}\delta'(q,a,r) = 1$ for all $(q,a) \in Q \times \Sigma$, and that $\sum_{q\in Q}\iota'(q) = 1$. We obtain $\overline{\mathcal{A}}$ from $\mathcal{A}'$ by swapping its final and non-final states. As expected, it is the case that $[\![\overline{\mathcal{A}}]\!] = 1 - [\![\mathcal{A}]\!]$.

▶ Remark (Preserving ambiguity). The ambiguity of a weighted-sum automaton is the sum of the ambiguities of the individual automata, and the ambiguity of a complement automaton may be larger than the ambiguity of the original one (see Figure 1).

3 Decidability of the case finitely ambiguous vs. unambiguous

Our aim is to decide whether $[\![\mathcal{A}]\!] \leq [\![\mathcal{B}]\!]$. We first give a translation of the problem into a problem about the existence of integral exponents for certain exponential inequalities.

Notation. In the rest of the paper, we write $\exp(x)$ to denote the exponential function $x \mapsto e^x$, and $\log(y)$ for the natural logarithm function $y \mapsto \log_e(y)$. For a real number x and a positive real number y, we write y^x for $\exp(x\log(y))$.

3.1 Translating the containment problem into exponential inequalities

We are going to translate the negation of the containment problem: Given two finitely ambiguous PA $\mathcal{A}$ and $\mathcal{B}$, does there exist a word w, such that $[\![\mathcal{A}]\!](w) > [\![\mathcal{B}]\!](w)$? Consider two positive integers k and n, and vectors $\boldsymbol{p} \in \mathbb{Q}^k_{>0}$ and $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_k \in \mathbb{Q}^n_{>0}$. We denote by $S(\boldsymbol{p}, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_k) : \mathbb{N}^n \to \mathbb{R}$ the function associating a vector $\boldsymbol{x} \in \mathbb{N}^n$ to $\sum_{i=1}^k p_i q_{i,1}^{x_1} \cdots q_{i,n}^{x_n}$, where $q_{i,j}$ is the j-th component of vector $\boldsymbol{q}_i$.

▶ **Proposition 3.** *Given a k-ambiguous automaton $\mathcal{A}$ and an ℓ-ambiguous automaton $\mathcal{B}$, one can compute a positive integer n and a finite set Δ of tuples $(\boldsymbol{p}, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_{k'}, \boldsymbol{r}, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{\ell'})$ of vectors $\boldsymbol{p} \in \mathbb{Q}^{k'}_{>0}, \boldsymbol{r} \in \mathbb{Q}^{\ell'}_{>0}$, for some $k' \leq k$ and $\ell' \leq \ell$; and $\boldsymbol{q}_i \in \mathbb{Q}^n_{>0}, \boldsymbol{s}_j \in \mathbb{Q}^n_{>0}$, for all i and j; such that the following two conditions are equivalent:*

- *there exists $w \in \Sigma^*$ such that $[\![\mathcal{A}]\!](w) > [\![\mathcal{B}]\!](w)$,*
- *there exist $(\boldsymbol{p}, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_{k'}, \boldsymbol{r}, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{\ell'}) \in \Delta$ and $\boldsymbol{x} \in \mathbb{N}^n$ such that*

$$S(\boldsymbol{p}, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_{k'})(\boldsymbol{x}) > S(\boldsymbol{r}, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{\ell'})(\boldsymbol{x}).$$

It thus follows that to prove Theorem 2, it suffices to show decidability of the second item of Proposition 3 for a given element of Δ in the cases where either k or ℓ are equal to 1. To prove Proposition 3, the idea is to decompose every run into a short path and simple cycles using the well-known simple-cycle decomposition (see, e.g., [21]). We do this simultaneously for all runs in $\mathcal{A}$ and $\mathcal{B}$. The vectors $\boldsymbol{p}$ and $\boldsymbol{r}$ then correspond to probabilities of simple paths in every run; $\boldsymbol{q}_i$ (resp. $\boldsymbol{s}_j$), to all simple cycles in the i-th run in $\mathcal{A}$ (resp. j-th run in $\mathcal{B}$). Finally, $\boldsymbol{x}$ specifies how many times each simple cycle occurs in the decomposition.

▶ **Example 4.** Consider the following instance of the problem, where $k = n = 2$, $\ell = 1$, and p is a fixed rational number $0 \leq p \leq 1$: Do there exist $x, y \in \mathbb{N}$ such that $p \cdot \left(\frac{1}{12}\right)^x \cdot \left(\frac{1}{2}\right)^y + (1-p) \cdot \left(\frac{1}{3}\right)^x \cdot \left(\frac{1}{18}\right)^y < \left(\frac{1}{6}\right)^x \cdot \left(\frac{1}{6}\right)^y$. This can be rewritten as

$$p \cdot \left(\frac{1}{2}\right)^x \cdot 3^y + (1-p) \cdot 2^x \cdot \left(\frac{1}{3}\right)^y < 1$$

or equivalently, using the exponential function, as follows

$$\exp(\log(p) - x\log(2) + y\log(3)) + \exp(\log(1-p) + x\log(2) - y\log(3)) < 1.$$

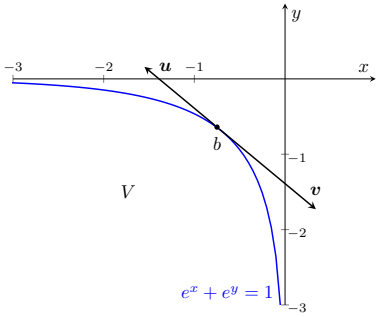


Figure 2 The set V is bounded by the plot $e^x + e^y = 1$ and the point b is on that plot.

Consider the set $V = \{(x, y) \in \mathbb{R}^2 \mid e^x + e^y < 1\}$ and denote by b the point $(\log(p), \log(1-p))$. Let $\boldsymbol{u} = (-\log(2), \log(2))$ and $\boldsymbol{v} = (\log(3), -\log(3))$ be two vectors. See Figure 2 for a geometric representation. The question is now: do there exist $x, y \in \mathbb{N}$ such that $b + x\boldsymbol{u} + y\boldsymbol{v} \in V$. We will show that the answer is yes if and only if $p \neq \frac{1}{2}$.

Let $C = \{(x, -x) \mid x \in \mathbb{R}\}$. For $p = \frac{1}{2}$, the affine line $C + p$ is tangent to the blue plot and so, whatever the values of x and y, $b + x\boldsymbol{u} + y\boldsymbol{v}$ cannot be in V. For $p \neq \frac{1}{2}$, there is a value δ such that the whole interval strictly between b and $b + (\delta, -\delta)$ is in V. Since $\log(2)$ and $\log(3)$ are rationally independent, the set $D = \{x\boldsymbol{u} + y\boldsymbol{v} \mid x, y \in \mathbb{N}\}$ is a dense subset of C, so in particular, there is a point of $D + b$ in the interval between b and $b + (\delta, -\delta)$ and thus there exist $x, y \in \mathbb{N}$ such that $b + x\boldsymbol{u} + y\boldsymbol{v} \in V$.

3.2 Decidability

We prove here the decidability of the containment problem when $\mathcal{A}$ is finitely ambiguous and $\mathcal{B}$ is unambiguous. The converse situation is tackled in Section 4.

▶ **Proposition 5.** *Determining whether $[\![\mathcal{A}]\!] \leq [\![\mathcal{B}]\!]$ is decidable when $\mathcal{A}$ is finitely ambiguous and $\mathcal{B}$ is unambiguous.*

Proof. Let $\mathcal{A}$ be k-ambiguous. Proposition 3 shows that it is sufficient to decide, given an integer n and positive rational numbers $p, q_j, r_i, s_{i,j}$ for $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, n\}$, whether there exists $x_1, \ldots, x_n \in \mathbb{N}$ such that

$$\sum_{i=1}^{k} p_i q_{i,1}^{x_1} \cdots q_{i,n}^{x_n} > r s_1^{x_1} \cdots s_n^{x_n}. \tag{1}$$

We consider two cases. First, assume that there exist i and j such that $q_{i,j} > s_j$. Then in that case, for a large enough $m \in \mathbb{N}$ condition (1) will be satisfied for $(x_1, \ldots, x_j, \ldots, x_n) = (0, \ldots, m, \ldots, 0)$. Otherwise, assume that $\max\{q_{i,j} \mid 1 \leq i \leq k\} \leq s_j$ for all $1 \leq j \leq n$. In this case, if there exists a valuation of the x_i satisfying (1) then $(x_1, \ldots, x_n) = (0, \ldots, 0)$ also satisfies it. It is then sufficient to test condition (1) for $x_1 = \cdots = x_n = 0$ to conclude. ◀

4 Decidability of the case unambiguous vs. finitely ambiguous

In this section we will show the more challenging part of Theorem 2, i.e., that the containment problem is decidable for $\mathcal{A}$ unambiguous and $\mathcal{B}$ finitely ambiguous. Our proof is conditional on the first-order theory of the reals with the exponential function being decidable. In [15], the authors show that this is the case if a conjecture due to Schanuel and regarding transcendental number theory is true.

▶ **Theorem 6.** *Determining whether* $[\![\mathcal{A}]\!] \leq [\![\mathcal{B}]\!]$ *is decidable when* $\mathcal{A}$ *is unambiguous and* $\mathcal{B}$ *is finitely ambiguous, assuming Schanuel's conjecture is true.*

4.1 Integer programming problem with exponentiation

Given two positive integers n and ℓ, we define $\mathcal{F}_{n,\ell}$ to be the set of all the functions $f : \mathbb{R}^n \to \mathbb{R}$ such that there exist $\boldsymbol{r} \in \mathbb{Q}^\ell_{>0}$ and $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_\ell \in \mathbb{Q}^n_{>0}$ such that $f(\boldsymbol{x}) = \sum_{i=1}^{\ell} r_i s_{i,1}^{x_1} \ldots s_{i,n}^{x_n}$. Observe that this is just a lifting of the $S(\cdot)$ function, defined in the previous section, to real-valued parameters. Consider the following integer programming problem with exponentiation.

▶ **Problem 7** (IP+EXP).
Input: *Three positive integers n, ℓ and m, a function $f \in \mathcal{F}_{n,\ell}$, a matrix $M \in \mathbb{Z}^{m\times n}$, and a vector $\boldsymbol{c} \in \mathbb{Z}^m$.*
Question: *Does there exist $\boldsymbol{x} \in \mathbb{Z}^n$ such that $f(\boldsymbol{x}) < 1$ and $M\boldsymbol{x} < \boldsymbol{c}$?*

In the sequel, we will show that the above problem is decidable.

▶ **Theorem 8.** *The IP+EXP problem is decidable, assuming Schanuel's conjecture is true.*

Theorem 6 is a direct corollary of Theorem 8.

Proof of Theorem 6. Proposition 3 shows that, in order to prove Theorem 6, it is sufficient to decide, given an integer n and positive rational numbers $p, r_i, q_j, s_{i,j}$ for $i \in \{1, \ldots, \ell\}$, $j \in \{1, \ldots, n\}$, whether there exist $x_1, \ldots, x_n \in \mathbb{N}$ such that $pq_1^{x_1} \cdots q_n^{x_n} > \sum_{i=1}^{\ell} r_i s_{i,1}^{x_1} \cdots s_{i,n}^{x_n}$ or equivalently, whether there exist $x_1, \ldots, x_n \in \mathbb{N}$ such that:

$$\sum_{i=1}^{\ell} r_i p^{-1} (s_{i,1} q_1^{-1})^{x_1} \cdots (s_{i,n} q_n^{-1})^{x_n} < 1. \tag{2}$$

Define $f : \mathbb{R}^n \to \mathbb{R}$ such that $f(\boldsymbol{x}) = \sum_{i=1}^{\ell} r_i p^{-1} (s_{i,1} q_1^{-1})^{x_1} \cdots (s_{i,n} q_n^{-1})^{x_n}$. Then, inequality (2) becomes $f(\boldsymbol{x}) < 1$. We can now apply Theorem 8 with m set to be n; M, to be $-Id$, where Id is the identity matrix; and $\boldsymbol{c}$ to be the null vector. ◀

Since the IP+EXP problem is semi-decidable (indeed, we can enumerate the vectors $\boldsymbol{x}$ in $\mathbb{Z}^n$ to find one satisfying the conditions), it will suffice to give a semi-decision procedure to determine whether the inequalities $f(\boldsymbol{x}) < 1 \wedge M\boldsymbol{x} < \boldsymbol{c}$ have no integer solution. We give now such a procedure.

4.2 Semi-decision procedure for the complement of IP+EXP

Consider as input for the IP+EXP problem three positive integers n, ℓ, m, a function $f \in \mathcal{F}_{n,\ell}$, a matrix $M \in \mathbb{Z}^{m\times n}$, and a vector $\boldsymbol{c} \in \mathbb{Z}^m$. Denote by X the set of *real* solutions of the problem, i.e., the set of vectors $X = \{\boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) < 1 \wedge M\boldsymbol{x} < \boldsymbol{c}\}$.

Proc(n, ℓ, m, f, M, $\boldsymbol{c}$):

1. Search for a non-zero vector $\boldsymbol{d} \in \mathbb{Z}^n$ and $a, b \in \mathbb{Z}$ such that $\{\boldsymbol{d}^\top \boldsymbol{x} \mid \boldsymbol{x} \in X\} \subseteq [a, b]$. Set $i = a$.
2. If $i > b$, then stop and return `YES`. Otherwise, let Y_i be the set of vectors $\boldsymbol{x} \in \mathbb{Z}^n$ satisfying $d_1 x_1 + \cdots + d_n x_n = i$. If Y_i is empty, then increment i and start again from step 2. Otherwise:
   a. Compute $N \in \mathbb{Z}^{n\times(n-1)}$ and $\boldsymbol{h} \in \mathbb{Z}^n$ such that $Y_i = \{N\boldsymbol{y} + \boldsymbol{h} \mid \boldsymbol{y} \in \mathbb{Z}^{n-1}\}$.

b. If $n-1=0$ and $f(\boldsymbol{h}) < 1 \wedge M\boldsymbol{h} < \boldsymbol{c}$ then return `NO`, otherwise increment i and start again from step 2.

c. If $n-1>0$ then recursively call $\mathbf{Proc}(n-1,\ \ell,\ m,\ f',\ M',\ \boldsymbol{c}')$, where $f' \in \mathcal{F}_{n-1,\ell}$ is defined as $f'(\boldsymbol{y}) = f(N\boldsymbol{y}+\boldsymbol{h})$; $M' \in \mathbb{Z}^{m\times(n-1)}$, as $M' = MN$; and $\boldsymbol{c}' \in \mathbb{Z}^m$, as $\boldsymbol{c} - M\boldsymbol{h}$. If the procedure stops and returns `YES` then increment i and start again from step 2. If the procedure stops and returns `NO` then return `NO`.

▶ **Lemma 9.** *The above semi-decision procedure stops and outputs `YES` if and only if there is no integer valuation of $\boldsymbol{x}$ that satisfies the constraints, i.e. $X \cap \mathbb{Z}^n$ is empty.*

First, notice that the only step which might not terminate in a call to our procedure is step 1. Indeed, once $\boldsymbol{d}$, a, and b are fixed, there are only finitely many integers $i \in [a,b]$ that have to be considered in step 2. For each of them, one can compute in a standard way (see, e.g., [3]) the set of integer solutions of the equation $d_1x_1 + \cdots + d_nx_n = i$ and the corresponding N and $\boldsymbol{h}$ as in the procedure.

Moreover, for each integer vector $\boldsymbol{d} \in \mathbb{Z}^n$ and $a,b \in \mathbb{Z}$, the inclusion $\{\boldsymbol{d}^\top\boldsymbol{x} \mid \boldsymbol{x} \in X\} \subseteq [a,b]$ that needs to be checked in step 1 can be formulated as a decision problem in the first-order logic over the structure $(\mathbb{R}, +, \times, \exp)$. Since this structure has a decidable first-order theory subject to Schanuel's conjecture [15], the inclusion can be decided for each fixed $\boldsymbol{d}$, a, and b.

To prove Lemma 9 we use the two following lemmata. The first one is the most technical contribution of the paper and is proved in Section 4.3. It ensures termination of step 1 in the procedure when there is no integer solution.

▶ **Lemma 10.** *If the set X contains no integer point then there must exist a non-zero integer vector $\boldsymbol{d} \in \mathbb{Z}^n$ and $a,b \in \mathbb{Z}$ such that $\{\boldsymbol{d}^\top\boldsymbol{x} \mid \boldsymbol{x} \in X\} \subseteq [a,b]$.*

▶ **Lemma 11.** *Given a non-zero vector $\boldsymbol{d} \in \mathbb{Z}^n$ and an integer i, there exists $\boldsymbol{x} \in \mathbb{Z}^n$ such that $f(\boldsymbol{x}) < 1 \wedge M\boldsymbol{x} < \boldsymbol{c} \wedge \boldsymbol{d}^\top\boldsymbol{x} = i$ if and only if there exists $\boldsymbol{y} \in \mathbb{Z}^{n-1}$ such that $f'(\boldsymbol{y}) < 1 \wedge M'\boldsymbol{y} < \boldsymbol{c}'$ where f', M' and $\boldsymbol{c}'$ are as defined in the procedure.*

We can now prove Lemma 9.

First direction: when the procedure returns YES. Suppose first that the semi-decision procedure stops and outputs `YES`. Then there exist a non-zero vector $\boldsymbol{d} \in \mathbb{Z}^n$ and $a,b \in \mathbb{Z}$ such that $\{\boldsymbol{d}^\top\boldsymbol{x} \mid \boldsymbol{x} \in X\} \subseteq [a,b]$ as in step 1, and for all integers $i \in [a,b]$, one of the following situations occurs:

1. Y_i is empty,
2. $n-1=0$, $Y_i = \{\boldsymbol{h}\}$ as defined in step 2.a but $\boldsymbol{h}$ is not an integer solution of the problem,
3. $n-1>0$ and the recursive call stops and outputs `YES`.

By definition of $\boldsymbol{d}$, in order to prove that there is no integer solution of the problem, we need to show that in all those cases, and for all $i \in [a,b]$, $Y_i \cap X = \emptyset$. It is clear for items 1 and 2 and we use Lemma 11 and an induction for item 3.

Second direction: when $X \cap \mathbb{Z}^n = \emptyset$. If there is no integer solution then by Lemma 10, there must exist a non-zero vector $\boldsymbol{d} \in \mathbb{Z}^n$ and $a,b \in \mathbb{Z}$ such that $\{\boldsymbol{d}^\top\boldsymbol{x} \mid \boldsymbol{x} \in X\} \subseteq [a,b]$ as in step 1. Moreover, for any of those choices, if for an integer $i \in [a,b]$, the set Y_i of vectors $\boldsymbol{x} \in \mathbb{Z}^n$ satisfying $d_1x_1 + \cdots + d_nx_n = i$ is non-empty, then,

1. if $n=1$, then $\boldsymbol{h}$ as defined in step 2.a, is not a solution of the problem (by hypothesis) and thus the procedure stops and returns `YES`,
2. if $n>1$, we use Lemma 11 and, by induction, the recursive call must return `YES`.

4.3 Proof of Lemma 10

Fix three positive integers n, ℓ, m, a function $f \in \mathcal{F}_{n,\ell}$, a matrix $M \in \mathbb{Z}^{m\times n}$, and a vector $\boldsymbol{c} \in \mathbb{Z}^m$. Recall that we denote by X the set of vectors

$$X = \{\boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) < 1 \wedge M\boldsymbol{x} < \boldsymbol{c}\}.$$

We want to prove that if the set X contains no integer point then there must exist a non-zero integer vector $\boldsymbol{d} \in \mathbb{Z}^n$ and $a, b \in \mathbb{Z}$ such that $\{\boldsymbol{d}^\top \boldsymbol{x} \mid \boldsymbol{x} \in X\} \subseteq [a, b]$.

We will use the following corollary of Kronecker's theorem on simultaneous Diophantine approximation. It generalises the fact that any line in the plane with irrational slope passes arbitrarily close to integer points in the plane.

▶ **Proposition 12.** *[12, Corollary 2.8]. Let $\boldsymbol{u}, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_s$ be vectors in $\mathbb{R}^n$. Suppose that for all $\boldsymbol{d} \in \mathbb{Z}^n$ we have $\boldsymbol{d}^\top \boldsymbol{u} = 0$ whenever $\boldsymbol{d}^\top \boldsymbol{u}_1 = \cdots = \boldsymbol{d}^\top \boldsymbol{u}_s = 0$. Then for all $\varepsilon > 0$ there exist real numbers $\lambda_1, \ldots, \lambda_s \geq 0$ and a vector $\boldsymbol{v} \in \mathbb{Z}^n$ such that $\|\boldsymbol{u} + \sum_{i=1}^s \lambda_i \boldsymbol{u}_i - \boldsymbol{v}\|_\infty \leq \varepsilon$.*

By definition, there exist vectors $\boldsymbol{r} \in \mathbb{Q}^\ell_{>0}$ and $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_\ell \in \mathbb{Q}^n_{>0}$ such that $f(\boldsymbol{x}) = \sum_{i=1}^\ell r_i s_{i,1}^{x_1} \ldots s_{i,n}^{x_n}$. Let $\boldsymbol{a} \in \mathbb{R}^\ell$ and $\boldsymbol{b}_i \in \mathbb{R}^n$ be defined by $a_i = \log(r_i)$ and $\boldsymbol{b}_i = (\log(s_{i,1}), \ldots, \log(s_{i,n}))$. We can then rewrite $f(\boldsymbol{x})$ as follows

$$f(\boldsymbol{x}) = \exp(\boldsymbol{b}_1^\top \boldsymbol{x} + a_1) + \cdots + \exp(\boldsymbol{b}_\ell^\top \boldsymbol{x} + a_\ell).$$

Let us now consider the cone

$$C = \left\{\boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \boldsymbol{b}_1^\top \boldsymbol{x} \leq 0 \wedge \cdots \wedge \boldsymbol{b}_\ell^\top \boldsymbol{x} \leq 0 \wedge M\boldsymbol{x} \leq 0\right\}. \tag{3}$$

It is easy to see that $X + C \subseteq X$.

▶ **Lemma 13.** *Suppose that X is non-empty and that no non-zero integer vector in $\mathbb{Z}^n$ is orthogonal to C. Then $X \cap \mathbb{Z}^n$ is non-empty.*

Proof. Let $\boldsymbol{u} \in X$. Since X is open, there exists $\varepsilon > 0$ such that the open ball $B_\varepsilon(\boldsymbol{u})$ is contained in X. We therefore have that $B_\varepsilon(\boldsymbol{u}) + C \subseteq X$.

We will apply Proposition 12 to show that $B_\varepsilon(\boldsymbol{u}) + C$ contains an integer point and hence that X contains an integer point. To this end, let vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_s \in C$ be such that $\operatorname{span}\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_s\} = \operatorname{span}(C)$. Then no non-zero vector in $\mathbb{Z}^n$ is orthogonal to $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_s$. By Proposition 12, there exist real numbers $\lambda_1, \ldots, \lambda_s \geq 0$ and an integer vector $\boldsymbol{v} \in \mathbb{Z}^n$ such that $\|\boldsymbol{u} + \sum_{i=1}^s \lambda_i \boldsymbol{u}_i - \boldsymbol{v}\|_\infty \leq \varepsilon$. Thus, $\boldsymbol{v} \in B_\varepsilon(\boldsymbol{u}) + C \subseteq X$. ◀

The contrapositive of the above result states that if X contains no integer point, then there must exist an integer vector that is orthogonal to C. For the desired result, it remains for us to prove the boundedness claim.

▶ **Lemma 14.** *Suppose that $\boldsymbol{d} \in \mathbb{Z}^n$ is orthogonal to the cone C. Then $\{\boldsymbol{d}^\top \boldsymbol{u} \mid \boldsymbol{u} \in X\}$ is bounded.*

Proof. Define the "enveloping polygon" of X to be

$$\widehat{X} = \left\{\boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \boldsymbol{b}_1^\top \boldsymbol{x} + \boldsymbol{a}_1 \leq 0 \wedge \cdots \wedge \boldsymbol{b}_\ell^\top \boldsymbol{x} + \boldsymbol{a}_\ell \leq 0 \wedge M\boldsymbol{x} \leq \boldsymbol{c}\right\}.$$

Clearly it holds that $X \subseteq \widehat{X}$. Moreover, by the Minkowski-Weyl decomposition theorem we can write $\widehat{X}$ as a sum $\widehat{X} = B + C$ for B a bounded polygon and C the cone defined in (3). Since $\boldsymbol{d}$ is orthogonal to C by assumption, it follows that $\{\boldsymbol{d}^\top \boldsymbol{u} \mid \boldsymbol{u} \in \widehat{X}\} = \{\boldsymbol{d}^\top \boldsymbol{u} \mid \boldsymbol{u} \in B\}$ is bounded and hence $\{\boldsymbol{d}^\top \boldsymbol{u} \mid \boldsymbol{u} \in X\}$ is bounded. The result immediately follows. ◀

We can now complete the proof of Lemma 10.

Proof of Lemma 10. By Lemma 13 there exists a non-zero integer vector $\boldsymbol{d} \in \mathbb{Z}^n$ such that $\boldsymbol{d}$ is orthogonal to the cone C defined in (3). Then by Lemma 14 we obtain that $\{\boldsymbol{d}^\top \boldsymbol{u} \mid \boldsymbol{u} \in X\}$ is contained in a bounded interval. ◀

5 Undecidability for linearly ambiguous automata

In this section we prove Theorem 1. That is, we argue that the containment problem is undecidable for the class of linearly ambiguous PA. For most of this section we deal with the following variant of the problem: does there exist a word w such that $[\![\mathcal{A}]\!](w) \leq [\![\mathcal{B}]\!](w)$. Towards the end, we shortly explain how our reduction proves undecidability for any variant of this problem (that is, with inequality $<$). Moreover, we derive from the proof that all variants of the emptiness problem are undecidable for a given linearly ambiguous automaton $\mathcal{A}$, i.e. whether there exists w such that $[\![\mathcal{A}]\!](w) \geq \frac{1}{2}$ (or resp. $>$, $\leq$, $<$).

The proof is done by a reduction from the halting problem for two-counter machines. The reduction resembles the one used to prove undecidability of the comparison problem for another quantitative extension of Boolean automata: max-plus automata [4, 1]. We give here an outline of the proof and the main ideas used in the reduction.

Two-counter machines. A *two-counter machine* (or a *Minsky machine*) is a deterministic finite-state machine, with an initial state, a final state, and two counters, each of which can be incremented or decremented (if its value is not 0). For each counter $i \in \{1, 2\}$, there are two types of transitions: $T_i^+ \subseteq Q^2$ such that a transition $(p, q) \in T_i^+$ moves from state p to state q and increments the i-th counter; and $T_i^- \subseteq Q^3$ such that a transition $(p, q, r) \in T_i^-$ moves from state p to state q if the i-th counter has value 0, and moves from state p to state r decrementing the i-th counter otherwise. The set of all transitions is denoted $T = T_1^+ \cup T_2^+ \cup T_1^- \cup T_2^-$.

The machine *halts* if there is a (unique) computation from the initial state to the final state. It is well-known that deciding whether a given machine halts is undecidable [16]. We use this fact to prove undecidability of the containment problem.

▶ **Proposition 15.** *Given a two-counter machine, one can construct two linearly ambiguous probabilistic automata $\mathcal{A}$ and $\mathcal{B}$, such that the machine halts if and only if there exists a word w such that $[\![\mathcal{A}]\!](w) \leq [\![\mathcal{B}]\!](w)$.*

Outline of the reduction. The general idea is to encode executions of the two-counter machine into words over the alphabet $\Sigma = \{a, b\} \cup T$. A block a^m (resp. b^m) encodes the fact that the value of the first (resp. second) counter is m. For example, given $t \in T_1^+$ and $t' \in T_2^-$, a word $a^n b^m t a^{n+1} b^m t' a^{n+1} b^{m'}$, encodes an execution starting with value n in the first counter and m in the second counter. Transition t then increases the value of the first counter to $n + 1$ without changing the value of the second one. The configuration is thus encoded by the infix $a^{n+1} b^m$. Next, transition t' is taken, and either $m' = m = 0$ or $m' = m - 1$. Moreover, if $t = (p, q)$ and $t' = (r, s, u)$ then $q = r$ (i.e., the states between transitions have to match).

Simulating executions faithfully. Our reduction has the following property: $[\![\mathcal{A}]\!](w) = [\![\mathcal{B}]\!](w)$ for the unique word w encoding the halting execution (if it exists), and $[\![\mathcal{A}]\!](w) > [\![\mathcal{B}]\!](w)$ for all other words. This suffices to prove Proposition 15.

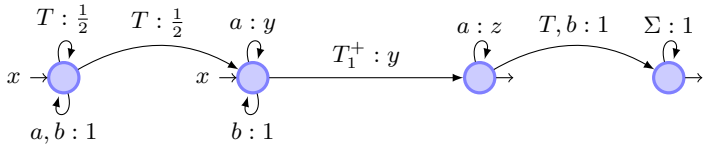


Figure 3 Gadget automaton $\mathcal{C}(x,y,z)$ used to check if the first counter is incremented properly.

There are several conditions that the constructed automata $\mathcal{A}$ and $\mathcal{B}$ need to check to ensure the correctness of the encoding: whether the states match; whether the counter values encoded by blocks of a and b are increased and decreased properly, etc. It is easy to define an automaton $\mathcal{A}_0$ that will allow us to focus only on *proper words* of the form

$$w = a^{n_1}b^{m_1}t_1a^{n_2}b^{m_2}t_2a^{n_3}b^{m_3}t_3\cdots a^{n_k}b^{m_k}t_k. \tag{4}$$

That is, $\mathcal{A}_0$ is such that $[\![\mathcal{A}_0]\!](w) = 0$ if the word w correctly alternates between blocks of a, b, and letters from T; it correctly implements the zero tests within decrement transitions; it observes the initial and final states and checks that the states of adjacent transitions match. Observe that the latter are all regular conditions. If w does not satisfy all these constraints, then $[\![\mathcal{A}_0]\!](w) = 1$.

For other (non-regular) conditions we construct pairs of automata $\mathcal{A}_i$ and $\mathcal{B}_i$, such that $[\![\mathcal{A}_i]\!](w) = [\![\mathcal{B}_i]\!](w)$ if a proper word w does not violate the condition and $[\![\mathcal{A}_i]\!](w) > [\![\mathcal{B}_i]\!](w)$ for all other proper words. In the end, the automata $\mathcal{A}$ and $\mathcal{B}$ are obtained as weighted sums of the automata $\mathcal{A}_i$ and $\mathcal{B}_i$.

Simulating increments. We present the construction for one of the components of $\mathcal{A}$ and $\mathcal{B}$. The remaining ones are obtained using similar gadgets. The automata $\mathcal{A}_1$ and $\mathcal{B}_1$ check that a proper word encodes an execution where the first counter is always correctly incremented after reading transitions from T_1^+. Consider the automaton $\mathcal{C}(x,y,z)$ in Figure 3. It is parameterised by three probability variables $x,y,z > 0$. The parameter x is the probability used by the initial distribution, and parameters y and z are used by some transitions.

Consider a word w as in (4). Notice that the only non-deterministic transitions in $\mathcal{C}(x,y,z)$ are the ones going out from the the leftmost state upon reading letters from T. It follows that $\mathcal{C}(x,y,z)$ is linearly ambiguous. In fact, for every position in w labelled by an element t from T_1^+ there is a unique accepting run that first reaches a final state upon reading t. By construction, we have

$$[\![\mathcal{C}(x,y,z)]\!](w) = \sum_{t_i \in T_1^+} x\left(\frac{1}{2}\right)^{i-1} y^{n_i+1}z^{n_{i+1}}. \tag{5}$$

Let $x = \frac{1}{2}$, $y = 1$ and $z = \frac{1}{4}$. We define $\mathcal{B}_1$ as $\mathcal{C}(x,x,x)$ and $\mathcal{A}_1$ as a weighted sum of $\mathcal{C}(x,y,z)$ and $\mathcal{C}(x,z,y)$ with weights $(\frac{1}{2},\frac{1}{2})$. Since $\mathcal{C}(\cdot,\cdot,\cdot)$ is linearly ambiguous the obtained automata are also linearly ambiguous. We prove that $[\![\mathcal{A}_1]\!](w) = [\![\mathcal{B}_1]\!](w)$ only if $n_i+1 = n_{i+1}$ for all i such that $t_i \in T_1^+$ and $[\![\mathcal{A}_1]\!](w) > [\![\mathcal{B}_1]\!](w)$ otherwise.

By (5) it suffices to show that for every i it holds that

$$\left(\frac{1}{2}\right)^{n_i+1+n_{i+1}} \leq \frac{1}{2}\left(\left(\frac{1}{4}\right)^{n_{i+1}} + \left(\frac{1}{4}\right)^{n_i+1}\right)$$

and that the equality holds only if $n_i + 1 = n_{i+1}$, which is the case.

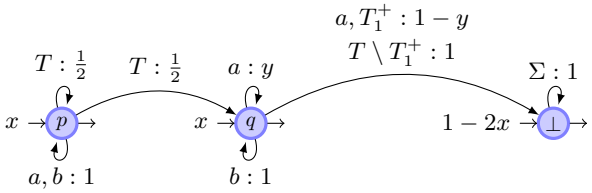


Figure 4 Complement automaton of $\mathcal{C}(x, y, z)$ after trimming.

Changes for version with strict inequalities. Observe that if PA $\mathcal{A}$ and $\mathcal{B}$ output different probabilities on a word w then the probabilities must differ by at least $x^{|w|+1}$ for some value x depending on the probabilities used in $\mathcal{A}$ and $\mathcal{B}$ (see, e.g., [10]). By a weighted sum construction, we can define a PA $\mathcal{B}'$ associating a word w to the value $[\![\mathcal{B}]\!](w) + x^{|w|+1}$. Then there is a word w such that $[\![\mathcal{A}]\!](w) \leq [\![\mathcal{B}]\!](w)$ if and only if there is a word w such that $[\![\mathcal{A}]\!](w) < [\![\mathcal{B}']\!](w)$.

Changes for the emptiness problem. Let us consider the containment problem $[\![\mathcal{A}]\!] \leq [\![\mathcal{B}]\!]$. For all words w, we have that $[\![\mathcal{A}]\!](w) \leq [\![\mathcal{B}]\!](w)$ holds if and only if $\frac{1}{2}[\![\mathcal{A}]\!](w) + \frac{1}{2}(1 - [\![\mathcal{B}]\!](w)) \leq \frac{1}{2}$. Thus, using the weighted sum and complement operations we can get a single probabilistic automaton that outputs $\frac{1}{2}[\![\mathcal{A}]\!](w) + \frac{1}{2}(1 - [\![\mathcal{B}]\!](w))$. Complementing an automaton may, in general, increase the ambiguity (see Figure 1). However, for the automata we have constructed, this is not the case. For example, it is easy to check that the complement automaton $\overline{\mathcal{C}(x, y, z)}$ depicted in Figure 4 is—just like $\mathcal{C}(x, y, z)$—also linearly ambiguous.

6 Conclusion

In this work we have shown that the containment problem for probabilistic automata is decidable if one of the automata is finitely ambiguous and the other one is unambiguous. Interestingly, for one of the two cases, our proposed algorithm uses a satisfiability oracle for a theory whose decidability is equivalent to a weak form of Schanuel's conjecture. We have complemented our decidability results with a proof of undecidability for the case when the given automata are linearly ambiguous.

Decidability of the containment problem when both automata are allowed to be finitely ambiguous remains open. One way to tackle it is to study generalizations of the IP+EXP problem introduced in Section 4. This problem asks whether there exists $\boldsymbol{x} \in \mathbb{N}^n$ such that $f(\boldsymbol{x}) < 1$ and $M\boldsymbol{x} < \boldsymbol{c}$ for a given function f defined using exponentiations, a given matrix M, and vector $\boldsymbol{c}$. A natural way to extend the latter would be to ask that $f(\boldsymbol{x}) < g(\boldsymbol{x})$, where g is obtained in a similar way as f. The main obstacle, when trying to generalize our decidability proof for that problem, is that we lack a replacement for the cone C needed in order to obtain a result similar to Lemma 14 using the Minkowski-Weyl decomposition.

References

1 Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? In Tevfik Bultan and Pao-Ann Hsiung, editors, *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings*, volume 6996 of *Lecture Notes in Computer Science*, pages 482–491. Springer, 2011. doi:10.1007/978-3-642-24372-1_37.

2 Rohit Chadha, A. Prasad Sistla, Mahesh Viswanathan, and Yue Ben. Decidable and expressive classes of probabilistic automata. In Andrew M. Pitts, editor, *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9034 of *Lecture Notes in Computer Science*, pages 200–214. Springer, 2015. doi:10.1007/978-3-662-46678-0_13.

3 Henri Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate texts in mathematics*. Springer, 1993. URL: http://www.worldcat.org/oclc/27810276.

4 Thomas Colcombet. On distance automata and regular cost function. Presented at the Dagstuhl seminar "Advances and Applications of Automata on Words and Trees", 2010.

5 Lu Feng, Tingting Han, Marta Z. Kwiatkowska, and David Parker. Learning-based compositional verification for synchronous probabilistic systems. In Tevfik Bultan and Pao-Ann Hsiung, editors, *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011*, volume 6996 of *Lecture Notes in Computer Science*, pages 511–521. Springer, 2011. doi:10.1007/978-3-642-24372-1_40.

6 Nathanaël Fijalkow. Undecidability results for probabilistic automata. *SIGLOG News*, 4(4):10–17, 2017. doi:10.1145/3157831.3157833.

7 Nathanaël Fijalkow, Hugo Gimbert, Edon Kelmendi, and Youssouf Oualhadj. Deciding the value 1 problem for probabilistic leaktight automata. *Logical Methods in Computer Science*, 11(2), 2015. doi:10.2168/LMCS-11(2:12)2015.

8 Nathanaël Fijalkow, Cristian Riveros, and James Worrell. Probabilistic automata of bounded ambiguity. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany*, volume 85 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CONCUR.2017.19.

9 Holger Giese, Nelly Bencomo, Liliana Pasquale, Andres J. Ramirez, Paola Inverardi, Sebastian Wätzoldt, and Siobhán Clarke. Living with uncertainty in the age of runtime models. In Nelly Bencomo, Robert B. France, Betty H. C. Cheng, and Uwe Assmann, editors, *Models@run.time - Foundations, Applications, and Roadmaps [Dagstuhl Seminar 11481, November 27 - December 2, 2011]*, volume 8378 of *Lecture Notes in Computer Science*, pages 47–100. Springer, 2014. doi:10.1007/978-3-319-08915-7_3.

10 Hugo Gimbert and Youssouf Oualhadj. Probabilistic automata on finite words: Decidable and undecidable problems. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 527–538. Springer, 2010. doi:10.1007/978-3-642-14162-1_44.

11 Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996. doi:10.1613/jair.301.

12 Leonid Khachiyan and Lorant Porkolab. Computing integral points in convex semi-algebraic sets. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 162–171. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646105.

13 Stefan Kiefer, Andrzej S. Murawski, Joël Ouaknine, Björn Wachter, and James Worrell. On the complexity of equivalence and minimisation for q-weighted automata. *Logical Methods in Computer Science*, 9(1), 2013. doi:10.2168/LMCS-9(1:8)2013.

14 Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. Assume-guarantee verification for probabilistic systems. In Javier Esparza and Rupak Majumdar,

editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 23–37. Springer, 2010. doi:10.1007/978-3-642-12002-2_3.

15 Angus Macintyre and Alex J. Wilkie. On the decidability of the real exponential field. In Piergiorgio Odifreddi, editor, *Kreiseliana. About and Around Georg Kreisel*, pages 441–467. AK Peters, 1996.

16 Marvin Lee Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

17 Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002. doi:10.1006/csla.2001.0184.

18 Krishna V. Palem and Lingamneni Avinash. Ten years of building broken chips: The physics and engineering of inexact computing. *ACM Transactions on Embedded Computing Systems*, 12(2s):87:1–87:23, 2013. doi:10.1145/2465787.2465789.

19 Martin L. Puterman. *Markov Decision Processes*. Wiley-Interscience, 2005.

20 Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963. doi:10.1016/S0019-9958(63)90290-0.

21 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Compututer Science*, 6:223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

22 Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010. URL: http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0136042597,00.html.

23 Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 327–338. IEEE Computer Society, 1985. doi:10.1109/SFCS.1985.12.

24 Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325–349, 1991. doi:10.1016/0304-3975(91)90381-B.

25 Abuzer Yakaryilmaz and A. C. Cem Say. Unbounded-error quantum computation with small space bounds. *Information and Computation*, 209(6):873–892, 2011. doi:10.1016/j.ic.2011.01.008.

On the Complexity of Infinite Advice Strings

Gaëtan Douéneau-Tabot[1]
École Normale Supérieure Paris-Saclay, Université Paris-Saclay, Cachan, France
gaetan.doueneau@ens-paris-saclay.fr

Abstract

We investigate in this paper a notion of comparison between infinite strings. In a general way, if $\mathcal{M}$ is a computation model (e.g. Turing machines) and $\mathcal{C}$ a class of objects (e.g. languages), the complexity of an infinite word α can be measured with respect to the amount of objects from $\mathcal{C}$ that are presentable with machines from $\mathcal{M}$ using α as an oracle.

In our case, the model $\mathcal{M}$ is finite automata and the objects $\mathcal{C}$ are either recognized languages or presentable structures, known respectively as advice regular languages and advice automatic structures. This leads to several different classifications of infinite words that are studied in detail; we also derive logical and computational equivalent measures. Our main results explore the connections between classes of advice automatic structures, MSO-transductions and two-way transducers. They suggest a closer study of the resulting hierarchy over infinite words.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases infinite words, advice automata, automatic structures, transducers

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.122

Related Version An extended version with full proofs is available at [9], `https://arxiv.org/abs/1801.04908`.

1 Introduction

Several measures have been defined to describe the (intuitive) complexity of infinite strings; among others we mention subword complexity, Kolmogorov complexity, and Turing degrees. Whereas the two first methods focus on the intrinsic information contained in a string, the other one studies the relation of computability from one word to another, defining a preorder whose properties are now quite well understood. Equivalently, this preorder compares the expressive power of Turing machines that use an infinite word as oracle.

This paper follows a similar idea: we consider finite automata that can access an infinite *advice* string while processing their input. Such automata define classes of *advice regular languages* [17], that generalize standard regularity. This notion enables us to introduce a way to compare infinite words: α is simpler (in the sense of languages) than β if every language recognized by an automaton with advice α can also be recognized with advice β. It corresponds to some intuition that α contains less information than β.

Before going further, we evoke the current motivations around advice regular languages. Standard regular languages can be used to encode finite-signature structures, known as *automatic structures*. This concept, derived from Büchi's early automata-logic connections, has been shown especially relevant since its formalization in the 1990's (see e.g. [8]). The model opened the door to a vast range of decision procedures via automata constructions, but it suffers from a lack of expressiveness, since e.g. $\langle \mathbb{Q}, + \rangle$ is not automatic [20]. However, $\langle \mathbb{Q}, + \rangle$ is an example of *advice automatic structure*: it can be encoded using advice regular

[1] This work was partially done during a stay of the author in RWTH Aachen University.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 122; pp. 122:1–122:13


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

languages (instead of regular languages) [13]. Such structures share many properties with the former automatic structures, furthermore the use of advices builds a rich framework to discuss algorithmic meta-theorems [2]. We shall not follow a model-theoretic point of view on advice automatic structures, but we use them to define another notion of comparison over infinite words as follows: α is simpler (in the sense of structures) than β if every automatic structure with advice α is also automatic with advice β.

Objectives and outline. This paper is structured as a quest for a relevant way to compare infinite strings through the notion of advice. The informal criteria we use to define a "good" complexity measure are the following: it should have a simple definition, be robust enough, but not too coarse because we want to separate simple classes of sequences. Note that Turing degrees do not match this intuition since they make no distinction between all computable (thus useful in practise) sequences. Our results will establish an interesting correspondence between the expressive power of advices (compared more or less using *languages*) and certain forms of *transductions*, when considering the way they classify infinite strings. This is somehow surprising, since the theory of transformations between words tends to be more fruitful and more difficult than the study of languages, following an early remark of Dana Scott [18]: "the functions computed by the various machines are more important - or at least more basic - than the sets accepted by these devices". The concept of advice helps unifying these frameworks. Furthermore, we shall use this idea to provide slightly new perspectives on (advice) automatic presentations and logic over infinite words.

After recalling preliminary results on formal languages, structures and logic, we present formally in Section 3 the notion of regularity with advice, under several variants. We study the comparisons of words provided by the classes of advice languages, as evoked above. An easy correspondence is drawn with transductions, for instance we show that every regular language with advice α is also regular with β if and only if α is the image of β under a Mealy machine. Nevertheless, we conclude that comparisons via languages are far from being robust. The last part of this section introduces the classes of advice automatic structures and briefly describes some of their properties. We then show that some variants of advice regular languages have no influence on the classes of presentable structures. This first involved result is also a first step to obtain a new robust notion of comparison.

Section 4 intends to understand the comparison over infinite words defined with respect to advice automatic structures (see above); it develops our most interesting contributions. Similar investigations were built in [14] under the formalism of set-interpretations, a very close notion. We particularize their results to show that every automatic structure with advice α is also automatic with β if and only if α is the image of β under an MSO-transduction (some logical transformation between words). We then give a more handy equivalent statement: α is the image of β under a two-way transducer. This result is quite specific and original, since such transducers are however not powerful enough to realize all *functions* of infinite words defined by MSO-transductions [4]. In Section 5 we investigate the structural properties of this relation of comparison (defined in particular by two-way transductions). Even if no previous research was done on the subject, a similar study was carried out in [10] for comparison by one-way finite transducers. In the light of their results, we rough out the structure of a new hierarchy and explain why a more involved questioning may be fruitful.

2 Preliminaries

Greek capitals Σ, Γ and Δ are used to denote alphabets, i.e. finite sets of letters; $\Box$ is a padding letter that never belongs to these alphabets. If w is a (possibly infinite) word, let

$|w| \in \mathbb{N} \cup \{\omega\}$ be its length, and for $n \geq 0$ let $w[n]$ be its $(n+1)$-th letter (when defined). For $0 \leq m \leq n$, let $w[m:n] = w[m]w[m+1]\cdots w[n-1]$ (when defined, possibly ε). We write $w[:n]$ for the prefix $w[0:n]$, and $w[n:]$ for the (possibly infinite) suffix $w[n]w[n+1]\cdots$. Denote by Reg (resp. ωReg) the class of regular (resp. ω-regular) languages.

We shall deal with structures over a finite (relational) signature, denoted by fraktur letters $\mathfrak{A}$, $\mathfrak{B}$, etc. Equality implicitly belongs to every signature. If τ is a signature and $\mathcal{L}$ a logic, $\mathcal{L}[\tau]$-formulas are $\mathcal{L}$-formulas over the signature τ. Let $\alpha \in \Gamma^\omega$, its word structure $\mathfrak{W}^\alpha = \langle \mathbb{N}, <, (P_a)_{a\in\Gamma}\rangle$ is defined with $<$ the usual ordering, and $n \in P_a$ if $\alpha[n] = a$. For succinctness reasons, $\alpha \models \phi$ stands for $\mathfrak{W}^\alpha \models \phi$ and $\mathsf{MSO}[<,\Gamma]$ for $\mathsf{MSO}[<,(P_a)_{a\in\Gamma}]$. Recall that MSO-formulas can be interpreted using weak semantics (WMSO), where we allow set quantifications to range only over finite sets.

▶ **Definition 1** (convolution). If u and v are (possibly infinite) words, their *convolution* $u \otimes v$ is the word of length $\max(|u|,|v|)$ such that:
- $(u\otimes v)[n] = (u[n], v[n])$ if $n < \min(|u|,|v|)$;
- $(u\otimes v)[n] = (u[n], \square)$ if $|v| \leq n < |u|$;
- $(u\otimes v)[n] = (\square, v[n])$ if $|u| \leq n < |v|$.

▶ **Definition 2** (presentation). Let $\mathfrak{A} := \langle A, R_1 \dots R_n\rangle$ be a relational structure and $\mathcal{C}$ a class of languages (possibly over infinite words). A *$\mathcal{C}$-presentation* of $\mathfrak{A}$ is a tuple $(L, L_=, L_1 \dots L_n)$ of languages from $\mathcal{C}$ such that there exists a surjective function $\nu : L \to A$ with:
- $L_= = \{w \otimes w' \mid w, w' \in L \text{ and } \nu(w) = \nu(w')\}$;
- for R_i (arity r_i), $L_i = \{w_1 \otimes \cdots \otimes w_{r_i} \mid \forall 1 \leq j \leq r_i, w_j \in L \text{ and } (\nu(w_1), \dots, \nu(w_{r_i})) \in R_i\}$.

The function ν describes how A is encoded in L. Since we never consider the elements of A directly, it does not belong explicitly to the presentation and can be considered as a notation. The alphabet of L is called *encoding alphabet* and often denoted Σ. The presentation is said *injective* if $L_= = \{w \otimes w \mid w \in L\}$. Fairly recently, the class of (ω)Reg-presentable structures generated much attention, under the name of *(ω-)automatic structures* [8]. Such structures can be effectively represented using a tuple of automata recognizing the previous languages. We denote by (ω)AutStr the class of (ω-)automatic structures.

▶ **Example 3.** $\langle \mathbb{N}, +, 0, 1\rangle \in$ AutStr.

▶ **Proposition 4** (folklore, [8]). *Every (ω-)automatic structure has a decidable FO-theory.*

Automatic structures enjoy several other useful properties, but the presentation fails for simple structures with decidable theory, as shown in the next theorem.

▶ **Theorem 5** ([20]). *$\langle \mathbb{Q}, +\rangle$ is not an (ω-)automatic structure.*

A model-theoretic notion closely related to presentations is concept of *interpretation*, where we describe a structure in another (host) structure via a tuple of logical formulas.

▶ **Definition 6** (interpretation). Let $\mathfrak{A}$ be a structure over a signature τ, $\mathcal{L}$ be a logic and $\mathcal{I} := (\phi_\delta(\overline{x}), \phi_=(\overline{x},\overline{y}), \phi_1(\overline{x_1}\dots\overline{x_{r_1}})\dots\phi_p(\overline{x_1}\dots\overline{x_{r_p}}))$ a tuple of $\mathcal{L}[\tau]$-formulas where $\overline{x}, \overline{y}$ and the $\overline{x_i}$ are k-tuples of free variables. Let
- $A_\delta := \{\overline{a} = (a_1 \dots a_k) \mid \mathfrak{A} \models \phi_\delta(\overline{a})\}$;
- $\sim$ is a binary relation on A_δ with $\overline{a} \sim \overline{b}$ if $\mathfrak{A} \models \phi_=(\overline{a},\overline{b})$;
- for $1 \leq i \leq p$, R_i is a relation on A_δ defined as $(\overline{a_1}\dots\overline{a_{r_i}}) \in R_i$ if $\mathfrak{A} \models \phi_i(\overline{a_1}\dots\overline{a_{r_i}})$;

we say that $\mathcal{I}$ is a *k-dimensional $\mathcal{L}$-interpretation* of a structure $\mathfrak{B}$ in the structure $\mathfrak{A}$ if:
- $\sim$ defines an congruence relation on A_δ with respect to $R_1 \dots R_p$;
- $\langle A_\delta, R_1 \dots R_p\rangle / \sim$ is isomorphic to $\mathfrak{B}$.

The interpretation is said *injective* if $\sim$ is the equality relation of A_δ. In the literature, interpretations are often directly assumed to be *1-dimensional injective interpretations*. The choice of the logic $\mathcal{L}$ provides several kinds of interpretation, detailed in Definition 7.

▶ **Definition 7.**
1. An *FO-interpretation* is a tuple of FO-formulas. The elements of $\mathfrak{A}$ are encoded as tuples of elements in the host structure $\mathfrak{B}$.
2. An *MSO-interpretation* is a tuple of MSO-formulas with free first-order variables. If we use the weak semantics, we speak of WMSO-interpretation. Once more, the elements of $\mathfrak{A}$ are encoded as tuples of elements of $\mathfrak{B}$.
3. An *S-interpretation* (set) is a tuple of MSO-formulas with free set variables. If we use weak semantic, we speak of *FS-interpretation* (finite set). The elements of $\mathfrak{A}$ are encoded as tuples of (finite) sets of elements in the host structure.

▶ **Fact 8** (closure under composition)**.**
1. *If $\mathfrak{A}$ is k-dimensionally FO-interpretable in $\mathfrak{B}$ which is l-dimensionally FO-interpretable in $\mathfrak{C}$, then $\mathfrak{A}$ is directly kl-dimensionally FO-interpretable in $\mathfrak{C}$.*
2. *If $\mathfrak{A}$ is k-dimensionally MSO-interpretable in $\mathfrak{B}$ which is 1-dimensionally MSO-interpretable in $\mathfrak{C}$, then $\mathfrak{A}$ is directly k-dimensionally MSO-interpretable in $\mathfrak{C}$*

▶ Remark. The presence of sets and the use of several dimensions force to be careful in the statements of Fact 8. Indeed, there is no reason why the composition of two S-interpretations should be a S-interpretation, since we obtain sets of sets in the whole transformation. A similar argument works for MSO-interpretations without restrictions on the dimension.

▶ Remark. The above composition properties allow - in specific cases - to transfer the decidability of the logical theory from the host structure to the other one.

Interpretations are a key concept to extend standard automata-logic equivalences from regular languages to automatic structures.

▶ **Proposition 9** ([12])**.** *A structure $\mathfrak{A}$ is automatic (resp. ω-automatic) if and only if $\mathfrak{A}$ is FS-interpretable (resp. S-interpretable) in $(\mathbb{N}, <)$.*

3 From advice regular languages to advice automatic structures

We introduce in this section an extension of regular languages known as *regular languages with advice*. This concept enables us to study some preorders over infinite words; we discuss their relevance and establish a first link with transductions. In the last subsection, we describe the structures that can be presented with these classes of languages.

3.1 Terminating languages

The idea of advice regularity is to consider languages accepted by automata that read an infinite advice string while processing its input [5]. We provide an equivalent definition which does not directly deal with automata but only languages.

▶ **Definition 10.** $L \subseteq \Sigma^*$ is *terminating regular* with advice $\alpha \in \Gamma^\omega$ if there exists a regular language $L' \subseteq (\Sigma \times \Gamma)^*$ such that $L = \{w \mid w \otimes \alpha[: |w|] \in L'\}$.

▶ **Example 11.**
1. If $L \subseteq \Sigma^*$ is regular, so is $\{w \otimes w' \mid w \in L, w' \in \Gamma^*, |w| = |w'|\}$, and considering this language shows that L is regular with any advice of Γ^ω;
2. the set $\mathrm{Pref}(\alpha) := \{\alpha[: n] \mid n \geq 0\}$ is regular with advice α.

We denote by $\mathrm{Reg}[\alpha]$ the class of regular languages with advice α. As evoked in the introduction, our goal is to measure the complexity of infinite words, through the expressiveness of their advice classes. We write $\alpha \preccurlyeq_{\mathrm{Reg}} \beta$ whenever $\mathrm{Reg}[\alpha] \subseteq \mathrm{Reg}[\beta]$, this relation is clearly a preorder over infinite words. Let the $\preccurlyeq_{\mathrm{Reg}}$-*degrees* be the equivalence classes of the relation $\preccurlyeq_{\mathrm{Reg}} \cap \succcurlyeq_{\mathrm{Reg}}$, they describe the sets of equally complex advices. We remark that ultimately periodic words (i.e. infinite words of the form uv^ω) form the least $\preccurlyeq_{\mathrm{Reg}}$-degree; indeed the inclusion $\mathrm{Reg} \subseteq \mathrm{Reg}[\alpha]$ is strict if and only if α is not ultimately periodic [5, 16]. We now provide a first equivalence with transductions.

▶ **Definition 12.** A *Mealy machine* is a 6-tuple $(Q, q_0, \Delta, \Gamma, \delta, \theta)$ where Q is the finite set of states, $q_0 \in Q$ initial state, Δ is the input alphabet, Γ is the output alphabet, $\delta : Q \times \Delta \to Q$ is the (partial) transition function, and $\theta : Q \times \Delta \to \Gamma$ is the (partial) output function.

A run of a Mealy machine is a run of the underlying deterministic automaton. On input β, the machine outputs α the concatenation of the outputs along the run on β.

▶ **Proposition 13.** *The following conditions are equivalent:*
1. $\mathrm{Reg}[\alpha] \subseteq \mathrm{Reg}[\beta]$*;*
2. α *is the image of* β *under some Mealy machine.*

Comparison via $\preccurlyeq_{\mathrm{Reg}}$ thus corresponds to computability via Mealy machines. The properties of this preorder were studied under this form in [7]. However, tiny changes in the words completely modify their $\preccurlyeq_{\mathrm{Reg}}$-degree: those classes are far from being robust.

▶ **Fact 14** ([7])**.** *Whenever* α *is not ultimately periodic, we have a strictly increasing chain* $\alpha \prec_{\mathrm{Reg}} \alpha[1:] \prec_{\mathrm{Reg}} \cdots \prec_{\mathrm{Reg}} \alpha[n:] \prec_{\mathrm{Reg}} \cdots$*. A strictly decreasing chain can be obtained similarly with* $\alpha \succ_{\mathrm{Reg}} \Box\alpha \succ_{\mathrm{Reg}} \cdots \succ_{\mathrm{Reg}} \Box^n\alpha \succ_{\mathrm{Reg}} \cdots$*.*

An interesting point is the closure properties of these classes.

▶ **Proposition 15** ([5])**.** $\mathrm{Reg}[\alpha]$ *is closed under boolean operations.*

However, when α is not ultimately periodic, $\mathrm{Reg}[\alpha]$ is not closed under projection (with respect to $\otimes$) [16]. This is a serious issue if one intends to encode logical theories, what may explain why automata with advice have remained unused for many years. A possible solution, detailed in the next paragraph, is to use ω-regularity instead of finite regularity.

3.2 Non-terminating languages and ω-regularity

Once more, we shall provide a definition in terms of languages, but it could equivalently be stated with ω-automata that read an advice string.

▶ **Definition 16** ([13])**.** $L \subseteq \Sigma^\omega$ is ω*-regular* with advice $\alpha \in \Gamma^\omega$ if there is an ω-regular language $L' \subseteq (\Sigma \times \Gamma)^\omega$ such that $L = \{w \mid w \otimes \alpha \in L'\}$.

▶ **Example 17.**
1. Every ω-regular language is also ω-regular with any advice;
2. $\{\alpha\}$ is ω-regular with advice α.

We denote by $\omega\mathrm{Reg}[\alpha]$ the class of ω-regular languages with advice α. The next definition generalizes ω-regularity with advice to finite-words languages.

▶ **Definition 18** ([13])**.** A language $L \subseteq \Sigma^*$ is *non-terminating regular* with advice $\alpha \in \Gamma^\omega$ if there is an ω-regular language $L' \subseteq ((\Sigma \uplus \Box) \times \Gamma)^\omega$ such that $L = \{w \mid w \otimes \alpha \in L'\}$.

▶ **Example 19.** $\forall n \geq 0$, $\mathrm{Pref}(\alpha[n:])$ is non-terminating regular with advice α.

Let $\mathrm{Reg}^\infty[\alpha]$ be the class of non-terminating regular languages with advice α. It follows from the definitions that $L \in \mathrm{Reg}^\infty[\alpha]$ if and only if $\{w\Box^\omega \mid w \in L\} \in \omega\mathrm{Reg}[\alpha]$. These new definitions increase the expressiveness of advice languages, since $\mathrm{Reg}[\alpha] \subseteq \mathrm{Reg}^\infty[\alpha]$ and the inclusion is strict when α is not ultimately periodic [13]. Furthermore, they solves the lack of closure properties evoked in the end of Subsection 3.1.

▶ **Proposition 20** ([13]). $\mathrm{Reg}^\infty[\alpha]$ *and* $\omega\mathrm{Reg}[\alpha]$ *are closed under boolean operations, cylindrification, and projection (with respect to* $\otimes$*).*

Let us compare infinite words with respect to this ω-regular use of advice. We define the preorders $\preccurlyeq_{\mathrm{Reg}^\infty}$ (resp. $\preccurlyeq_{\omega\mathrm{Reg}}$) based on the inclusion of the Reg^∞ (resp. $\omega\mathrm{Reg}$) classes, and the corresponding notions of degrees. It is not hard to see that ultimately periodic words are again the least $\preccurlyeq_{\mathrm{Reg}^\infty}$- and $\preccurlyeq_{\omega\mathrm{Reg}}$-degree. We now make a non-trivial step towards a generic correspondence between advices, machine transductions, and logic.

▶ **Definition 21.** An ω-regular function f is a (partial) mapping $\Gamma^\omega \to \Delta^\omega$ whose graph $\{w \otimes f(w) \mid w \in \mathrm{dom}(f)\}$ is an ω-regular language.

▶ **Definition 22** (MSO-relabelling)**.** We say that $\alpha \in \Gamma^\omega$ is the image of $\beta \in \Delta^\omega$ under an *MSO-relabelling* if there is a tuple $\mathsf{MSO}[<, \Delta]$-formulas $(\phi_a(x))_{a\in\Gamma}$ such that $\forall n \geq 0$, $\alpha[n] = a$ if and only if $\beta \models \phi_a(n)$.

▶ **Proposition 23.** *The following conditions are equivalent:*
1. $\mathrm{Reg}^\infty[\alpha] \subseteq \mathrm{Reg}^\infty[\beta]$*;*
2. $\omega\mathrm{Reg}[\alpha] \subseteq \omega\mathrm{Reg}[\beta]$*;*
3. α *is the image of* β *under some* ω*-regular function;*
4. α *is the image of* β *under some MSO-relabelling.*

▶ Remark. A word α is the image of β under some Mealy machine if and only if α is the image of β under a *relativized MSO-relabelling*, defined as a relabelling where in the formulas $\phi_a(x)$ every quantification is relativized under x, i.e. of the form $Qy/Y \leq x$.

We obtain in particular $\preccurlyeq_{\omega\mathrm{Reg}} = \preccurlyeq_{\mathrm{Reg}^\infty}$ and $\preccurlyeq_{\mathrm{Reg}} \subsetneq \preccurlyeq_{\mathrm{Reg}^\infty}$ (see Fact 14 and Example 19). To understand its structure, we briefly give a simple necessary condition for $\alpha \preccurlyeq_{\mathrm{Reg}^\infty} \beta$.

▶ **Proposition 24.** *Let* p_γ *be the* subword complexity *function of* γ *[3]. If* $\alpha \in \Gamma^\omega$ *is the image of* $\beta \in \Delta^\omega$ *under some* ω*-regular function, then* $p_\alpha \leq K \times p_\beta$ *for some constant* K*.*

For all $n \geq 1$, there exists a (computable) string α_n such that $p_{\alpha_n} : k \mapsto n^k$. Necessarily $\mathrm{Reg}^\infty[\alpha_n]$ is not contained in any $\mathrm{Reg}^\infty[\beta]$ for $\beta \in \{1, \ldots, n-1\}^\omega$ because $p_\beta(k) \leq (n-1)^k$. This observation shows that the size of the alphabet is an unavoidable parameter for $\preccurlyeq_{\mathrm{Reg}^\infty}$, which is not good news when looking for a robust notion of complexity. The rest of this paper will no longer deal with the preorders defined by languages, but it move towards presentable structures in order to describe a more relevant notion of comparison.

3.3 Advice automatic structures

We now turn to classes of structures that are presentable by advice languages. Following Definition 2 and the notations of [2], we denote by $\mathrm{AutStr}[\alpha]$ the class of $\mathrm{Reg}[\alpha]$-presentable structures, $\mathrm{AutStr}^\infty[\alpha]$ for $\mathrm{Reg}^\infty[\alpha]$-presentable, and $\omega\mathrm{AutStr}[\alpha]$ for $\omega\mathrm{Reg}[\alpha]$-presentable. Such structures are said to be *(ω-)automatic with advice* α. Their study is located a level of abstraction higher than what was done above, since the languages have no longer importance in theirselves, but are only used to encode other objects.

An advice automatic structure can be described "effectively" via a tuple of automata (as for standard automatic structures), and a certain advice α. In fact, the decidability feature of automatic structures is preserved as soon as α is decidable enough.

▶ **Proposition 25** ([2]). *If $\mathfrak{W}^\alpha$ has a decidable MSO-theory, every structure in* $\omega\mathrm{AutStr}[\alpha]$, $\mathrm{AutStr}^\infty[\alpha]$ *or* $\mathrm{AutStr}[\alpha]$ *has a decidable FO-theory.*

Large classes of infinite words with decidable MSO-theory have been described, see e.g. [6] or [19]. We briefly show why the generalization from automatic structures to advice automatic structures can be fruitful (compare the next result to Theorem 5).

▶ **Fact 26** ([13]). $\langle \mathbb{Q}, + \rangle \in \mathrm{AutStr}[\alpha]$ *for some advice α with decidable MSO-theory.*

We now briefly describe basic properties of presentations with advice.

▶ **Fact 27.** *Inclusion of language classes give* $\mathrm{AutStr} \subseteq \mathrm{AutStr}[\alpha] \subseteq \mathrm{AutStr}^\infty[\alpha]$ *and* $\omega\mathrm{AutStr} \subseteq \omega\mathrm{AutStr}[\alpha]$. *Inclusions are equalities if α is ultimately periodic.*

▶ Remark. There is however no immediate argument to deduce $\mathrm{AutStr} \subsetneq \mathrm{AutStr}[\alpha]$ when α is not ultimately periodic. We shall see in Section 5 that this statement is true.

As an immediate consequence of the definitions, $\mathrm{AutStr}^\infty[\alpha] \subseteq \omega\mathrm{AutStr}[\alpha]$ and $\omega\mathrm{AutStr}[\alpha]$ contains uncountable structures, whereas $\mathrm{AutStr}^\infty[\alpha]$ does not. This idea can be refined.

▶ **Theorem 28** ([2]). $\mathrm{AutStr}^\infty[\alpha]$ *is exactly the subclass of countable structures of* $\omega\mathrm{AutStr}[\alpha]$.

The next result shows to what extent the advice contains the seeds of every presentation, and how we generalized the case of automatic structures.

▶ **Proposition 29** ([1]).
1. $\mathfrak{A} \in \omega\mathrm{AutStr}[\alpha]$ *if and only if $\mathfrak{A}$ is S-interpretable in $\mathfrak{W}^\alpha$;*
2. $\mathfrak{A} \in \mathrm{AutStr}^\infty[\alpha]$ *if and only if $\mathfrak{A}$ is FS-interpretable in $\mathfrak{W}^\alpha$.*

▶ Remark ([1]). If the presentation is injective and the encoding is alphabet binary, the resulting interpretation can be done 1-dimensional and injective.

Dealing directly with $\mathrm{Reg}[\alpha]$-presentations seems more difficult, since basic properties lack to this class of languages. We now show $\mathrm{AutStr}^\infty[\alpha] = \mathrm{AutStr}[\alpha]$, hence the expression "advice automatic structure" is not ambiguous. To give an intuition of the proof, we note that an ω-automaton performs an infinite run on $w \otimes \alpha$ (for w finite) in two steps: first, it follows a finite run on $w \otimes \alpha[: |w|]$, then it checks some ω-regularity on $\square^\omega \otimes \alpha[|w| :] \simeq \alpha[|w| :]$. Basically, the ω-regularity feature is only used on suffixes of the advice. On the other hand, a automaton for $\mathrm{Reg}[\alpha]$ is blind to the ω-future. We show that it can nevertheless look at some "finite amount of future" and deduce corresponding ω-regularity on the suffixes. A key idea is that since the advice is *fixed*, so are several properties of its suffixes.

▶ **Theorem 30.** *Let L be an ω-regular language and $\alpha \in \Gamma^\omega$ a fixed word. There is a (finite words) regular language L' and $N \geq 0$ such that for all $n \geq N$, $\alpha[n :] \in L$ if and only if $\alpha[n :]$ has a finite prefix in L'. Furthermore, if L can be described by an $\mathsf{FO}[<, \Gamma]$-sentence, L' can be described by an $\mathsf{FO}[<, \Gamma]$-sentence as well.*

Proof sketch. The case of FO is treated via equivalence with LTL, known [15] as Kamp's Theorem. For MSO in general, we deduce the result from the work of A.L. Semenov [19]. ◀

Corollary 31 will formalize our intuition that terminating automata can check ω-regular properties on suffixes. It thus enables us to explicit the relationships between $\mathrm{Reg}[\alpha]$ and $\mathrm{Reg}^\infty[\alpha]$, and between $\mathrm{AutStr}[\alpha]$ and $\mathrm{AutStr}^\infty[\alpha]$.

▶ **Corollary 31.** *Let $L \subseteq \Gamma^\omega$ be an ω-regular language and $\alpha \in \Gamma^\omega$. There is a function $f : \mathbb{N} \to \mathbb{N}$ such that $\{0^n\Box^{f(n)} \mid \alpha[n:] \in L\} \in \mathrm{Reg}[\alpha]$.*

▶ **Corollary 32.** *Let $\alpha \in \Gamma^\omega$. For every language $L \in \mathrm{Reg}^\infty[\alpha]$, there is a function $f : \mathbb{N} \to \mathbb{N}$ such that $\{w\Box^{f(|w|)} \mid w \in L\} \in \mathrm{Reg}[\alpha]$.*

▶ **Corollary 33.** *For every advice α,* $\mathrm{AutStr}[\alpha] = \mathrm{AutStr}^\infty[\alpha]$.

▶ Remark. Thanks to these results, we also managed to build a normal form for MSO-formulas (with free variables) when interpreted in a fixed word model, see [9].

4 Complexity of advices when describing structures

After the first results of the previous section on advice automatic structures, we are now able to understand which preorder they describe over infinite words. Corollary 33 implies in particular that $\mathrm{AutStr}[\alpha] \subseteq \mathrm{AutStr}[\beta]$ if and only if $\mathrm{AutStr}^\infty[\alpha] \subseteq \mathrm{AutStr}^\infty[\beta]$. The objective of this section is to show equivalence with $\omega\mathrm{AutStr}[\alpha] \subseteq \omega\mathrm{AutStr}[\beta]$ and give several other characterizations. The climax lies in Theorem 39 and Theorem 45, where we relate our notions to well-known logical transformations and finite transducers.

▶ **Definition 34.** A (k-copying) *MSO-transduction (MSOT)* from Δ^ω to Γ^ω is a tuple of $\mathsf{MSO}[<, \Delta]$-formulas with free first-order variables.

$$(\phi_1^a(x))_{a\in\Gamma} \dots (\phi_k^a(x))_{a\in\Gamma}, (\phi_{i,j}^<(x,y))_{1\le i,j\le k})$$

The semantics of an MSOT τ is defined as that of an MSO-interpretation in k disjoint copies of a host word structure. More precisely, the structure $I_\tau(\mathfrak{W}^\beta)$ (not necessarily a word) has signature $\{<, (P_a)_{a\in\Gamma}\}$ and is defined as follows:

- $\mathrm{dom}(I_\tau(\mathfrak{W}^\beta)) = \bigcup_{1\le i\le k}\{(n,i) \mid$ there is $a \in A$ such that $\beta \models \phi_i^a(n)\}$;
- if $(n,i) \in \mathrm{dom}(I_\tau(\mathfrak{W}^\beta))$, then $(n,i) \in P_a$ if and only if $\beta \models \phi_i^a(n)$;
- if $(m,j) \in \mathrm{dom}(I_\tau(\mathfrak{W}^\beta))$, then $(n,i) < (m,j)$ if and only if $\mathcal{U} \models \phi_{i,j}^<(n,m)$.

Since we are interested in transformations between words, we only consider the case when $I_\tau(\mathfrak{W}^\beta)$ is a word structure (what is syntactically definable by adding an $\mathsf{MSO}[<,\Delta]$-sentence for the domain). Each MSO-transduction τ then realizes a (partial) function $\tau : \Delta^\omega \to \Gamma^\omega$ whose domain is $\{\beta \in \Delta^\omega \mid I_\tau(\mathfrak{W}^\beta)$ is (isomorphic to) a word structure$\}$, the image $\tau(\beta)$ of β being the unique α such that $I_\tau(\mathfrak{W}^\beta) \simeq \mathfrak{W}^\alpha$.

The reader is asked to keep in mind that MSOT define a certain class of functions on infinite strings, even if our main concern is only the existence of a transduction between two fixed words. We write $\alpha \preccurlyeq_{\mathsf{MSOT}} \beta$ if there is a MSO-transduction τ such that $\tau(\beta) = \alpha$.

▶ Remark. MSO-relabelings (see Definition 22), relativized MSO-relabelings, and 1- dimensional MSO-interpretations can all seen as syntactical fragments of 1-copying MSOT.

▶ Remark. Even if MSO-interpretations in general are not closed under composition, it is the case of MSOT [4]. Thus $\preccurlyeq_{\mathsf{MSOT}}$ is transitive, and is even a preorder over infinite words.

▶ **Example 35.**

1. If $\alpha \preccurlyeq_{\mathrm{Reg}^\infty} \beta$ then $\alpha \preccurlyeq_{\mathsf{MSOT}} \beta$ (thus $\preccurlyeq_{\mathsf{MSOT}}$ is a more generic notion of comparison than the preorders of Section 3, we shall see that the increase of power is strict);
2. modifying a finite part of α does not change its MSOT-degree;
3. if w is a finite word, we denote by $\widetilde{w}$ its mirror image; if $\alpha := w_1\#w_2\#\dots \in (\Gamma^*\#)^\omega$, let $\widetilde{\alpha} := \widetilde{w_1}\#\widetilde{w_2}\#\dots$; then $\widetilde{\alpha} \preccurlyeq_{\mathsf{MSOT}} \alpha$.

4.1 From automatic structures to MSO-transductions

When searching a complete structure of an advice, a naive idea is that $\mathfrak{W}^\alpha \in \mathrm{AutStr}^\infty[\beta]$ if and only if $\mathrm{AutStr}^\infty[\alpha] \subseteq \mathrm{AutStr}^\infty[\beta]$. However, this statement will turn out to be false. We need a stronger object that is presented in Definition 36.

▶ **Definition 36** ([14]). Let $\mathfrak{A} = \langle A, R_1 \dots R_n \rangle$ be a structure, we define its *weak powerset structure* $\mathcal{P}^f(\mathfrak{A})$ as the structure $\langle \mathcal{P}^f(A), R'_1 \dots R'_n, \subseteq \rangle$ where:

- $\mathcal{P}^f(A)$ is the weak powerset (set of finite subsets) of A;
- $\subseteq$ is the inclusion relation on $\mathcal{P}^f(A)$;
- $R'_i(A_1, \dots, A_{r_i})$ holds in $\mathcal{P}^f(\mathfrak{A})$ if and only if $A_1, \dots A_{r_i}$ are singletons $\{a_1\}, \dots, \{a_{r_i}\}$ and $R_i(a_1, \dots a_{r_i})$ holds in $\mathfrak{A}$.

▶ Remark. $\mathfrak{A}$ is FS-interpretable in $\mathfrak{B}$ if and only if $\mathfrak{A}$ is FO-interpretable in $\mathcal{P}^f(\mathfrak{B})$.

▶ **Fact 37.** $\mathrm{AutStr}^\infty[\alpha]$ *is the class of structures FO-interpretable in* $\mathcal{P}^f(\mathfrak{W}^\alpha)$ *(see Proposition 29). We have* $\mathrm{AutStr}^\infty[\alpha] \subseteq \mathrm{AutStr}^\infty[\beta]$ *if and only if* $\mathcal{P}^f(\mathfrak{W}^\alpha) \in \mathrm{AutStr}^\infty[\beta]$.

This result provides a characterization which is abstract and, in some respects, trivial. Nevertheless, we get the intuition that powerset structures are a key notion to understand advice automaticity. In the sequel, a (Δ-labelled) tree structure has the form $\langle A, <, (P_a)_{a \in \Delta} \rangle$ where the domain A is a prefix-closed subset of $\{0,1\}^*$, $w < w'$ holds whenever w is a prefix of w' and the P_a label the nodes of A with $a \in \Delta$. Word structures are particular trees.

▶ **Theorem 38** ([14], Corollary 4.4). *Let* $\mathfrak{A}$ *a structure and* $\mathfrak{T}$ *a tree structure. If* $\mathcal{P}^f(\mathfrak{A})$ *is* 1*-dimensionally injectively FS-interpretable in* $\mathfrak{T}$*, then* $\mathfrak{A}$ *is* 1*-dimensionally injectively WMSO-interpretable in* $\mathfrak{T}$.

In the case of advice automatic structures, Theorem 38 is at the same time too generic and too restrictive. On the one hand, we only use interpretations in word structures $\mathfrak{W}^\alpha$. On the other hand, we need arbitrarily dimensional FS-interpretations, and they are not supposed to be injective. We will manage to meet this conditions, up to a slight modification of the advice, and the WMSO-interpretation will be transformed into a more generic MSOT.

▶ **Theorem 39.** *The following conditions are equivalent:*

1. $\omega\mathrm{AutStr}[\alpha] \subseteq \omega\mathrm{AutStr}[\beta]$*;*
2. $\mathrm{AutStr}^\infty[\alpha] \subseteq \mathrm{AutStr}^\infty[\beta]$*;*
3. $\alpha \preccurlyeq_{MSOT} \beta$.

Proof sktech. We use Proposition 29 several times. The way from 1. to 2. is a consequence of Theorem 28. If 2. holds, we show that $\mathcal{P}^f(\mathfrak{W}^\alpha)$ has an injective binary $\mathrm{Reg}^\infty[\beta']$-presentation for some infinite word β' so that $\beta' \preccurlyeq_{\mathsf{MSOT}} \beta$. As remarked above, $\mathcal{P}^f(\mathfrak{W}^\alpha)$ is thus 1-dimensionally injectively FS-interpretable in the tree $\mathfrak{W}^{\beta'}$, hence Theorem 38 provides a 1-dimensionally WMSO-interpretation of $\mathfrak{W}^\alpha$ in $\mathfrak{W}^{\beta'}$, what implies $\alpha \preccurlyeq_{\mathsf{MSOT}} \beta'$. Composing MSOT concludes that $\alpha \preccurlyeq_{\mathsf{MSOT}} \beta$. If 3. is true and $\mathfrak{A}$ is S-interpretable in $\mathfrak{W}^\alpha$, then $\mathfrak{A}$ is S-interpretable in $\mathfrak{W}^\beta$ by some composition argument. ◀

As a consequence, all the preorders defined by advice-presentable structures converge towards the same comparison via MSO-transductions. This point gives a deep theoretical meaning to their study. Another virtue of Theorem 39 is the ability to translate immediately the results of Example 35 in terms of advice automatic structures.

▶ **Example 40.**

1. If $\alpha \preccurlyeq_{\mathrm{Reg}^\infty} \beta$ then $\mathrm{AutStr}[\alpha] \subseteq \mathrm{AutStr}[\beta]$;
2. modifying a finite part of α does not modify $\mathrm{AutStr}[\alpha]$;
3. if $\alpha \in (\Gamma^*\#)^\omega$, then $\mathrm{AutStr}[\alpha] = \mathrm{AutStr}[\widetilde{\alpha}]$.

4.2 An equivalent computational model: two-way transducers

We will complete our parallel with transductions via an equivalent simple machine model. Furthermore, it will be very useful to describe the structural properties of the preorder.

▶ **Definition 41.** A *two-way finite transducer* (2WFT) is a 6-tuple $(Q, q_0, \Delta \uplus \{\vdash\}, \Gamma, \delta, \theta)$ where Q is the finite set of states, $q_0 \in Q$ is initial, Δ is the input alphabet, Γ is the output alphabet, $\delta : Q \times (\Delta \uplus \{\vdash\}) \to Q \times \{\triangleleft, \triangleright\}$ is the (partial) transition function, and $\theta : Q \times (\Delta \uplus \{\vdash\}) \to \Gamma^*$ is the (partial) output function.

A 2WFT has a two-way read-only input tape and a one-way output mechanism. The component $\{\triangleleft, \triangleright\}$ determines the left or right move of the head on the input tape. When the 2WFT is given $\beta \in \Delta^\omega$ as an input word, this tape contains $\vdash \beta$ (adding a symbol $\vdash$ helps the transducer to notice the beginning of its input when going left). The definition of the (partial) function $\Delta^\omega \to \Gamma^\omega$ realized the 2WFT follows like for Mealy machines.

▶ Remark. The transducer is said to be *one-way* (1WFT, or just finite transducer) if all its transitions are of the form $(q, \triangleright)$. Mealy machines are a particular case of 1WFT.

▶ **Example 42.** There is a three-state 2WFT outputting $\widetilde{\alpha}$ on every $\alpha \in (\Gamma\#)^\omega$. Its behavior is the following: scan a maximal #-free block, read it in a reversed way while outputting, then output # and move to the next block.

When considering definable *functions* between *finite* strings, a well-known equivalence holds between MSOT and 2WFT (Theorem 43). The definitions of MSOT and 2WFT have to be slightly sharpened to get the exact correspondence, see details in [11].

▶ **Theorem 43** ([11])**.** *(Partial) functions over finite words $\Delta^* \to \Gamma^*$ definable by MSOT are the (partial) functions realized by 2WFT.*

Fairly recently, this result was extended to functions between infinite strings, but some complications quickly appear: deciding the validity of MSO-sentences is not always possible without reading the (variable) input entirely. Thus 2WFT alone are not powerful enough and they need extra features like ω-regular lookahead, i.e. ability to check instantly ω-regular properties of the suffixes of the input starting in the position of the reading head.

▶ **Theorem 44** ([4])**.** *(Partial) functions over infinite words $\Delta^\omega \to \Gamma^\omega$ definable by MSOT are the (partial) functions realized by 2WFT with ω-regular lookahead whose runs always visit the whole input string.*

When looking closely at Theorem 43 and Theorem 44 in the light of our previous results, a question arises naturally: it is possible to get rid of the lookaheads when fixing the input infinite word? Indeed, we have always considered transformations from a *fixed* word and we noticed in Subsection 3.3 that this restriction simplified certain notions. Theorem 45 gives a positive answer. This involved result is not a direct consequence of Theorem 44, since we are not aware of a simple manner to remove the ω-lookaheads when fixing the input.

▶ **Theorem 45.** *$\alpha \preccurlyeq_{MSOT} \beta$ if and only if $\alpha \preccurlyeq_{2WFT} \beta$.*

Proof sketch. If $\alpha \preccurlyeq_{\mathsf{2WFT}} \beta$, the result follows from Theorem 44. Indeed the transformation can be computed by some 2WFT (with a trivial ω-lookahead) whose run visits the whole input. Assume now that $\alpha \preccurlyeq_{\mathsf{MSOT}} \beta$. It follows from [4] that α can be computed from β by an *ω-streaming string transducer (SST)*. We provide a rather long argument to show that an SST can be transformed into a 2WFT with a *lookbehind* feature, when the input word is fixed. Lastly, the lookbehind can be removed by some standard techniques (a lookbehind only deals with a *finite* part of the input, which is not the case of an ω-lookahead). ◀

5 The two-way transductions hierarchy

We initiate in this section a study of the previous two-way transductions between infinite words. It can equivalently be seen as the preorder defined by MSOT, or classes AutStr[α], AutStr$^\infty$[α] and ωAutStr[α]; but the 2WFT formulation is - as predicted above - the easiest way to deduce interesting statements. We shall use the term *2WFT hierarchy* to describe the ordered set of 2WFT-degrees (i.e. equivalence classes of $\preccurlyeq_{\mathsf{2WFT}} \cap \succcurlyeq_{\mathsf{2WFT}}$).

A more or less similar work has been done in [10], with the relation $\preccurlyeq_{\mathsf{1WFT}}$ defined by computability via 1WFT. This definition clearly describes a preorder. Even if no previous research exists on the 2WFT hierarchy, we shall see that several results on the 1WFT can be adapted in our context, after a variable amount of work. Note that $\preccurlyeq_{\mathsf{1WFT}} \subseteq \preccurlyeq_{\mathsf{2WFT}}$.

▶ **Proposition 46.**
1. *There are uncountably many distinct 2WFT-degrees;*
2. *a set of 2WFT-degree has an upper bound if and only if it is countable;*
3. *the 2WFT hierarchy has no greatest degree;*
4. *every 2WFT-degree contains a binary string.*

▶ Remark. Considering binary strings is thus sufficient to describe all the degrees. Comparing this result with Proposition 24 shows that the preorder $\preccurlyeq_{\mathrm{Reg}^\infty}$ defined by MSO-relabelings is strictly weaker than $\preccurlyeq_{\mathsf{2WFT}} = \preccurlyeq_{\mathsf{MSOT}}$.

As a consequence of Proposition 46, the 2WFT hierarchy is not trivial. We now show that it is fine-grained enough to distinguish ultimately periodic words.

▶ **Proposition 47.** *Ultimately periodic words are the least 2WFT-degree.*

This result shows, through the equivalences of Section 4, that non-trivial advices *strictly* increase the class of presentable structures (what had no reason to be obvious).

▶ **Corollary 48.** *$\mathcal{P}^f(\mathfrak{W}^\alpha)$ is automatic if and only if α is ultimately periodic.*

▶ Remark. There are non-ultimately periodic sequences α such that $\mathfrak{W}^\alpha$ is automatic [6]. However, no sufficient and necessary condition is known to describe such sequences.

We now turn to a more involved statement. A sequence β is said to be *prime* if it is a minimal but non-trivial word. Formally, β non-ultimately periodic is prime in the 2WFT hierarchy if for all $\alpha \preccurlyeq_{\mathsf{2WFT}} \beta$, either $\beta \preccurlyeq_{\mathsf{2WFT}} \alpha$ or α is ultimately periodic. The existence of prime sequences shows in particular that the 2WFT hierarchy is not dense.

▶ **Theorem 49.** *The sequence $\pi := \prod_{n=0}^{\infty} 0^n 1$ is prime in the 2WFT hierarchy.*

Proof sketch. Our work is to show that if $\alpha \preccurlyeq_{\mathsf{2WFT}} \pi$, then $\alpha \preccurlyeq_{\mathsf{1WFT}} \pi$. Now, since π is prime in the 1WFT-hierarchy [10], either $\pi \preccurlyeq_{\mathsf{1WFT}} \alpha$ or α is in the least 1WFT-degree, which is also the set of ultimately periodic words. ◀

Classifying all infinite strings may neither be relevant nor useful in practice. We now look at two particular classes of infinite words closed under 2WFT transformations.

▶ **Proposition 50** (subhierarchies)**.**
1. *If $\alpha \preccurlyeq_{\mathsf{2WFT}} \beta$ and if β is computable, then α is computable;*
2. *if $\alpha \preccurlyeq_{\mathsf{2WFT}} \beta$ and if $\mathfrak{W}^\beta$ has a decidable MSO-theory, so has $\mathfrak{W}^\alpha$.*

▶ **Fact 51.** *The string π has a decidable MSO-theory (see e.g. [6]).*

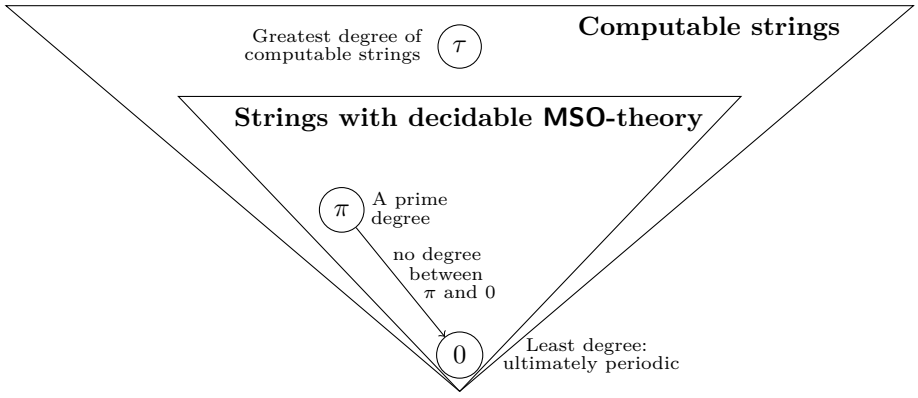


Figure 1 An partial look on the 2WFT hierarchy

Table 1 Equivalent definitions for preorders over ω-words

Advice	Reg	Reg^∞ ωReg	AutStr AutStr^∞ ωAutStr
Logic	rel. MSO-relabelings	MSO-relabelings	MSOT
Machine	Mealy machines	ω-regular functions	2WFT

▶ **Proposition 52** (adapted from [10] for 1WFT). *There exists a greatest degree τ of computable strings in the 2WFT hierarchy.*

▶ **Fact 53.** *The MSO theory of τ is not decidable.*

Figure 1 summarizes the previous results. Note that the 2WFT-degree of ultimately periodic sequences, the 2WFT-degree of π and the 2WFT-degree of τ have to be distinct. Several challenging issues naturally arise about the structure of the 2WFT hierarchy and its subhierarchies. Among others, an interesting question is to describe the degrees of well-known sequences with decidable MSO-theory, for instance morphic words [6].

6 Conclusion and outlook

Preorders of advices, logic and transducers. Our first concern in this paper was the study of various preorders over infinite words, related to the notion of advice strings. The results draw a generic correspondance between definability with advice, logical transductions and machine transductions. Table 1 summarizes this philosophy in an elegant way, note that the notion of (relativized) MSO-relabelings is less standard than MSOT. The gap between MSO-relabelings and MSOT shows that having basic knowledge on the languages is far from being sufficient to understand the richness of presentable structures.

A meaningful hierarchy of infinite words. Two-way transductions appear here to be more basic than relations defined by one-way machines, since they are clearly motivated by logical issues. Furthermore, it fits our informal conditions to be a "good" complexity measure over infinite words. A more involved study of the 2WFT hierarchy may help classifying certain hierarchies of structures, or even understand standard automatic presentations. We recall that such transductions over infinite words are (rather) unexplored.

References

1. Faried Abu Zaid. *Algorithmic solutions via model theoretic interpretations*. PhD thesis, RWTH Aachen University, 2016. doi:10.18154/RWTH-2017-07663.
2. Faried Abu Zaid, Erich Grädel, and Frederic Reinhardt. Advice Automatic Structures and Uniformly Automatic Classes. In *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, 2017.
3. Jean-Paul Allouche and Jeffrey Shallit. *Automatic sequences: theory, applications, generalizations*. Cambridge university press, 2003.
4. Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, pages 65–74. IEEE Computer Society, 2012.
5. Robert M. Baer and Edwin H. Spanier. Referenced automata and metaregular families. *Journal of Computer and System Sciences*, 3(4):423–446, 1969.
6. Vince Bárány. A hierarchy of automatic ω-words having a decidable MSO theory. *RAIRO-Theoretical Informatics and Applications*, 42(3):417–450, 2008.
7. Aleksandrs Belovs. Some algebraic properties of machine poset of infinite words. *RAIRO-Theoretical Informatics and Applications*, 42(3):451–466, 2008.
8. Achim Blumensath and Erich Grädel. Automatic structures. In *Logic in Computer Science, 2000. Proceedings. 15th Annual IEEE Symposium on*, pages 51–62. IEEE, 2000.
9. Gaëtan Douéneau-Tabot. Comparing the power of advice strings: a notion of complexity for infinite words. *arXiv preprint arXiv:1801.04908*, 2018.
10. Jörg Endrullis, Jan Willem Klop, Aleksi Saarela, and Markus Whiteland. Degrees of transducibility. In *International Conference on Combinatorics on Words*, pages 1–13. Springer, 2015.
11. Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic (TOCL)*, 2(2):216–254, 2001.
12. Bakhadyr Khoussainov, André Nies, Sasha Rubin, and Frank Stephan. Automatic structures: richness and limitations. In *Logic in Computer Science, 2004. Proceedings of the 19th Annual IEEE Symposium on*, pages 44–53. IEEE, 2004.
13. Alex Kruckman, Sasha Rubin, John Sheridan, and Ben Zax. A Myhill-Nerode theorem for automata with advice. In *GandALF*, pages 238–246, 2012.
14. Christof Löding and Thomas Colcombet. Transforming structures by set interpretations. *Logical Methods in Computer Science*, 3, 2007.
15. Alexander Rabinovich. A proof of Kamp's theorem. *Logical Methods in Computer Science*, 10(1), 2014.
16. Frédéric Reinhardt. Automatic structures with parameters. Master's thesis, Rheinisch-Westfälischen Technischen Hochschule Aachen, 2013.
17. Arto Salomaa. On finite automata with a time-variant structure. *Information and Control*, 13(2):85–98, 1968.
18. Dana Scott. Some definitional suggestions for automata theory. *Journal of Computer and System Sciences*, 1(2):187–212, 1967.
19. Aleksei Lvovich Semenov. Logical theories of one-place functions on the set of natural numbers. *Mathematics of the USSR-Izvestiya*, 22(3):587, 1984.
20. Todor Tsankov. The additive group of the rationals does not have an automatic presentation. *The Journal of Symbolic Logic*, 76(04):1341–1351, 2011.

Resynchronizing Classes of Word Relations

María Emilia Descotte
LaBRI, Université de Bordeaux

Diego Figueira
CNRS, LaBRI, Université de Bordeaux

Gabriele Puppis
CNRS, LaBRI, Université de Bordeaux

Abstract

A natural approach to define binary word relations over a finite alphabet $\mathbb{A}$ is through two-tape finite state automata that recognize regular languages over $\{1,2\} \times \mathbb{A}$, where (i,a) is interpreted as reading letter a from tape i. Accordingly, a word $w \in L$ denotes the pair $(u_1, u_2) \in \mathbb{A}^* \times \mathbb{A}^*$ in which u_i is the projection of w onto i-labelled letters. While this formalism defines the well-studied class of Rational relations (*a.k.a.* non-deterministic finite state transducers), enforcing restrictions on the reading regime from the tapes, which we call *synchronization*, yields various sub-classes of relations. Such synchronization restrictions are imposed through regular properties on the projection of the language onto $\{1,2\}$. In this way, for each regular language $C \subseteq \{1,2\}^*$, one obtains a class $\textsc{Rel}(C)$ of relations. Regular, Recognizable, and length-preserving rational relations are all examples of classes that can be defined in this way.

We study the problem of containment for synchronized classes of relations: given $C, D \subseteq \{1,2\}^*$, is $\textsc{Rel}(C) \subseteq \textsc{Rel}(D)$? We show a characterization in terms of C and D which gives a decidability procedure to test for class inclusion. This also yields a procedure to re-synchronize languages from $\{1,2\} \times \mathbb{A}$ preserving the denoted relation whenever the inclusion holds.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases synchronized word relations, containment, resynchronization

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.123

Funding Work supported by ANR project DELTA, grant ANR-16-CE40-0007, and LIA INFINIS.

1 Introduction

We study are relations of finite words, that is, binary relations $R \subseteq \mathbb{A}^* \times \mathbb{A}^*$ for a finite alphabet $\mathbb{A}$. The study of these relations dates back to the works of Büchi, Elgot, Mezei, and Nivat in the 1960s [4, 8, 13], with much subsequent work done later (*e.g.*, [2, 6]). Most of the investigations focused on extending the standard notion of regularity from languages to relations. This effort has followed the long-standing tradition of using equational, operational, and descriptive formalisms – that is, finite monoids, automata, and regular expressions – for describing relations, and gave rise to three different classes of relations: the *Recognizable*, the *Automatic* (*a.k.a. Regular* [2] or *Synchronous* [6]), and the *Rational* relations.

The above classes of relations can be seen as three particular examples of a much larger (in fact infinite) range of possibilities, where relations are described by special languages over extended alphabets, called *synchronizing languages* [10]. Intuitively, the idea is to describe a binary relation by means of a two-tape automaton with two heads, one for each tape, which can move independently one of the other. In the basic framework of synchronized relations, one lets each head of the automaton to either move right or stay in the same

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 123; pp. 123:1–123:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

position. In addition, one can constrain the possible sequences of head motions by a suitable regular language $C \subseteq \{1,2\}^*$. In this way, each regular language $C \subseteq \{1,2\}^*$ induces a class of binary relations, denoted $\textsc{Rel}(C)$, which is contained in the class of Rational relations (due to Nivat's Theorem [13]). For example, the class of Recognizable, Automatic, and Rational relations are captured, respectively, by the languages $C_{\text{Rec}} = \{1\}^* \cdot \{2\}^*$, $C_{\text{Aut}} = \{12\}^* \cdot \{1\}^* \cup \{12\}^* \cdot \{2\}^*$, and $C_{\text{Rat}} = \{1,2\}^*$. However, it should be noted that other well-known subclasses of rational relations, such as deterministic or functional relations, are not captured by notion of synchronization. In general, the correspondence between a language $C \subseteq \{1,2\}^*$ and the induced class $\textsc{Rel}(C)$ of synchronized relations is not one-to-one: it may happen that different languages C, D induce the same class of synchronized relations. There are thus fundamental questions that arise naturally in this framework: *When do two classes of synchronized relations coincide, and when is one contained in the other?* Our contribution is a precise algorithmic answer to this type of questions.

More concretely, given a binary alphabet $\mathbb{2} = \{1,2\}$ and another finite alphabet $\mathbb{A}$, a word $w \in (\mathbb{2} \times \mathbb{A})^*$ is said to *synchronize* the pair $(w_1, w_2) \in \mathbb{A}^* \times \mathbb{A}^*$ if, for both $i = 1, 2$, w_i is the projection of w on $\mathbb{A}$ restricted to the positions marked with i. For short, we denote this by $[\![w]\!] = (w_1, w_2)$ – *e.g.*, $[\![(1,a)(1,b)(2,b)(1,a)(2,c)]\!] = (aba, bc)$. According to this definition, every word over $\mathbb{2} \times \mathbb{A}$ synchronizes a pair of words over $\mathbb{A}$, and every pair of words over $\mathbb{A}$ is synchronized by (perhaps many) words over of $\mathbb{2} \times \mathbb{A}$. This notion is readily lifted to languages: a language $L \subseteq (\mathbb{2} \times \mathbb{A})^*$ synchronizes the relation $[\![L]\!] = \{[\![w]\!] \mid w \in L\} \subseteq \mathbb{A}^* \times \mathbb{A}^*$. For example, $[\![((1,a)(2,a) \cup (1,b)(2,b))^*]\!]$ denotes the equality relation over $\mathbb{A} = \{a,b\}$.

In this setup, one can define classes of relations by restricting the set of admitted synchronizations. The natural way of doing so is to fix a language $C \subseteq \mathbb{2}^*$, called *control language*, and let L vary over all regular languages over the alphabet $\mathbb{2} \times \mathbb{A}$ whose projections onto $\mathbb{2}$ are in C. Thus, for every regular $C \subseteq \mathbb{2}^*$, there is an associated class $\textsc{Rel}(C)$ of C-controlled relations, namely, relations synchronized by regular languages $L \subseteq (\mathbb{2} \times \mathbb{A})^*$ whose projection onto $\mathbb{2}$ are in C. Clearly, $C \subseteq D \subseteq \mathbb{2}^*$, implies $\textsc{Rel}(C) \subseteq \textsc{Rel}(D)$, but the converse does not hold: while $\textsc{Rel}(C_{\text{Rec}}) = \textit{Recognizable} \subseteq \textit{Automatic} = \textsc{Rel}(C_{\text{Aut}})$, we have $C_{\text{Rec}} \not\subseteq C_{\text{Aut}}$. Moreover, as we have mentioned earlier, different control languages may induce the same class of synchronized relations. For example, once again, the class of Recognizable relations is induced by the control language $C_{\text{Rec}} = \{1\}^*\{2\}^*$, but also by $C'_{\text{Rec}} = \{1\}^*\{2\}^*\{1\}^*$, and the class of Automatic relations is induced by $C_{\text{Aut}} = \{12\}^* \cdot \{1\}^* \cup \{12\}^* \cdot \{2\}^*$, or equally by $C'_{\text{Aut}} = \{21\}^* \cdot \{1\}^* \cdot \{2\}^*$. This 'mismatch' between control languages and induced classes of relations gives rise to the following algorithmic problem.

Class Containment Problem	
Input:	Two regular languages $C, D \subseteq \mathbb{2}^*$
Question:	Is $\textsc{Rel}(C) \subseteq \textsc{Rel}(D)$?

Note that the above problem is different from the (C, D)-membership problem on synchronized relations, which consists in deciding whether $R \in \textsc{Rel}(D)$ for a given $R \in \textsc{Rel}(C)$, and which can be decidable or undecidable depending on C, D [5]. The Class Containment Problem can be seen as the problem of whether *every* C-controlled regular language L has a D-controlled regular language L' so that $[\![L]\!] = [\![L']\!]$. It was proved in [10] that this problem is decidable for some particular instances of D, namely, for $D =$ *Recognizable*, *Automatic*, *Length-preserving* or *Rational*. More specifically, given a regular language C over the binary alphabet $\mathbb{2}$, it is decidable whether $\textsc{Rel}(C)$ is contained or not in *Recognizable* (respectively, *Automatic*, *Length-preserving* and *Rational*). Our main contribution is a procedure for deciding the Class Containment Problem in full generality, i.e. for arbitrary C and D.

▶ **Main Theorem.** *The Class Containment Problem is decidable.*

In addition, our results show that, for positive instances (C, D), one can effectively transform any regular C-controlled language L into a regular D-controlled language L' so that $[\![L]\!] = [\![L']\!]$. By 'effectively transform' we mean that one can receive as input an automaton (or a regular expression) for L and produce an automaton (or a regular expression) for L'. In particular, we show a normal form of control languages, implying that every synchronized class can be expressed through a control language of star-height at most 1.

Related work. The formalization of a framework in which one can describe classes of word relations by means of synchronization languages is quite recent [10]. As already mentioned, the class containment problem was only addressed for the classes of Recognizable, Automatic and Rational relations, for which several characterizations have been proposed [10]. The formalism of synchronizations has been extended beyond rational relations by means of semi-linear constraints [9] in the context of path querying languages for graph databases.

The paper [3] studies relations with origin information, as induced by non-deterministic (one-way) finite state transducers. Origin information can be seen as a way to describe a synchronization between input and output words – somehow in the same spirit of our synchronization languages – and was exploited to recover decidability of the equivalence problem for transducers. The paper [11] pursues further this principle by studying "distortions" of the origin information, called resynchronizations. Despite the similar terminology and the connection between origins and synchronizing languages, the problems studied in [3, 11] are of rather different nature than our Class Containment Problem.

Organization. After the preliminaries on subclasses of regular languages, we define in Section 3 the framework of synchronized relations. Section 4 provides a roadmap with the three key ingredients of our characterization. Sections 5, 6 and 7 contain the technical details for these main ingredients. In Section 8 we discuss the computability of the characterization.

2 Preliminaries

We denote by $\mathbb{N}, \mathbb{Q}$ the sets of non-negative integers and rationals. We use standard interval notation as in, for example, $(a, b]_{\mathbb{Q}} = \{c \in \mathbb{Q} \mid a < c \leq b\}$. $\mathbb{A}, \mathbb{B}$ denote arbitrary finite alphabets, and $\mathbb{2}$ the special binary alphabet $\{1, 2\}$.

Words and shuffles. For a word $w \in \mathbb{A}^*$, $|w|$ is its length, and $|w|_a$ is the number of occurrences of symbol a in w. We denote by $w[i, j]$ the factor of w between positions i and j (included), for $1 \leq i \leq j \leq |w|$, and we write $w[i]$ for $w[i, i]$. We will also make use of the **shuffle** operation, which maps a finite set of words $w_1, \ldots, w_n$ to the language $\mathrm{shuffle}\{w_1, \ldots, w_n\}$ of all words w for which there is a partition $I_1, \ldots, I_n$ of $[1, |w|]$ so that each w_i is the projection of w onto I_i. For example, $\mathrm{shuffle}\{ab, cd\} = \{abcd, cdab, acbd, acdb, \ldots\}$.

Parikh image. The **Parikh image** of a word w over $\mathbb{A}$ is the tuple $\pi(w)$ associating each symbol $a \in \mathbb{A}$ to its number of occurrences $|w|_a$ in w. We will mostly use Parikh images for words over $\mathbb{2}^*$, which are thus pairs $\pi(w) = (|w|_1, |w|_2)$. We naturally extend this to languages by letting $\pi(L) \stackrel{\text{def}}{=} \{\pi(w) \mid w \in L\}$ $(\subseteq \mathbb{N}^2)$. For $\bar{x}, \bar{x}_1, \ldots, \bar{x}_n \in \mathbb{N}^2$, we denote by $\langle \bar{x}, P \rangle$ the 2-dimensional linear set $\{\bar{x} + \alpha_1 \bar{x}_1 + \cdots + \alpha_n \bar{x}_n \mid \alpha_1, \ldots, \alpha_n \in \mathbb{N}\}$, and call $\bar{x} \in \mathbb{N}^2$ its **basis** and $\bar{x}_1, \ldots, \bar{x}_n$ its **periods**. A **semi-linear** set is a finite union of linear sets.

Regular languages. We use standard notation for regular expressions without complement, namely, for expressions build up from the empty set, the empty word ε and the symbols $a \in \mathbb{A}$, using the operations $\cdot$, $\cup$, and $(\,)^*$. For economy of space and clarity we also use the abbreviated notation $(\,)^k$, $(\,)^{k*}$ – which is a shorthand for $((\,)^k)^*$ – , $(\,)^{\geq k}$, $(\,)^{<k}$, and we identify regular expressions with the defined languages. For example, we may write $abbc \in a \cdot b^{\geq 2} \cdot (c \cup d)^*$, $b(ab)^* = (ba)^*b$ and $\{a, b\}^* \cdot c = (a \cup b)^* \cdot c$. Given $u = a_1 \cdots a_n \in \mathbb{A}^*$ and $v = b_1 \cdots b_n \in \mathbb{B}^*$, we write $u \otimes v$ for the word $(a_1, b_1) \cdots (a_n, b_n) \in (\mathbb{A} \times \mathbb{B})^*$. Similarly, given $U \subseteq \mathbb{A}^*$, $V \subseteq \mathbb{B}^*$, we write $U \otimes V \subseteq (\mathbb{A} \times \mathbb{B})^*$ for the set $\{u \otimes v \mid u \in U, v \in V, |u| = |v|\}$.

The **star-height** of a regular expression is the maximum number of nestings of Kleene stars $(\,)^*$. By abuse of terminology, when referring to the star-height of a language, we mean the star-height of some regular expression that represents it (in particular, we do not need to work with the minimum star-height over all expressions). Besides regular expressions, we also work with automata, and use classical techniques on them (notably, pumping arguments). Given an accepting run γ of an automaton $\mathcal{A}$, we often identify **cycles** in it, that is, factors that start and end in the same state, and that can thus be pumped. Such cycles are called **simple** if they do not contain proper factors that are also cycles. Moreover, to avoid mentioning explicitly an automaton for a language L and a run of it, we call **cycle of** L (resp. **simple cycle of** L) the word spelled out by any cycle (resp. simple cycle) of any accepting run of the minimal deterministic automaton recognizing L, and denote the set of all cycles (resp. simple cycles) of L by *cycles*(L) (resp. *simple-cycles*(L)). We remark that, however, that the use of the minimal automaton as a presentation of a regular language L is only to avoid ambiguity when referring to the cycles of L – in fact, our results do not depend on determinism or minimality, and can thus be applied to arbitrary non-deterministic automata, without any difference in the characterizations we present.

A regular language C is **concat-star** (*a.k.a. unit-form* [1]), if it is of the form

$$C = C_1^* u_1 C_2^* u_2 \cdots C_n^* u_n, \tag{$\star$}$$

for $n \in \mathbb{N}$, words $u_1, \ldots, u_n$, and regular languages $C_1, \ldots, C_n$. Without loss of generality, we can always assume that the empty word does not belong to any of the languages C_i. The following trivial decomposition lemma will be used throughout.

▶ **Lemma 1.** *Every regular language is a finite union of concat-star languages.*

The C_i^*'s from $(\star)$ are called **components** of the concat-star language C. Note that (an expression of) a concat-star language as in $(\star)$ has star-height 1 if and only if every C_i is finite. A component C_i^* is **homogeneous** if $C_i^* \subseteq 1^*$ or $C_i^* \subseteq 2^*$. A component which is not homogeneous is called **heterogeneous** (e.g. $C_i^* = \{1, 2\}^*$). It will also be convenient to distinguish a few types of concat-star languages. We say that C is

- **heterogeneous** if it contains at least one heterogeneous component, otherwise it is **homogeneous**;
- **smooth** if every homogeneous component is a language of the form 1^{k*} or 2^{k*}, for some $k > 0$, and there are no consecutive homogeneous components;
- **simple** if it has star-height 1 and it is either homogeneous or smooth heterogeneous.

Hereafter, by "simple language" we mean simple concat-star language. The picture below summarizes the different types of control languages, together with some separating examples.

	homogeneous	smooth heterogeneous	non-smooth heterogeneous	non concat-star
s.-h. > 1	$(1^*1)^*2^*$	$1^*(1^*2)^*2^*$	$1^*2^*(1^*2)^*$	$(1^*2)^* \cup (12)^*$
s.-h. $= 1$	$1^*(11)^*2^*$	$1^*(12)^*2^*$	$1^*2^*(12)^*$	$(12)^*1^* \cup (12)^*2^*$
	simple			

In Section 5 we will see that the Class Containment Problem is reduced to the case of finite unions of simple languages. The latter languages thus form the basis of our characterization.

3 Synchronized relations

A **synchronization** of a pair (w_1, w_2) of words over $\mathbb{A}$ is a word over $\mathbb{2} \times \mathbb{A}$ so that the projection on $\mathbb{A}$ of positions labeled i is exactly w_i, for $i = 1, 2$ – in other words, $\text{shuffle}\{1^{|w_1|} \otimes w_1, 2^{|w_2|} \otimes w_2\}$ is the set of all synchronizations of (w_1, w_2). For example, the words $(1, a)(1, b)(2, a)$ and $(1, a)(2, a)(1, b)$ are two possible synchronizations of the same pair (ab, a). Every word $w \in (\mathbb{2} \times \mathbb{A})^*$ is a synchronization of a unique pair (w_1, w_2), where w_i is the sequence of $\mathbb{A}$-letters corresponding to the symbol i in the first position of $\mathbb{2} \times \mathbb{A}$. We denote such pair (w_1, w_2) by $[\![w]\!]$ and extend the notation to languages $L \subseteq (\mathbb{2} \times \mathbb{A})^*$ by $[\![L]\!] \stackrel{\text{def}}{=} \{[\![w]\!] \mid w \in L\}$.

Given a regular language $C \subseteq \mathbb{2}^*$, we define the class of ***C*-controlled relations** as

$$\textsc{Rel}(C) \stackrel{\text{def}}{=} \big\{[\![L]\!] \;\mid\; L \subseteq C \otimes \mathbb{A}^* \text{ is regular}, \mathbb{A} \text{ is some finite alphabet}\big\}.$$

A slightly different definition is possible, which restricts the class of C-controlled relations to be over a fixed alphabet $\mathbb{A}$, that is, one can define $\textsc{Rel}_{\mathbb{A}}(C) = \{[\![L]\!] \mid L \subseteq C \otimes \mathbb{A}^* \text{ regular}\}$. As far as we are concerned with comparing classes of relations controlled by different languages, the two definitions are somehow interchangeable, in the sense that containment between classes is not sensible to whether we fix or not the alphabet. For example, we will see that, for any alphabet $\mathbb{A}$ with at least two symbols, $\textsc{Rel}_{\mathbb{A}}(C) \subseteq \textsc{Rel}_{\mathbb{A}}(D)$ iff $\textsc{Rel}(C) \subseteq \textsc{Rel}(D)$.

For economy of space, we use $C \subseteq_{\textsc{Rel}} D$ and $C =_{\textsc{Rel}} D$ as shorthands for $\textsc{Rel}(C) \subseteq \textsc{Rel}(D)$ and $\textsc{Rel}(C) = \textsc{Rel}(D)$, respectively. The following properties are easy to verify.

▶ **Lemma 2.** *For every regular $C, D, C', D' \subseteq \mathbb{2}^*$,*

P1. *if $C \subseteq D$, then $C \subseteq_{\textsc{Rel}} D$;*
P2. *if $C \subseteq_{\textsc{Rel}} D$ and $C' \subseteq_{\textsc{Rel}} D'$, then $C \cdot C' \subseteq_{\textsc{Rel}} D \cdot D'$ and $C \cup C' \subseteq_{\textsc{Rel}} D \cup D'$;*
P3. *if $C \subseteq_{\textsc{Rel}} D$, then $C^* \subseteq_{\textsc{Rel}} D^*$;*
P4. *if $C \subseteq 1^*$ and $D \subseteq 2^*$, then $C \cdot D =_{\textsc{Rel}} D \cdot C$;*
P5. *if C is finite, then $C \cdot D =_{\textsc{Rel}} D \cdot C$;*
P6. *if $C \subseteq_{\textsc{Rel}} D$ then $\pi(C) \subseteq \pi(D)$; moreover, if C is finite, the converse also holds;*
P7. *if C is homogeneous concat-star, then $C =_{\textsc{Rel}} \bigcup_{i \in I} 1^{\ell_i *} 1^{k_i} 2^{\hat{\ell}_i *} 2^{\hat{k}_i}$ for a finite I;*
P8. *if C is homogeneous concat-star, $C \subseteq_{\textsc{Rel}} D$ if and only if $\pi(C) \subseteq \pi(D)$.*

Proof idea. P1 is immediate from definitions; henceforth we use it without referencing it. P2 and P3 follow readily from the following decomposition properties.

(a) For every $R \in \textsc{Rel}(C \cdot C')$, there are $R_1, \ldots, R_n \in \textsc{Rel}(C)$, $R'_1, \ldots, R'_n \in \textsc{Rel}(C')$ so that $R = \bigcup_i R_i \cdot R'_i$.
(b) For every $R \in \textsc{Rel}(C \cup C')$, there are $R_1 \in \textsc{Rel}(C)$, $R_2 \in \textsc{Rel}(C')$ so that $R = R_1 \cup R_2$.

(c) For every $R \in \text{REL}(C^*)$, there are $R_1, \ldots, R_n \in \text{REL}(C)$ and $I \subseteq \{1, \ldots, n\}^*$ regular so that $R = \bigcup_{w \in I} R_{w[1]} \cdots R_{w[|w|]}$.

P4 can be verified by first decomposing any relation $R \in \text{REL}(C \cdot D)$ into $\bigcup_i R_i \cdot R'_i$ as in (a), and then observing that in this case $[\![\bigcup_i R_i \cdot R'_i]\!] = [\![\bigcup_i R'_i \cdot R_i]\!]$. For P5, it is easy to see that $1 \cdot D =_{\text{REL}} D \cdot 1$ and $2 \cdot D =_{\text{REL}} D \cdot 2$ for any D, and thus by P2 this extends to commuting with arbitrary finite languages. For P6, observe that if $C \subseteq_{\text{REL}} D$ then $[\![C \otimes a^*]\!] \in \text{REL}(D)$ for $a \in \mathbb{A}$, which means that $\pi(C) \subseteq \pi(D)$. P7 is a consequence of P4 and the so-called Chrobak normal form for regular languages over unary alphabets [7]. Finally, the proof of P8 is a variant of the proof that the operation of shuffle preserves regularity of languages. ◀

4 Characterization of the Class Containment Problem

We give an overview of the main ingredients of our decision procedure for class containment.

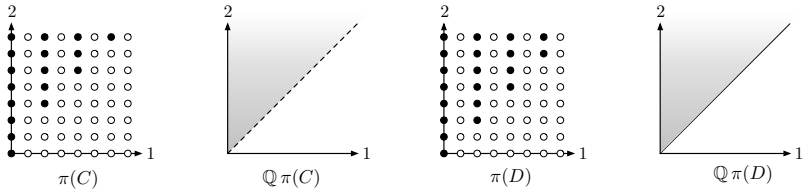
Decomposition. A first ingredient is a decomposition result for regular control languages into $=_{\text{REL}}$-equivalent finite unions of simple languages. Here we only state the result with a short proof sketch; the complete proof will be given in Section 5.

▶ **Proposition 3.** *Every regular language $C \subseteq 2^*$ is effectively $=_{\text{REL}}$-equivalent to a finite union of simple languages.*

Proof idea. One first applies Lemma 1, so as to decompose the regular language C into a finite union of concat-star languages. Then, the concat-star languages are further decomposed into unions of concat-star languages of star-height 1. For example, $(112(12)^* \cup 122)^* =_{\text{REL}} (122)^* \cup (112 \cup 122)^*112 \cup (112 \cup 122)^*11122 \cup (112 \cup 122)^*1111222$. This latter step is more difficult and exploits the increased flexibility of the relation $=_{\text{REL}}$ compared to equality. It also exploits in a crucial way properties of linear sets, and more specifically those that result from taking the Parikh images of concat-star languages. Finally, to get the desired decomposition, one needs to decompose further the concat-star languages of star-height 1 into finite unions of simple languages as in, for example, $(12)^*1^*2^* =_{\text{REL}} (12)^*1^* \cup (12)^*2^*$. This last decomposition makes use of some basic properties from Lemma 2. ◀

Parikh ratios. The **Parikh ratio** of a pair $\bar{x} = (n_1, n_2) \in \mathbb{N}^2 \setminus \{(0,0)\}$ is $\rho(\bar{x}) = \frac{n_1}{n_1+n_2}$. We naturally extend this to non-empty words $w \in 2^*$ by letting $\rho(w) = \rho(\pi(w))$ (this describes the proportion of 1's in w). We further extend the notation to languages: $\rho(C) = \{\rho(w) \mid w \in C \setminus \{\varepsilon\}\}$. Note that $\rho(C) \subseteq [0,1]_{\mathbb{Q}}$. It is sometimes useful to think of $\rho(C)$ as the cone $\mathbb{Q}\,\pi(C) = \{q \cdot \pi(w) \mid q \in \mathbb{Q}, w \in C\}$ inside the rational plane $\mathbb{Q} \times \mathbb{Q}$.

▶ **Example 4.** The Parikh images of the languages $C = (2(2112)^*)^*$ and $D = (2 \cup 2112)^*$ are depicted below. Note that $\rho(C) = \left[0, \frac{1}{2}\right)_{\mathbb{Q}}$, while $\rho(D) = \left[0, \frac{1}{2}\right]_{\mathbb{Q}}$.



The following lemma summarizes the main properties of Parikh ratios that we will need.

▶ **Lemma 5.** *The Parikh ratio of a concat-star language C verifies the following properties:*

1. *If $C = C_1^* u_1 \cdots C_n^* u_n$, then $\rho(\text{cycles}(C)) \subseteq [\min_i \inf \rho(C_i^*), \max_i \sup \rho(C_i^*)]_{\mathbb{Q}}$;*
2. *Moreover, if $C = D^*$ for a finite D, then $\rho(C) = \rho(\text{cycles}(C)) = [\min \rho(D), \max \rho(D)]_{\mathbb{Q}}$.*

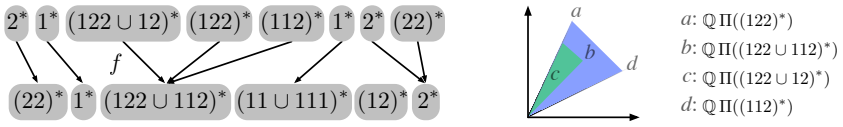
Synchronizing morphisms. Another fundamental ingredient is the notion of *synchronizing morphism*, which intuitively relates the components of a concat-star language C to the components of a concat-star language D by comparing the Parikh ratios.

Let $C = C_1^* u_1 \cdots C_n^* u_n$ be a heterogeneous concat-star language and $D = D_1^* v_1 \cdots D_m^* v_m$ any concat-star language. We say that a function $f : [1, n] \to [1, m]$ is a **synchronizing morphism** (abbreviated *s.m.*) from C to D if

- it is monotonic: $f(i) \leq f(j)$ whenever $i \leq j$; and
- it preserves Parikh-ratio: for every $i \in [1, n]$, $\rho(C_i^*) \subseteq \rho(D_{f(i)}^*)$.

We write $C \xrightarrow{s.m.} D$ to denote the existence of such synchronizing morphism. By convention, if C is homogeneous, then we say that there is always a synchronizing morphism from C to D. In particular, $u \xrightarrow{s.m.} v$ for every $u, v \in \mathbb{2}^*$. The sole purpose of this trivial definition on homogeneous concat-star languages is to make the characterization statements simpler.

▶ **Example 6.** The following function f is a synchronizing morphism:



Observe that synchronizing morphisms are closed under composition and hence $\xrightarrow{s.m.}$ defines a pre-order on concat-star languages.

Class Containment Problem for simple languages. The existence of synchronizing morphism is the key property that characterizes $\subseteq_{\text{Rel}}$ on simple languages. A complete proof of the following proposition will be the theme of Section 6.

▶ **Proposition 7.** *For all simple $C, D \subseteq \mathbb{2}^*$, $C \subseteq_{\text{Rel}} D$ iff $\pi(C) \subseteq \pi(D)$ and $C \xrightarrow{s.m.} D$.*

Note that the case of C homogeneous follows from P8. Intuitively, for any C smooth heterogeneous concat-star language of star-height 1, the characterization says that, $C \subseteq_{\text{Rel}} D$ iff $\pi(C) \subseteq \pi(D)$ and for every component of C, there is a component of D that contains its Parikh ratio. Further, the matching between components is monotonic. For example, we have $(12)^*(112)^* \subseteq_{\text{Rel}} (12 \cup 11122)^*(121)^*1^*2^*$, because the Parikh ratios of $(12)^*$ and $(112)^*$ are included in those of $(12 \cup 11122)^*$ and $(121)^*$, respectively. On the other hand, we have $(112)^*(12)^* \not\subseteq_{\text{Rel}} (12 \cup 11122)^*(121)^*1^*2^*$ because in this case there is no motononic matching between components.

Generalization to unions of simple languages. Section 7 concerns the generalization of the characterization to finite unions of simple languages, which cover arbitrary regular languages up to $=_{\text{Rel}}$-equivalence. The previous characterization for simple languages thus constitutes the base case of our characterization. The lemma below allows a first generalization when C is a union of simple languages and D is a simple language.

▶ **Lemma 8.** *$C_1 \cup C_2 \subseteq_{\text{Rel}} D$ iff $C_1 \subseteq_{\text{Rel}} D$ and $C_2 \subseteq_{\text{Rel}} D$.*

The analogous of Lemma 8 for unions on the right hand-side does not hold in general, as shown by the following example.

▶ **Example 9.** Let $C = (12)^*$, $D_1 = (112 \cup 1122)^*$, and $D_2 = (122 \cup 1122)^*12$. We have $C \subseteq_{\text{Rel}} D_1 \cup D_2$, although $C \not\subseteq_{\text{Rel}} D_1$ and $C \not\subseteq_{\text{Rel}} D_2$.

Neither it holds that Parikh image containment together with the existence of s.m. to one of the disjuncts suffices. For instance, for $C' = (12)^*$, $D'_1 = (1212)^*$, $D'_2 = 1^*2^*$, we have $C' \not\subseteq_{\text{Rel}} D'_1 \cup D'_2$ although $\pi(C') \subseteq \pi(D'_1 \cup D'_2)$ and $C' \xrightarrow{s.m.} D'_1$.

The characterization we provide is inductive on the number of languages that are unioned on the right hand-side. Concretely, for a union of two languages, we will show that $C \subseteq_{\text{Rel}} D_1 \cup D_2$ iff $C \xrightarrow{s.m.} D_i$ for some i and $C \setminus [D_i]_\pi \subseteq_{\text{Rel}} D_{3-i}$, where $[D_i]_\pi$ is the closure of D_i under permutations, that is, $[D_i]_\pi \stackrel{\text{def}}{=} \{w \in \mathbb{2}^* \mid \pi(w) \in \pi(D_i)\}$. The idea that underlies the proof of the necessity of our characterization is that C can be split into a disjoint union of $C \cap [D_i]_\pi$ and $C \setminus [D_i]_\pi$, in such a way that $C \cap [D_i]_\pi \subseteq_{\text{Rel}} D_i$ and $C \setminus [D_i]_\pi \subseteq_{\text{Rel}} D_{3-i}$.

For finite unions of simple languages, we have the following characterization. A complete proof of this theorem will be the theme of Section 7.

▶ **Theorem 10.** *For finite unions $C = \bigcup_i C_i$ and $D = \bigcup_j D_j$ of simple languages, the following are equivalent:*

- $C \subseteq_{\text{Rel}} D$,
- *For all i $\pi(C_i) \subseteq \pi(D)$ and there is j with $C_i \xrightarrow{s.m.} D_j$. In addition, if C_i is heterogeneous, then $C_i \setminus [D_j]_\pi$ is regular and $C_i \setminus [D_j]_\pi \subseteq_{\text{Rel}} \bigcup_{j' \neq j} D_{j'}$.*

Coming back to Example 9, note that $\rho(C) = \{\frac{1}{2}\}$, $\rho(D_1) = [\frac{1}{2}, \frac{2}{3}]_{\mathbb{Q}}$ and $\rho((122 \cup 1122)^*) = [\frac{1}{3}, \frac{1}{2}]_{\mathbb{Q}}$. Therefore, one can explain $C \subseteq_{\text{Rel}} D_1 \cup D_2$ by the fact of having $C \xrightarrow{s.m.} D_1$ and $C \setminus [D_1]_\pi = (1212)^*12 \subseteq_{\text{Rel}} D_2$, where the latter containment holds by the fact that $(1212)^*12 \xrightarrow{s.m.} D_2$ and $\pi((1212)^*12) \subseteq \pi(D_2)$.

Note that there's a caveat in the statement of Theorem 10: $C_i \setminus [D_j]_\pi$ needs to be *regular*. And in fact this is not the case in general: if $C_i = 1^*2^*$ and $D_j = (12)^*$, we get a non-regular language $C_i \setminus [D_j]_\pi = \{1^n 2^m \mid n \neq m\}$. However, provided $C_i \xrightarrow{s.m.} D_j$ for C_i heterogeneous, we show that $C_i \setminus [D_j]_\pi$ is effectively regular (in the sense that an automaton recognizing it can be computed from automata recognizing C_i and D_j). This is a non-trivial fact, and will be proved in Section 5 (Proposition 12).

The second key ingredient is that if $C_i \subseteq_{\text{Rel}} D_1 \cup \cdots \cup D_n$, then there must be some j so that $C_i \xrightarrow{s.m.} D_j$. This will be proved in Section 6 (Lemma 15).

5 Decomposition into simple languages

As already mentioned, we start by reducing the Class Containment Problem for arbitrary regular languages to the case of finite unions of simple languages (Proposition 3 below). We do this in two steps. First, we decompose regular languages into finite unions of concat-star languages of star-height 1 (Lemma 13 below). Then, we further decompose the latter languages into finite unions of simple languages (Lemma 14 below).

Unions of star-height 1 languages. Lemma 13 relies on two key results, which are also of independent interest. The first result is a normal form representation of the Parikh image $\pi(C)$ of a concat-star language C. Formally, we say that a linear set $\langle \bar{x}, P \rangle$ is in **normal form** if the elements of P are linearly independent. We extend this notion to semi-linear sets by saying that $\langle \bar{x}_1, P_1 \rangle \cup \cdots \cup \langle \bar{x}_n, P_n \rangle$ is in normal form if the vectors in $\bigcup_i P_i$ are linearly independent. In particular, in dimension 2, this means that there are at most two vectors in P. Note that if the representation of a semi-linear set is in normal form then all its linear sets are in normal form, but the converse does not hold – for example, consider $\langle \bar{0}, \{(2,0)\} \rangle \cup \langle \bar{0}, \{(3,0)\} \rangle$. The following lemma shows that Parikh images of concat-star languages enjoy normal forms.

▶ **Lemma 11.** *For every concat-star language $C = C_1^* u_1 \cdots C_n^* u_n$, there exists a normal form representation of its Parikh image $\pi(C)$. Moreover, if C is infinite, the union of the period sets is $\{\bar{x}_-, \bar{x}_+\}$, where $\rho(\bar{x}_-) = \min_j(\inf \rho(C_j^*))$ and $\rho(\bar{x}_+) = \max_j(\sup \rho(C_j^*))$.*

Proof idea. Using some basic properties of Parikh images, we reduce to the case where C is a concatenation of expressions of the form u^* (for u a non-empty word) or $(u_1^* \cdots u_n^* u)^*$ (for $u_1, \ldots, u_n, u$ non-empty words). For any C in this form, the Parikh image of words in C can be expressed in terms of some words w_-, w_+ such that $\pi(w_-) = \bar{x}_-$ and $\pi(w_+) = \bar{x}_+$. Then, any word of C can be represented as a constrained iteration of these two words. ◀

It is worth pointing out the difference with the normal form from [12]. While the normal form of [12] holds for arbitrary regular languages, our normal form holds only for *concat-star* languages over *binary* alphabets (*e.g.*, it fails for $(12)^* \cup 1^* \cup 2^*$). Conversely, the normal form from [12] does not guarantee the linear independence of the vectors in the union of the periods, as we do here instead. Proposition 12 below relies on such an additional property. (Also, [1] gives a procedure to compute Parikh images, though no normal form is implied.)

The second result shows that, under certain conditions, one can intersect a regular language C by a language of the form $[D]_\pi = \pi^{-1}(\pi(D))$, with D concat-star, and obtain a language that is again regular. This result not only enables the decomposition into star-height 1 languages, but will be used also later to formalize a recursive characterization of $\subseteq_{\text{REL}}$ for unions of simple languages (cf. Section 7).

▶ **Proposition 12.** *Given C regular and D concat-star so that $\rho(\mathit{cycles}(C)) \subseteq \rho(\mathit{cycles}(D))$, the languages $C \cap [D]_\pi$ and $C \setminus [D]_\pi$ are effectively regular. If in addition D is of the form $D_1^* u$, then $C \cap [D]_\pi \subseteq_{\text{REL}} D$.*

Proof idea. We exploit the fact that words in $\mathbb{2}^*$ are in bijection with paths inside $\mathbb{N}^2$ that originate in $\bar{0} = (0, 0)$ and, furthermore, that words with the same Parikh image correspond to paths with the same endpoints. The claim boils down to considering some word $w \in \mathbb{2}^*$ and proving that, under suitable hypotheses, the path induced by w can be approximated by a path inside $\pi(D)$ that stays sufficiently close to the former path. The use of Lemma 11 will be crucial here, since it gives a normal form $\bigcup_i \langle \bar{x}_i, P_i \rangle$ for the latter set $\pi(D)$. Intuitively, it implies that the words from $[D]_\pi$ are represented by paths that never get too far from the linear set $\langle \bar{0}, \bigcup_i P_i \rangle$. For example, by pairing this property with the assumption that $\rho(\mathit{cycles}(C)) \subseteq \rho(\mathit{cycles}(D))$, one can show that the path induced by a word $w \in C$ stays close to $\langle \bar{0}, \bigcup_i P_i \rangle$, and hence also to $\pi(D)$. Stronger variants of this property are shown, that take into account the exact displacement of points along the path induced by w from the points in $\pi(D)$. These latter properties are used by suitable automata that recognize the languages $C \cap [D]_\pi$ and $C \setminus [D]_\pi$. ◀

As we explained in the proof sketch, the above proposition relies on the normal form for the semi-linear set $\pi(D)$, which in turns relies on the fact that D is concat-star. The proposition does not hold if we replace D with an arbitrary regular language. For instance, consider $C = 1(11)^*2(22)^*$ and $D = (12)^* \cup (11)^*(22)^*$, and observe that $\rho(\mathit{cycles}(C)) = [0, 1]_{\mathbb{Q}} = \rho(\mathit{cycles}(D))$, but $C \cap [D]_\pi = \{1(11)^n 2(22)^n \mid n \in \mathbb{N}\}$ is clearly not regular.

Although Proposition 12 is stated in full generality, that is, for every regular language C so that $\rho(\mathit{cycles}(C)) \subseteq \rho(\mathit{cycles}(D))$, in the proof of the decomposition result below we will use it only for a smooth heterogeneous concat-star language C so that $C \xrightarrow{s.m.} D$ (this is sufficient but not necessary for verifying the hypothesis $\rho(\mathit{cycles}(C)) \subseteq \rho(\mathit{cycles}(D))$).

▶ **Lemma 13.** *Every regular $C \subseteq \mathbb{2}^*$ is $=_{\text{REL}}$-equivalent to a finite union $\bigcup_i D_i$ of concat-star languages of star-height* 1.

Towards the proof of this lemma, note that, by Lemma 1, C is a finite union of concat-star languages $C_1^* u_1 \cdots C_n^* u_n$. The lemma then follows from applying Claim 1 below to each component of the concat-star languages, and then using P2.

▶ **Claim 1.** *Every regular D^* is $=_{\text{REL}}$-equivalent to a finite union $\bigcup_i D_i^* u_i$, with finite D_i's.*

Proof idea of Claim 1. Since $\pi(D^*)$ is a finite union of linear sets, from the latter we can extract languages of the form $D_i^* u_i$. Then we can decompose D^* as the union of $D^* \cap [D_i^* u_i]_\pi$. From there, the result follows easily from Proposition 12 and P2. ◀

Unions of simple languages. We finally show how to decompose into simple languages.

▶ **Lemma 14.** *Every concat-star $C \subseteq \mathbb{2}^*$ of star-height 1 is $=_{\text{REL}}$-equivalent to a finite union $\bigcup_i C_i$ of simple languages.*

Proof idea. By using the basic properties given in Lemma 2, we can reduce the problem to the case where C is of the form $1^{k*}\hat{2}^{\hat{k}*}w^*$ for some heterogeneous word w and some natural numbers $k, \hat{k}$. This case is easy to prove by using again those basic properties. ◀

As a corollary of Lemmas 13 and 14, we have our desired result.

▶ **Proposition 3.** Every regular language $C \subseteq \mathbb{2}^*$ is effectively $=_{\text{REL}}$-equivalent to a finite union of simple languages.

6 Simple languages

We prove the characterization result for simple languages, which we recall here.

▶ **Proposition 7.** For all simple $C, D \subseteq \mathbb{2}^*$, $C \subseteq_{\text{REL}} D$ iff $\pi(C) \subseteq \pi(D)$ and $C \xrightarrow{s.m.} D$.

For the left-to-right direction, by P6, $C \subseteq_{\text{REL}} D$ implies $\pi(C) \subseteq \pi(D)$. The proof that $C \subseteq_{\text{REL}} D$ implies $C \xrightarrow{s.m.} D$ is given in a more general setup where D is a finite union of simple languages. This statement will be used in the characterization of the next section.

▶ **Lemma 15.** *For C a simple language and $D = \bigcup_i D_i$ finite union of simple languages, if $C \subseteq_{\text{REL}} D$, then $C \xrightarrow{s.m.} D_i$ for some i. In particular, for C, D simple languages, if $C \subseteq_{\text{REL}} D$, then $C \xrightarrow{s.m.} D$. Further, the statement holds even if we consider $\text{REL}_{\mathbb{A}}$-containment for any $\mathbb{A}$ with at least two letters.*

Proof idea. The idea is to construct a relation $R \in \text{REL}(C)$ so that from $R \in \text{REL}(D)$, using suitable pumping arguments, one can extract a synchronizing morphism from C to some D_i. The relation R must depend on both languages C, D, but the underlying alphabet can be fixed and taken binary, say $\mathbb{A} = \{a, b\}$. For example, if C is of the form C_1^* and contains two words u^- and u^+ with minimum and maximum Parikh ratios, and if the automaton for D has a single strongly connected component, then one can define the relation $R = [\![(u^- \otimes a^{|u^-|})^* \cdot (u^+ \otimes b^{|u^+|})^*]\!]$. In this case, $R \in \text{REL}(D)$ would imply $\rho(u^-), \rho(u^+) \in \rho(D)$, and hence $C \xrightarrow{s.m.} D$. This construction can be modified for more general languages C, D, by using words with different Parikh ratios from each component of C and by increasing the number of alternations between these ratios on the basis of the number of components of D. While the construction is more involved in the general case, and in particular needs to include iterations of words which are not necessarily of minimum or maximum Parikh ratios for a component, the intuition remains the same. ◀

▶ **Observation 16.** *The previous Lemma 15 does not hold for arbitrary concat-star languages C. For example, consider $(12)^*1^*2^* =_{\text{Rel}} (12)^*1^* \cup (12)^*2^*$, where there is no s.m. from $(12)^*1^*2^*$ to $(12)^*1^*$, nor from $(12)^*1^*2^*$ to $(12)^*2^*$.*

Conversely, to show that the conditions $\pi(C) \subseteq \pi(D)$ and $C \xrightarrow{s.m.} D$ are sufficient to have $C \subseteq_{\text{Rel}} D$, where C, D are simple, it is useful to introduce a normal form for languages of the form C^*, with C finite.

▶ **Lemma 17.** *For every $p, q > 0$, finite $C \subseteq \mathbb{2}^*$, and $u_-, u_+ \in C$ so that $\rho(u_-) = \min \rho(C)$ and $\rho(u_+) = \max \rho(C)$, there exists a finite $C' \subseteq C^*$ so that $C^* =_{\text{Rel}} (u_-^p \cup u_+^q)^* \cdot C'$.*

In particular, the lemma implies that $C^* =_{\text{Rel}} (u_- \cup u_+)^* \cdot C'$ for some finite $C' \subseteq C^*$ and u_-, u_+ words of C of minimum and maximum ratio. In other words, it just suffices to iterate two words from C and then append tails of bounded length to obtain the class $\text{Rel}(C^*)$. With this in mind, we can easily prove our characterization for simple languages.

Proof idea of Proposition 7. The left-to-right direction follows from P6 and Lemma 15. For the opposite direction, the case where C is homogeneous is straightforward by P8. For C heterogeneous, we use P5 to we assume wlog that $C = C_1^* \cdots C_n^* u$ and $D = D_1^* \cdots D_m^* v$. Since every C_i is finite (recall that simple languages have star-height 1), we can consider words $w_{i,-}, w_{i,+}$ of minimum and maximum Parikh ratio. Using the normal form of Lemma 17 plus the existence of s.m., we obtain $C_i^* \subseteq_{\text{Rel}} D_{f(i)}^* C_i'$ for a finite $C_i' \subseteq C_i^*$. Thus, $C_1^* \cdots C_n^* u \subseteq_{\text{Rel}} D_{j_1}^* C_1' \cdots D_{j_n}^* C_n' u =_{\text{Rel}} D_{j_1}^* \cdots D_{j_n}^* C_1' \cdots C_n' u \subseteq_{\text{Rel}} D_1^* \cdots D_m^* v$. ◀

7 Regular languages

We now prove the characterization theorem for unions of simple languages. Thanks to this theorem and to Proposition 3, we will obtain an effective characterization for arbitrary regular languages, and thus solve the Class Containment Problem in its full generality.

▶ **Theorem 10.** For finite unions $C = \bigcup_i C_i$ and $D = \bigcup_j D_j$ of simple languages, we have $C \subseteq_{\text{Rel}} D$ if and only if for all i $\pi(C_i) \subseteq \pi(D)$, there is j with $C_i \xrightarrow{s.m.} D_j$ and if C_i is heterogeneous, then $C_i \setminus [D_j]_\pi$ is regular and $C_i \setminus [D_j]_\pi \subseteq_{\text{Rel}} \bigcup_{j' \neq j} D_{j'}$.

Note in particular that the conditions in the characterization of Theorem 10 require that $C_i \setminus [D_j]_\pi$ is regular. Despite that, this property is always verified when $C_i \xrightarrow{s.m.} D_j$ and C_i is heterogeneous by Proposition 12 from Section 5. Indeed, $C_i \xrightarrow{s.m.} D_j$ for C_i heterogeneous implies that all components of C_i are mapped to components of D_j. In view of Lemma 5 and the fact that C_i and D_j have star-height 1, this implies that $\rho(\mathit{cycles}(C_i)) \subseteq \rho(\mathit{cycles}(D_j))$, and hence, by Proposition 12, $C_i \setminus [D_j]_\pi$ is regular. We are now ready to prove the theorem.

Proof of Theorem 10. For the left-to-right implication, by Lemma 8, we have that $C_i \subseteq_{\text{Rel}} D$ for every i. Containment of Parikh images follows then from P6. For any fixed i, if C_i is homogeneous we have $C_i \xrightarrow{s.m.} D_j$ for every j, and if it is smooth heterogeneous, then Lemma 15 yields the existence of some j so that $C_i \xrightarrow{s.m.} D_j$. By Proposition 12, $C_i \setminus [D_j]_\pi$ is regular, and we now prove that $C_i \setminus [D_j]_\pi \subseteq_{\text{Rel}} \bigcup_{j' \neq j} D_{j'}$. Take $R \in \text{Rel}(C_i \setminus [D_j]_\pi)$ and a regular $L \subseteq (C_i \setminus [D_j]_\pi) \otimes \mathbb{A}^*$ so that $[\![L]\!] = R$. Since $C_i \setminus [D_j]_\pi \subseteq C_i$, we have $R \in \text{Rel}(C_i) \subseteq \text{Rel}(D)$, by P1 and hypothesis. Let $L' \subseteq D \otimes \mathbb{A}^*$ be a regular language so that $[\![L']\!] = [\![L]\!] = R$. Since the projection onto $\mathbb{2}$ of L and L' have necessarily the same Parikh image, it follows that $L' \cap (D_j \otimes \mathbb{A}^*) = \emptyset$, and thus that $L' \subseteq (\bigcup_{j' \neq j} D_{j'}) \otimes \mathbb{A}^*$ or, in other words, that $R \in \text{Rel}(\bigcup_{j' \neq j} D_{j'})$.

For the right-to-left implication, for C_i homogeneous, $\pi(C_i) \subseteq \pi(D)$ implies $C_i \subseteq_{\text{REL}} D$ by P8. For C_i heterogeneous, we have $C_i = (C_i \setminus [D_j]_\pi) \cup (C_i \cap [D_j]_\pi)$. By hypothesis plus property P1, $C_i \setminus [D_j]_\pi \subseteq_{\text{REL}} D$. Then, by Lemma 8, it only remains to check that $C_i \cap [D_j]_\pi \subseteq_{\text{REL}} D$. Now, by Proposition 12 and Proposition 3, $C_i \cap [D_j]_\pi$ is $=_{\text{REL}}$-equivalent to a finite union of simple languages $(C'_k)_{k\in K}$. Note that $C'_k \subseteq_{\text{REL}} C_i$ for all $k \in K$. Then, by the left-to-right direction of Proposition 7, we have $C'_k \xrightarrow{s.m.} C_i$ for all k. By composition of synchronizing morphisms, we obtain $C'_k \xrightarrow{s.m.} D_j$ for all $k \in K$. Since we also have that $\pi(C'_k) \subseteq \pi(D_j)$, by the right-to-left direction of Proposition 7, we have that $C'_k \subseteq_{\text{REL}} D_j$ for all $k \in K$. Then, from Lemma 8 it follows that $C_i \subseteq_{\text{REL}} D_j \subseteq D$. Since this happens for every C_i, again by Lemma 8 the statement follows. ◀

8 Decidability and complexity

We have given a characterization of the pairs C, D of regular languages that satisfy $C \subseteq_{\text{REL}} D$. We argue that this characterization is effective.

As explained in Section 7, there are three main steps that one need to take for deciding whether $C \subseteq_{\text{REL}} D$, for two given regular languages C, D: First, one needs to decompose C and D as finite unions $\bigcup_i C_i$ and $\bigcup_j D_j$ of simple languages. This preprocessing relies on two constructions: the computation of the normal form for semi-linear sets and the construction of an automaton for $C \cap [D]_\pi$, proving that is regular. A close inspection of these proofs in Section 5 shows that both procedures are effective, and thus so is the decomposition.

Then, based on the characterization of Theorem 10, one has to identify suitable synchronizing morphisms from each C_i to some D_j. This step boils down to checking whether two components $C^*_{i,i'}$ and $D^*_{j,j'}$ of concat-star languages satisfy $\rho(C^*_{i,i'}) \subseteq \rho(D^*_{j,j'})$. Thanks to the insight of Lemma 5, the containment of Parikh ratios and thus the existence of such synchronizing morphism is decidable.

Finally, the third step uses Theorem 10, reducing the problem $\bigcup_i C_i \subseteq_{\text{REL}} \bigcup_j D_j$ to sub-problems of the form $C_i \setminus [D_{j_i}]_\pi \subseteq_{\text{REL}} \bigcup_{j' \neq j_i} D_{j'}$, which has a smaller union in the right hand-side and thus can be solved recursively (but in principle non-elementary).

The above arguments show that the Class Containment Problem is decidable. Once we know that $C \subseteq_{\text{REL}} D$ for two given regular languages C, D, it is reasonable to ask whether it is possible to resynchronize any relation from C to D, namely, whether there is an algorithm that transforms any automaton $\mathcal{A}$ recognizing $L \subseteq C \otimes \mathbb{A}^*$ into an automaton $\mathcal{A}'$ recognizing $L' \subseteq D \otimes \mathbb{A}^*$ so that $[\![L']\!] = [\![L]\!]$. A close inspection to our decision procedure for $C \subseteq_{\text{REL}} D$ gives a positive answer to the question. Indeed, all our proofs are constructive.

We can summarize the above arguments with the following corollary.

▶ **Corollary 18.** *There is a non-elementary algorithm that, given two regular languages $C, D \subseteq \mathbb{2}^*$, decides whether $C \subseteq_{\text{REL}} D$.*
There is also a non-elementary algorithm that, given an automaton for $L \subseteq C \otimes \mathbb{A}^*$, constructs an automaton for some $L' \subseteq D \otimes \mathbb{A}^*$ so that $[\![L']\!] = [\![L]\!]$, provided $C \subseteq_{\text{REL}} D$.

9 Discussion

The overall picture we obtain from our results is that $\text{REL}(C) \subseteq \text{REL}(D)$ depends on comparing the ratio growth of the two coordinates on the cycles of the transition graph of the automata $\mathcal{A}_C$, $\mathcal{A}_D$ recognizing C, D. Concretely, our reduction into synchronizing morphisms for simple languages can be thought of restricting our attention to cycles $c_1, \ldots, c_n$ of $\mathcal{A}_C$ so that: c_{i+1} is reachable from c_i, and c_i or c_{i+1} is heterogeneous (recall that in a

simple concat-star language, there are no consecutive homogeneous components). Intuitively, $\textsc{Rel}(C) \subseteq \textsc{Rel}(D)$ whenever $\pi(C) \subseteq \pi(D)$ and for every sequence of cycles $c_1, \ldots, c_n$ as before, there exists a corresponding sequence of cycles $c'_1, \ldots, c'_n$ in $\mathcal{A}_D$ with the same properties so that c_i and c'_i have the same Parikh ratio for every i.

We also recall (cf. proof of Lemma 15) that our characterization holds for the containment problem $\textsc{Rel}(C) \subseteq \textsc{Rel}(D)$, but also for any variant with a fixed alphabet of cardinality at least 2. For the variant with a unary alphabet $\mathbb{A}$, it is easy to see that $\textsc{Rel}_{\mathbb{A}}(C) \subseteq \textsc{Rel}_{\mathbb{A}}(D)$ is equivalent to $\pi(C) \subseteq \pi(D)$. As concerns relations of higher arity defined by control languages $C \subseteq \mathbb{k}^* = [1, k]^*$, it is not clear if a similar characterization may hold. For example, the normal form of Lemma 11 does not generalize to control alphabets of more than two letters. Finally, we leave for future work the issue of determining the precise complexity of the Class Containment Problem.

References

1. Bahareh Badban and Mohammad Torabi Dashti. Semi-linear parikh images of regular expressions via reduction. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 6281 of *Lecture Notes in Computer Science*, pages 653–664. Springer, 2010. doi:10.1007/978-3-642-15155-2_57.
2. Jean Berstel. *Transductions and Context-Free Languages*. B. G. Teubner, 1979.
3. Mikołaj Bojańczyk. Transducers with origin information. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 8573 of *Lecture Notes in Computer Science*, pages 26–37. Springer, 2014. doi:10.1007/978-3-662-43951-7.
4. Julius Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.
5. Olivier Carton, Christian Choffrut, and Serge Grigorieff. Decision problems among the main subfamilies of rational relations. *Informatique Théorique et Applications (ITA)*, 40(2):255–275, 2006. doi:10.1051/ita:2006005.
6. Christian Choffrut. Relations over words and logic: A chronology. *Bulletin of the EATCS*, 89:159–163, 2006.
7. Marek Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47(3):149–158, 1986. doi:10.1016/0304-3975(86)90142-8.
8. Calvin C. Elgot and Jorge E. Mezei. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68, 1965. doi:10.1147/rd.91.0047.
9. Diego Figueira and Leonid Libkin. Path logics for querying graphs: Combining expressiveness and efficiency. In *Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 329–340. IEEE Computer Society Press, 2015. doi:10.1109/LICS.2015.39.
10. Diego Figueira and Leonid Libkin. Synchronizing relations on words. *Theory of Computing Systems*, 57(2):287–318, 2015. doi:10.1007/s00224-014-9584-2.
11. Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 125:1–125:14. Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.125.
12. Eryk Kopczyński and Anthony Widjaja To. Parikh images of grammars: Complexity and applications. In *Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 80–89. IEEE Computer Society Press, 2010. doi:10.1109/LICS.2010.21.
13. Maurice Nivat. Transduction des langages de Chomsky. *Annales de l'Institut Fourier*, 18:339–455, 1968.

Reachability Switching Games

John Fearnley
University of Liverpool, UK
john.fearnley@liverpool.ac.uk

Martin Gairing
University of Liverpool, UK
gairing@liverpool.ac.uk

Matthias Mnich[1]
Universität Bonn, Germany
and Maastricht University, The Netherlands
mmnich@uni-bonn.de

Rahul Savani
University of Liverpool, UK
rahul.savani@liverpool.ac.uk

Abstract

In this paper, we study the problem of deciding the winner of reachability switching games. We study zero-, one-, and two-player variants of these games. We show that the zero-player case is NL-hard, the one-player case is NP-complete, and that the two-player case is PSPACE-hard and in EXPTIME. For the zero-player case, we also show P-hardness for a succinctly-represented model that maintains the upper bound of NP ∩ coNP. For the one- and two-player cases, our results hold in both the natural, explicit model and succinctly-represented model. We also study the structure of winning strategies in these games, and in particular we show that exponential memory is required in both the one- and two-player settings.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Deterministic Random Walks, Model Checking, Reachability, Simple Stochastic Game, Switching Systems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.124

Related Version A full version of this paper with all proofs is available at [8], https://arxiv.org/abs/1709.08991.

1 Introduction

A *switching system* (also known as a Propp machine) attempts to replicate the properties of a random system in a deterministic way [14]. It does so by replacing the nodes of a Markov chain with *switching nodes*. Each switching node maintains a queue over its outgoing edges. When the system arrives at the node, it is sent along the first edge in this queue, and that edge is then sent to the back of the queue. In this way, the switching node ensures that, after a large number of visits, each outgoing edge is used a roughly equal number of times.

The Propp machine literature has focussed on *many-token* switching systems and has addressed questions such as how well these systems emulate Markov chains. Recently, Dohrau

[1] Supported by ERC Starting Grant 306465 (BeyondWorstCase) and DFG grant MN 59/4-1.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 124; pp. 124:1–124:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

et. al. [7] initiated the study of *single-token* switching systems and found that the reachability problem raised interesting complexity-theoretic questions. Inspired by that work, we study the question *how hard is it to model check single-token switching systems?* A switching node is a simple example of a fair scheduler, and thus it is natural to consider model checking of switching systems. We already have a good knowledge about the complexity of model checking Markovian systems, but how does this change when we instead use switching nodes?

Our contribution. In this paper, we initiate the study of model checking in switching systems. We focus on *reachability* problems, one of the simplest model checking tasks. This corresponds to determining the winner of a *two-player reachability switching game*. We study zero-, one-, and two-player variants of these games, which correspond to switching versions of Markov chains, Markov decision processes [20], and simple stochastic games [2], respectively.

The main message of the paper is that deciding reachability in one- and two-player switching games is harder than deciding reachability in Markovian systems. Specifically, we show that deciding the winner of a one-player game is NP-complete, and that the problem of deciding the winner of a two-player game is PSPACE-hard and in EXPTIME.

We also study the complexity of zero-player games, where we show hardness results that complement the upper bounds shown in previous work [7]. For the standard model of switching systems, which we call *explicit games*, we are able to show a lower bound of NL-hardness, which is still quite far from the known upper bound of $\mathtt{UP} \cap \mathtt{coUP}$. We also show that if one extends the model by allowing the switching order to be represented in a concise way, then a stronger lower bound of P-hardness can be shown, while still maintaining an $\mathtt{NP} \cap \mathtt{coNP}$ upper bound. We call these concisely represented games *succinct games*, and we also observe that all of our other results, both upper and lower bounds, still apply to succinct games. Our results are summarised in the following table.

	Markovian	Switching (explicit)	Switching (succinct)
0-player	PL-complete[2]	NL-hard; in CLS, in $\mathtt{UP} \cap \mathtt{coUP}$	P-hard; in $\mathtt{NP} \cap \mathtt{coNP}$
1-player	P-complete	NP-complete	NP-complete
2-player	$\mathtt{NP} \cap \mathtt{coNP}$	PSPACE-hard; in EXPTIME	PSPACE-hard; in EXPTIME

For the explicit zero-player case the first bound was an $\mathtt{NP} \cap \mathtt{coNP}$ upper bound given by Dohrau et al. [7], and a PLS upper bound was then given by Karthik [15]. The CLS and $\mathtt{UP} \cap \mathtt{coUP}$ upper bounds, which subsume the two earlier bounds, were given by Gärtner et al. [10], who also produced a $O(1.4143^n)$ algorithm for solving explicit zero-player games. All the other upper and lower bounds in the table are proved in this paper.

Finally, we address the memory requirements of winning strategies in reachability switching games. It is easy to see that winning strategies exist that use exponential memory. These strategies simply remember the current switch configuration of the switching nodes, and their existence can be proved by blowing up a switching game into an exponentially sized reachability game, and then following the positional winning strategies from that reachability game. This raises the question of whether there are winning strategies that use less than

[2] PL, or probabilistic L, is the class of languages recognizable by a polynomial time logarithmic space randomized machine with probability $> 1/2$.

exponential memory. We answer this negatively, by showing that the reachability player may need $\Omega(2^{n/2})$ memory states to win a one-player reachability switching game, and that both players may need to use $\Omega(2^n)$ memory states to win a two-player game.

Related work. Switching games are part of a research thread at the intersection of computer science and physics. This thread has studied zero-player switching systems, also known as *deterministic random walks*, *rotor-router walks*, the *Eulerian walkers model* [19] and *Propp machines* [3–6, 13, 14]. Propp machines have been studied in the context of derandomizing algorithms and pseudorandom simulation, and in particular have received a lot of attention in the context of load balancing [1, 9]. However, most work on Propp machines has focused on how well multi-token switching systems simulate Markov chains. The idea of studying *single-token* reachability should be credited to Dohrau at al. [7].

Katz et al. [16], Groote and Ploeger [12], and others [12, 18, 21], considered *switching graphs*; these are graphs in which certain vertices (switches) have exactly one of their two outgoing edges activated. However, the activation of the alternate edge does not occur when a vertex is traversed by a run; this is the key difference to switching games in this paper.

Markov decision processes [20] and simple stochastic games [2] are important objects of study in *probabilistic model checking*, which is an central topic in the field of formal verification. Probabilistic model checking is now a mature topic, with tools like PRISM [17] providing an accessible interface to the research that has taken place.

2 Preliminaries

A reachability switching game (RSG) is defined by a tuple $(V, E, V_{\mathrm{R}}, V_{\mathrm{S}}, V_{\mathrm{Swi}}, \mathrm{Ord}, s, t)$, where (V, E) is a finite directed graph, and V_{R}, V_{S}, V_{Swi} partition V into *reachability vertices*, *safety vertices*, and *switching vertices*, respectively. The reachability vertices V_{R} are controlled by the *reachability player*, the safety vertices V_{S} are controlled by the *safety player*, and the switching vertices V_{Swi} are not controlled by either player, but instead follow a predefined "switching order". The function Ord defines this *switching order*: for each switching vertex $v \in V_{\mathrm{Swi}}$, we have that $\mathrm{Ord}(v) = \langle u_1, u_2, \ldots, u_k \rangle$ where we have that $(v, u_i) \in E$ for all u_i in the sequence. Note that a particular vertex u may appear more than once in the sequence. The vertices $s, t \in V$ specify *source* and *target* vertices for the game.

A *state* of the game is defined by a tuple (v, C), where v is a vertex in V, and $C : V_{\mathrm{Swi}} \to \mathbb{N}$ is a function that assigns a number to each switching vertex, which represents how far that vertex has progressed through its switching order. Hence, it is required that $C(u) \leq |\mathrm{Ord}(v)| - 1$, since the counts specify an index to the sequence $\mathrm{Ord}(v)$.

When the game is at a state (v, C) with $v \in V_{\mathrm{R}}$ or $v \in V_{\mathrm{S}}$, then the respective player chooses an outgoing edge at v, and the count function does not change. For states (v, C) with $v \in V_{\mathrm{Swi}}$, the successor state is determined by the count function. More specifically, we define $\mathrm{Upd}(v, C) : V_{\mathrm{Swi}} \to \mathbb{N}$ so that for each $u \in V_{\mathrm{Swi}}$ we have $\mathrm{Upd}(v, C)(u) = C(u)$ if $v \neq u$, and $\mathrm{Upd}(v, C)(u) = (C(u) + 1) \bmod |\mathrm{Ord}(u)|$ otherwise. This function increases the count at v by 1, and wraps around to 0 if the number is larger than the length of the switching order at v. Then, the successor state of (v, C), denoted as $\mathrm{Succ}(v, C)$ is $(u, \mathrm{Upd}(v, C))$, where u is the element at position $C(v)$ in $\mathrm{Ord}(v)$.

A *play* of the game is a (potentially infinite) sequence of states $(v_1, C_1), (v_2, C_2), \ldots$ with the following properties:

1. $v_1 = s$ and $C_1(v) = 0$ for all $v \in V_{\mathrm{Swi}}$;
2. If $v_i \in V_{\mathrm{R}}$ or $v_i \in V_{\mathrm{S}}$ then $(v_i, v_{i+1}) \in E$ and $C_i = C_{i+1}$;

3. If $v_i \in V_{\text{Swi}}$ then $(v_{i+1}, C_{i+1}) = \text{Succ}(v_i, C_i)$;
4. If the play is finite, then the final state (v_n, C_n) must either satisfy $v_n = t$, or v_n must have no outgoing edges.

A play is *winning for the reachability player* if it is finite and the final state is the target vertex. A (deterministic, history dependent) *strategy for the reachability player* is a function that maps each play prefix $(v_1, C_1), (v_2, C_2), \ldots, (v_k, C_1)$, with $v_k \in V_{\text{R}}$, to an outgoing edge of v_k. A play $(v_1, C_1), (v_2, C_2), \ldots$ is *consistent* with a strategy if, whenever $v_i \in V_{\text{R}}$, we have that (v_i, v_{i+1}) is the edge chosen by the strategy. Strategies for the safety player are defined analogously. A strategy is *winning* if all plays consistent with it are winning.

The representation of the switching order. Recall that $\text{Ord}(v) = \langle u_1, u_2, \ldots, u_k \rangle$ gives a sequence of outgoing edges for every switching vertex. We consider two possible ways of representing $\text{Ord}(v)$ in this paper. In *explicit* RSGs, $\text{Ord}(v)$ is represented by simply writing down the sequence $\langle u_1, u_2, \ldots, u_k \rangle$.

We also consider games in which $\text{Ord}(v)$ is written down in a more concise way, which we call *succinct* RSGs. In these games, for each switching vertex v, we have a sequence of pairs $\langle (u_1, t_1), (u_2, t_2), \ldots, (u_k, t_k) \rangle$, where each u_i is a vertex with $(v, u_i) \in E$, and each t_i is a natural number. The idea is that $\text{Ord}(v)$ should contain t_1 copies of u_1, followed by t_2 copies of u_2, and so on. So, if $\text{Rep}(u, t)$ gives the sequence containing t copies of u, and if $\cdot$ represents sequence concatenation, then $\text{Ord}(v) = \text{Rep}(u_1, t_1) \cdot \text{Rep}(u_2, t_2) \cdot \ \ldots \ \cdot \text{Rep}(u_k, t_k)$. Any explicit game can be written down in the succinct encoding by setting all $t_i = 1$. Note, however, that in a succinct game $\text{Ord}(v)$ may have exponentially many elements, even if the input size is polynomial, since each t_i is represented in binary.

3 One-player reachability switching games

In this section we consider one-player RSGs, i.e., where $V_{\text{S}} = \emptyset$.

3.1 Containment in NP

We show that deciding whether the reachability player wins a one-player RSG is in NP. Our proof holds for both explicit and succinct games. The proof uses *controlled switching flows*. These extend the idea of switching flows, which were used by Dohrau et al. [7] to show containment of the zero-player reachability problem in $\text{NP} \cap \text{coNP}$.

Controlled switching flow. A *flow* is a function $F : E \to \mathbb{N}$ that assigns a natural number to each edge in the game. For each vertex v, we define $\text{Bal}(F, v) = \sum_{(v,u) \in E} F(v, u) - \sum_{(w,v) \in E} F(w, v)$, which is the difference between the outgoing and incoming flow at v. For each switching node $v \in V_{\text{Swi}}$, let $\text{Succ}(v)$ denote the set of vertices that appear in $\text{Ord}(v)$, and for each index $i \leq |\text{Ord}(v)|$ and each vertex $u \in \text{Succ}(v)$, let $\text{Out}(v, i, u)$ be the number of times that u appears in the first i entries of $\text{Ord}(v)$. In other words, $\text{Out}(v, i, u)$ gives the amount of flow that should be sent to u if we send exactly i units of flow into v.

A flow F is a *controlled switching flow* if it satisfies the following constraints:

- The source vertex s satisfies $\text{Bal}(F, s) = 1$, and the target vertex t satisfies $\text{Bal}(F, t) = -1$.
- Every vertex v other than s or t satisfies $\text{Bal}(F, v) = 0$.
- Let $v \in V_{\text{Swi}}$ be a switching node, $k = |\text{Ord}(v)|$, and let $I = \sum_{(u,v) \in E} F(u, v)$ be the total amount of flow incoming to v. Define p to be the largest integer such that $p \cdot k \leq I$ (which may be 0), and $q = I \bmod k$. For every vertex $w \in \text{Succ}(v)$ we have that $F(v, w) = p \cdot \text{Out}(v, k, w) + \text{Out}(v, q, w)$.

The first two constraints ensure that F is a flow from s to t, while the final constraint ensures that the flow respects the switching order at each switching node. Note that there are no constraints on how the flow is split at the nodes in V_{R}. For each flow F, we define the size of F to be $\sum_{e \in E} F(e)$. A flow of size k can be written down using at most $|E| \cdot \log k$ bits.

Marginal strategies. A *marginal* strategy for the reachability player is defined by a function $M : E \to \mathbb{N}$, which assigns a target number to each outgoing edge of the vertices in V_{R}. The strategy ensures that each edge e is used no more than $M(e)$ times. That is, when the play arrives at a vertex $v \in V_{\mathrm{R}}$, the strategy checks how many times each outgoing edge of v has been used so far, and selects an arbitrary outgoing edge e that has been used strictly less than $M(e)$ times. If there is no such edge, then the strategy is undefined.

Observe that a controlled switching flow defines a marginal strategy for the reachability player. We prove that this strategy always reaches the target.

▶ **Lemma 1.** *If a one-player RSG has a controlled switching flow F, then any corresponding marginal strategy is winning for the reachability player.*

In the other direction, if the reachability player has a winning strategy, then there exists a controlled switching flow, and we can give an upper bound on its size.

▶ **Lemma 2.** *If the reachability player has a winning strategy for a one-player RSG , then that game has a controlled switching flow F, and the size of F is at most $n \cdot l^n$, where n is the number of nodes in the game and $l = \max_{v \in V_{Swi}} |\operatorname{Ord}(v)|$.*

▶ **Corollary 3.** *If the reachability player has a winning strategy for a one-player RSG, then he also has a marginal winning strategy.*

Finally, we can show that solving a one-player RSG is in NP.

▶ **Theorem 4.** *Deciding the winner of an explicit or succinct one-player RSG is in* NP.

3.2 NP-hardness

In this section we show that deciding the winner of a one-player RSG is NP-hard. Our construction will produce an explicit RSG, so we obtain NP-hardness for both explicit and succinct games. We reduce from 3SAT. Throughout this section, we will refer to a 3SAT instance with variables $x_1, x_2, \ldots, x_n$, and clauses $C_1, C_2, \ldots, C_m$. It is well-known [22, Thm. 2.1] that 3SAT remains NP-hard even if all variables appear in at most three clauses. We make this assumption during our reduction.

Overview. The idea behind the construction is that the player will be asked to assign values to each variable. Each variable x_i has a corresponding vertex that will be visited 3 times during the game. Each time this vertex is visited, the player will be asked to assign a value to x_i in a particular clause C_j. If the player chooses an assignment that *does not* satisfy C_j, then the game records this by incrementing a counter. If the counter corresponding to any clause C_j is incremented to three (or two if the clause only has two variables), then the reachability player immediately loses, since the chosen assignment fails to satisfy C_j.

The problem with the idea presented so far is that there is no mechanism to ensure that the reachability player chooses a consistent assignment to the same variable. Since each variable x_i is visited three times, there is nothing to stop the reachability player from choosing contradictory assignments to x_i on each visit. To address this, the game also counts

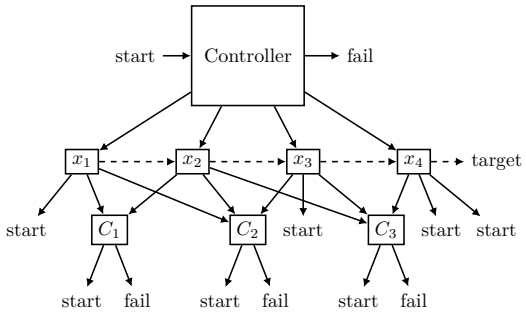


Figure 1 Overview of our construction for one player for the example formula $C_1 \wedge C_2 \wedge C_3 = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4)$. Note that the negations of variables in the formula are not relevant for this high-level view; they will feature in the clause gadgets as explained below. The edges for the variable phase are solid, and the edges for the verification phase are dashed.

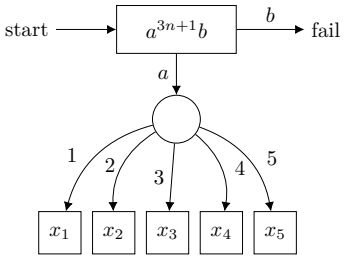


Figure 2 The control gadget.

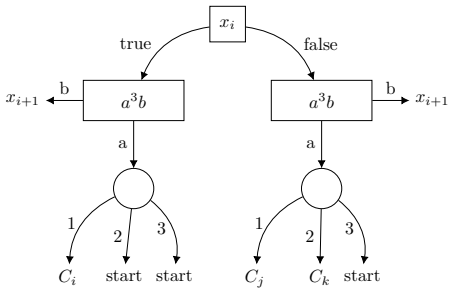


Figure 3 A variable gadget.

how many times each assignment is chosen for x_i. At the end of the game, if the reachability player has not already lost by failing to satisfy the formula, the game is configured so that the target is only reachable if the reachability player chose a consistent assignment. A high-level overview of the construction for an example formula is given in Fig. 1.

The control gadget. The sequencing in the construction is determined by the control gadget, which is shown in Fig. 2. In our diagramming notation, square vertices belong to the reachability player. Circle vertices are switching nodes, and the switching order of each switching vertex is labelled on its outgoing edges. Our diagrams also include *counting gadgets*, which are represented as non-square rectangles that have labelled output edges. The counting gadget is labelled by a sequence over these outputs, with the idea being that if the play repeatedly reaches the gadget, then the corresponding output sequence will be produced. In Fig. 2 the gadget is labelled by $a^{3n+1}b$, which means the first $3n+1$ times the gadget is used the token will be moved along the a edge, and the $3n+2$nd time the gadget is used the token will be moved along the b edge. This gadget can be easily implemented by a switching node that has $3n+2$ outgoing edges, the first $3n+1$ of which go to a, while the $3n+2$nd edge goes to b. We use gadgets in place of this because it simplifies our diagrams.

The control gadget has two phases. In the *variable phase*, each variable gadget, represented by the vertices x_1 through x_n is used exactly 3 times, and thus overall the gadget will be used $3n$ times. This is accomplished by a switching node that ensures that each variable is used 3 times. After each variable gadget has been visited 3 times, the control gadget then sends the token to the x_1 variable gadget for the *verification phase* of the game. In this phase, the reachability player must prove that he gave consistent assignments to all variables.

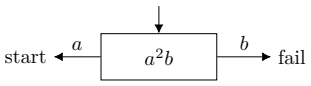


Figure 4 A gadget for a clause with three variables.

If the control gadget is visited $3n + 2$ times, then the token will be moved to the *fail* vertex. This vertex has no outgoing edges, and thus is losing for the reachability player.

The variable gadgets. Each variable x_i is represented by a variable gadget, which is shown in Fig. 3. This gadget will be visited 3 times in total during the variable phase, and each time the reachability player must choose either the true or false edges at the vertex x_i. In either case, the token will then pass through a counting gadget, and then move to a switching vertex which either moves the token to a clause gadget, or back to the start vertex.

It can be seen that the gadget is divided into two almost identical branches. One corresponds to a true assignment to x_i, and the other to a false assignment to x_i. The clause gadgets are divided between the two branches of the gadget. In particular, a clause appears on a branch if and only if the variable appears in that clause and the choice made by the reachability player *fails* to satisfy the clause. So, the clauses in which x_i appears positively appear on the false branch of the gadget, while the clauses in which x_i appears negatively appear on the true branch.

The switching vertices each have exactly three outgoing edges. These edges use an arbitrary order over the clauses assigned to the branch. If there are fewer than 3 clauses on a particular branch, the remaining edges of the switching node go back to the start vertex. Note that this means that a variable can be involved with fewer than three clauses.

The counting gadgets will be used during the verification phase of the game, in which the variable player must prove that he has chosen consistent assignments to each of the variables. Once each variable gadget has been used 3 times, the token will be moved to x_1 by the control gadget. If the reachability player has used the same branch three times, then he can choose that branch, and move to x_2, which again has the same property. So, if the reachability player gives a consistent assignment to all variables, he can eventually move to x_n, and then on to x_{n+1}, which is the target vertex of the game. Since, as we will show, there is no other way of reaching x_{n+1}, this ensures that the reachability player must give consistent assignments to the variables in order to win the game.

The clause gadgets. Each clause C_j is represented by a clause gadget, an example of which is shown in Fig. 4. The gadget counts how many variables have failed to satisfy the corresponding clause. If the number of times the gadget is visited is equal to the number of variables involved with the clause, then the game moves to the fail vertex, and the reachability player immediately loses. In all other cases, the token moves back to the start vertex.

Correctness. The following lemma shows that the reachability player wins the one-player RSG if and only if the 3SAT instance is satisfiable.

▶ **Lemma 5.** *The reachability player wins the one-player RSG if and only if the 3SAT instance is satisfiable.*

Note that our game can be written down as an explicit game, so our lower bound applies to both explicit and succinct games. Hence, we have the following theorem.

▶ **Theorem 6.** *Deciding the winner of an explicit or succinct one-player RSG is* NP*-hard.*

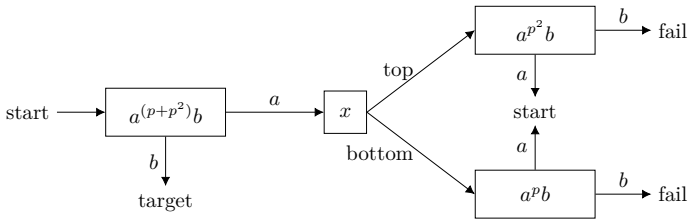


Figure 5 One-player memory lower bound construction.

3.3 Memory requirements of winning strategies in one-player games

Consider the game shown in Fig. 5, which takes as input a parameter p that we will fix later. The only control state for the player is x. By construction, x will be visited $p + p^2$ times. Each time, the player must choose either the top or bottom edge. If the player uses the top edge strictly more than p^2 times, or the bottom edge strictly more than p times, then he will immediately lose the game. If the player does not lose the game in this way, then after $p^2 + p$ rounds the target will be reached, and the player will win the game.

The player has an obvious winning strategy: use the top edge p^2 times and the bottom edge p times. Intuitively, there are two ways that the player could implement the strategy. (1) Use the bottom edge p times, and then use the top edge p^2 times. This approach uses p memory states to count the number of times the bottom edge has been used. (2) Use the bottom edge once, use the top edge p times, and then repeat. This approach uses p memory states to count the number of times the top state has been used after each use of the bottom edge. We can prove that one cannot do significantly better.

▶ **Lemma 7.** *The reachability player must use at least $p - 1$ memory states to win the game shown in Fig. 5.*

Setting $p = 2^{n/2}$ gives us our lower bound. Even though p is exponential, it is possible to create an explicit switching gadget that produces the sequence $a^{2^n}b$ with n switching nodes.

▶ **Lemma 8.** *For all $x \in \mathbb{N}$ there is an explicit switching gadget of size $\log_2(x)$ with output $a^x b$.*

▶ **Theorem 9.** *The number of memory states needed in an explicit one-player RSG is $\Omega(2^{\frac{n}{2}})$.*

4 Two-player reachability switching games

4.1 Containment in EXPTIME

We first observe that solving a two-player RSG lies in `EXPTIME`. This can be proved easily, either by blowing the game up into an exponentially sized reachability game, or equivalently, by simulating the game on an alternating polynomial-space Turing machine.

▶ **Theorem 10.** *Deciding the winner of an RSG is in* `EXPTIME`.

4.2 PSPACE-hardness

We show that deciding the winner of an explicit two-player RSG is `PSPACE`-hard, by reducing *true quantified boolean formula* (TQBF), the canonical `PSPACE`-complete problem, to our problem. Throughout this section we will refer to a TQBF instance $\exists x_1 \forall x_2 \ldots \exists x_{n-1} \forall x_n \cdot \phi(x_1, x_2, \ldots, x_n)$, where ϕ denotes a boolean formula given in negation normal form, which requires that negations are only applied to variables, and not sub-formulas. The problem is to decide whether this formula is true.

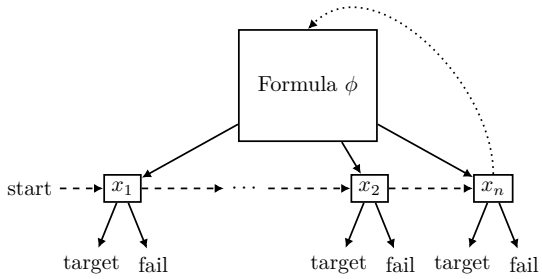


Figure 6 High-level overview of our construction for two players. The dashed lines between variables are part of the first, quantifier phase; the dotted line from variable x_n to the Formula is the transition between phases, and the solid edges are part of the second, formula phase.

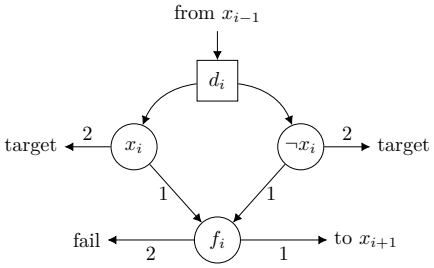


Figure 7 The initialization gadget for an existentially quantified variable x_i.

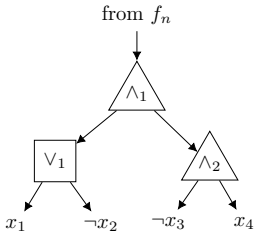


Figure 8 The formula phase game for the formula $(x_1 \vee \neg x_2) \wedge \neg x_3 \wedge x_4$.

Overview. We will implement the TQBF formula as a game between the reachability player and the safety player. This game will have two phases. In the *quantifier phase*, the two players assign values to their variables in the order specified by the quantifiers. In the *formula phase*, the two players determine whether ϕ is satisfied by these assignments by playing the standard model-checking game for propositional logic. The target state of the game is reached if and only if the model checking game determines that the formula is satisfied. This high-level view of our construction is depicted in Fig. 6.

The quantifier phase. Each variable in the TQBF formula will be represented by an *initialization gadget*. The initialization gadget for an existentially quantified variable is shown in Fig. 7. The gadget for a universally quantified variable is almost identical, but the state d_i is instead controlled by the safety player.

During the quantifier phase, the game will start at d_1, and then pass through the gadgets for each of the variables in sequence. In each gadget, the controller of d_i must move to either x_i or $\neg x_i$. In either case, the corresponding switching node moves the token to f_i, which then subsequently moves the token on to the gadget for x_{i+1}.

The important property to note here is that once the player has made a choice, any subsequent visit to x_i or $\neg x_i$ will end the game. Suppose that the controller of d_i chooses to move to x_i. If the token ever arrives at x_i a second time, then the switching node will move to the target vertex and the reachability player will immediately win the game. If the token ever arrives at $\neg x_i$ the token will move to f_i and then on to the fail vertex, and the Safety player will immediately win the game. The same property holds symmetrically if the controller of d_i chooses $\neg x_i$ instead. In this way, the controller of d_i selects an assignment to x_i. Hence, the reachability player assigns values to the existentially quantified variables, and the safety player assigns values to the universally quantified variables.

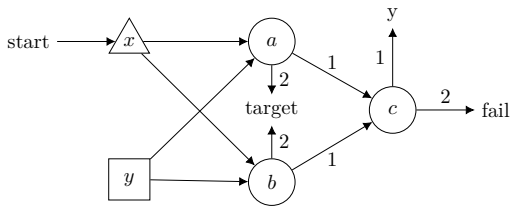


Figure 9 An RSG in which the reachability player needs to use memory.

The formula phase. Once the quantifier phase has ended, the game moves into the formula phase. In this phase the two players play a game to determine whether ϕ is satisfied by the assignments to the variables. This is the standard model checking game for first order logic. The players play a game on the parse tree of the formula, starting from the root. The reachability player controls the $\vee$ nodes, while the safety player controls the $\wedge$ nodes (recall that the game is in negation normal form, so there are no internal $\neg$ nodes.) Each leaf is either a variable or its negation, which in our game are represented by the x_i and $\neg x_i$ nodes in the initialization gadgets. An example of this game is shown in Fig. 8. In our diagramming notation, nodes controlled by the safety player are represented by triangles.

Intuitively, if ϕ is satisfied by the assignment to $x_1, \ldots, x_n$, then no matter what the safety player does, the reachability player is able to reach a leaf node corresponding to a true assignment, and as mentioned earlier, he will then immediately win the game. Conversely, if ϕ is not satisfied, then no matter what the reachability player does, the safety player can reach a leaf corresponding to a false assignment, and then immediately win the game.

▶ **Lemma 11.** *The reachability player wins if and only if the QBF formula is true.*

Since we have shown the lower bound for explicit games, we also get the same lower bound for succinct games as well. We have shown the following theorem.

▶ **Theorem 12.** *Deciding the winner of an explicit or succinct RSG is* PSPACE*-hard.*

Note that all runs of the game have polynomial length, a property that is not shared by all RSGs. This gives us the following corollary.

▶ **Corollary 13.** *Deciding the winner of a polynomial-length RSG is* PSPACE*-complete.*

4.3 Memory requirements for two player games

We can show even stronger memory lower bounds in two-player games compared to one-player games. Fig. 9 shows a simple gadget that forces the reachability player to use memory. The game starts by allowing the safety player to move the token from x to either a or b. Whatever the choice, the token then moves to c and then on to y. At this point, if the reachability player moves the token to the node chosen by the safety player, then the token will arrive at the target node and the reachability player will win. If the reachability player moves to the other node, the token will move to c for a second time, and then on to the fail vertex, which is losing for the reachability player. Thus, every winning strategy of the reachability player must remember the choice made by the safety player.

We can create a similar gadget that forces the safety player to use memory by swapping the players. In the modified gadget, the safety player has to choose the vertex *not chosen* by the reachability player. Thus, in an RSG, winning strategies for both players need to use memory. By using n copies of the memory gadget, we can show the following lower bound.

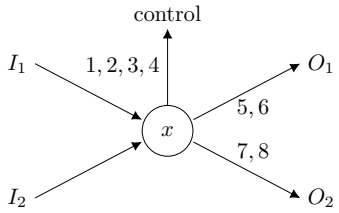


Figure 10 An AND-gate of depth 2.

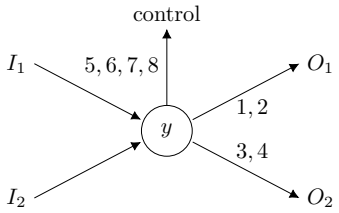


Figure 11 An OR-gate of depth 2.

▶ **Lemma 14.** *In an explicit or succinct RSG, winning strategies for both players may need to use 2^n memory states, where n is the number of switching nodes.*

5 Zero-player reachability switching games

5.1 Explicit zero-player games

We show that deciding the winner of an explicit zero-player game is NL-hard. To do this, we reduce from the problem of deciding s-t connectivity in a directed graph. The idea is to make every node in the graph a switching node. We then begin a walk from s. If, after $|V|$ steps we have not arrived at t, we go back to s and start again. So, if there is a path from s to t, then the switching nodes must eventually send the token along that path.

Formally, given a graph (V, E), we produce a zero-player RSG played on $V \times V \cup \{\mathsf{fin}\}$, where the second component of each state is a counter that counts up to $|V|$. Every vertex is a switching node, the start vertex is $(s, 1)$, and the target vertex is fin. Each vertex (v, k) with $v \neq t$ and $k < |V|$ has outgoing edges to $(u, k + 1)$ for each outgoing edge $(v, u) \in E$. Each vertex $(v, |V|)$ with $v \neq t$ has a single outgoing edge to $(s, 1)$. Every vertex (t, k) with $1 \leq k \leq |V|$ has a single outgoing edge to fin. This game can be constructed in logarithmic space by looping over each element in $V \times V$ and producing the correct outgoing edges.

▶ **Theorem 15.** *Deciding the winner of an explicit zero-player RSG is* NL*-hard under logspace reductions.*

5.2 Succinct games

Deciding reachability for succinct zero-player games still lies in NP ∩ coNP. This can be shown using essentially the same arguments that were used to show NP ∩ coNP containment for explicit games [7]. The fact that the problem lies in NP follows from Theorem 4, since every succinct zero-player game is also a succinct one-player game, and so a switching flow can be used to witness reachability. To put the problem in coNP, one can follow the original proof given by Dohrau et al. [7, Theorem 3] for explicit games. This proof condenses all losing and infinite plays into a single failure state, and then uses a switching flow to witness reachability for that failure state. Their transformation uses only the graph structure of the game, and not the switching order, and so it can equally well be applied to succinct games.

In contrast to explicit games, we can show a stronger lower bound of P-hardness for succinct games. We will reduce from the problem of evaluating a boolean circuit (the *circuit value problem*), which is one of the canonical P-complete problems. We will assume that the circuit has fan-in and fan-out 2, that all gates are either AND-gates or OR-gates, and that the circuit is *synchronous*, meaning that the outputs of the circuit have depth 1, and all gates at depth i get their inputs from gates of depth exactly $i + 1$. This is Problem A.1.6

"Fanin 2, Fanout 2 Synchronous Alternating Monotone CVP" of Greenlaw et al. [11]. We will reduce from the following decision problem: for a given input bit-string $B \in \{0,1\}^n$, and a given output gate g, is g evaluated to true when the circuit is evaluated on B?

Boolean gates. We will simulate the gates of the circuit using switching nodes. A gate at depth $i > 1$ is connected to exactly two gates of depth $i+1$ from which it gets its inputs, and exactly two gates at depth $i-1$ to which it sends its output. If a gate evaluates to true, then it will send a *signal* to the output-gates, by sending the token to that gate's gadget. More precisely, for a gate of depth $i > 1$, the following signals are sent. If the gate evaluates to true, then the gate will send the token exactly 2^{i-1} times to each output gate. If the gate evaluates to false, then the gate will send the token exactly 0 times to each output gate. So the number of signals sent by a gate grows exponentially with the depth of that gate.

Fig. 10 shows our construction for an AND-gate of depth 2. It consists of a single switching node (with a succinct order). I_1 and I_2 are two input edges that come from the two inputs to this gate, and O_1 and O_2 are two output edges that go to the outputs of this gate. The control state is a special state that drives the construction, which will be described later. The switching order was generated by the following rules. For a gate at depth i, the switching order of an AND-gate is defined so that the first 2^i positions in the switching order go to control, the next 2^{i-1} positions in the switching order go to O_1, and the final 2^{i-1} positions in the switching order go to O_2. Observe that this switching order captures the behavior of an AND-gate. If the gadget receives 2^i signals from both inputs, then it sends 2^{i-1} signals to both outputs. On the other hand, if at least one of the two inputs sends no signals, then the gadget sends no signals to the outputs.

The same idea is used to implement OR-gates. Fig. 11 shows the construction for an OR-gate of depth 2. For an OR-gate of depth i we have that the first 2^{i-1} positions in the switching order go to O_1, the next 2^{i-1} positions in the switching order go to O_2, and the final 2^i positions in the switching order go to control. These conditions simulate an OR-gate. If either of the inputs produces 2^i input signals, then 2^{i-1} signals are sent to both outputs. If both inputs produce no signals, then no signals are sent to either output.

The control state and the depth 1 gates. Suppose that the inputs to the circuit are at depth d. The control state is a single switching node that has the following switching order. Each input edge to a gate at depth d refers to some bit contained in the bit-string B. The control state sends 2^d inputs using that edge if that bit is true, and 0 inputs using that edge if that bit is false. Once those signals have been sent, the control state moves the token to an absorbing failure state. The token begins at the control state.

Each gate at depth 1 is represented by a single state, and has the same structure and switch configuration as the gates at depth $i > 1$. The only difference is the destination of the edges O_1 and O_2. The gate g (which we must evaluate) sends all outputs to an absorbing target state. All other gates send all outputs back to the control state.

▶ **Lemma 16.** *The token reaches the target state if and only if the gate g evaluates to true when the circuit is evaluated on the bit-string B.*

Since these gadgets use exponential switching orders, this construction would have exponential size if written down in the explicit format. Note, however, that all of the switching orders can be written down in the succinct format in polynomially many bits. Moreover, the construction has exactly one switching state for each gate in the circuit, and three extra states for the control, target, and failure nodes. Every state in the construction

can be created using only the inputs and outputs of the relevant gate in the circuit, which means that the reduction can be carried out in logarithmic space. Thus, we have the following.

▶ **Theorem 17.** *Deciding the winner of a succinct zero-player RSG is* P*-hard under logspace reductions.*

References

1 Hoda Akbari and Petra Berenbrink. Parallel rotor walks on finite graphs and applications in discrete load balancing. In *Proc. of SPAA*, pages 186–195, 2013.

2 Anne Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.

3 Joshua Cooper, Benjamin Doerr, Tobias Friedrich, and Joel Spencer. Deterministic random walks on regular trees. *Random Structures Algorithms*, 37(3):353–366, 2010.

4 Joshua Cooper, Benjamin Doerr, Joel Spencer, and Gábor Tardos. Deterministic random walks on the integers. *European J. Combin.*, 28(8):2072–2090, 2007.

5 Joshua Cooper and Joel Spencer. Simulating a random walk with constant error. *Combin. Probab. Comput.*, 15(6):815–822, 2006.

6 Benjamin Doerr and Tobias Friedrich. Deterministic random walks on the two-dimensional grid. *Combin. Probab. Comput.*, 18(1-2):123–144, 2009.

7 Jérôme Dohrau, Bernd Gärtner, Manuel Kohler, Jiří Matoušek, and Emo Welzl. ARRIVAL: a zero-player graph game in NP ∩ coNP. In *A journey through discrete mathematics*, pages 367–374. Springer, Cham, 2017.

8 John Fearnley, Martin Gairing, Matthias Mnich, and Rahul Savani. Reachability switching games. *CoRR*, abs/1709.08991, 2017. arXiv:1709.08991.

9 Tobias Friedrich, Martin Gairing, and Thomas Sauerwald. Quasirandom load balancing. *SIAM J. Comput.*, 41(4):747–771, 2012.

10 Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubáček, Karel Král, Hagar Mosaad, and Veronika Slívová. ARRIVAL: Next stop in CLS. In *Proc. of ICALP*, 2018. To appear.

11 Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to parallel computation: P-completeness theory.* The Clarendon Press, Oxford University Press, New York, 1995.

12 Jan Friso Groote and Bas Ploeger. Switching graphs. *Internat. J. Found. Comput. Sci.*, 20(5):869–886, 2009.

13 Alexander E. Holroyd, Lionel Levine, Karola Mészáros, Yuval Peres, James Propp, and David B. Wilson. Chip-firing and rotor-routing on directed graphs. In *In and out of equilibrium. 2*, volume 60 of *Progr. Probab.*, pages 331–364. Birkhäuser, Basel, 2008.

14 Alexander E. Holroyd and James Propp. Rotor walks and Markov chains. In *Algorithmic probability and combinatorics*, volume 520 of *Contemp. Math.*, pages 105–126. Amer. Math. Soc., Providence, RI, 2010.

15 Karthik C. S. Did the train reach its destination: The complexity of finding a witness. *Inf. Process. Lett.*, 121:17–21, 2017.

16 Bastian Katz, Ignaz Rutter, and Gerhard Woeginger. An algorithmic study of switch graphs. *Acta Inform.*, 49(5):295–312, 2012.

17 Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. of CAV*, pages 585–591, 2011.

18 Christoph Meinel. Switching graphs and their complexity. In *Proc. MFCS 1989*, volume 379 of *Lecture Notes Comput. Sci.*, pages 350–359, 1989.

19 Vyatcheslav B. Priezzhev, Deepak Dhar, Abhishek Dhar, and Supriya Krishnamurthy. Eulerian walkers as a model of self-organized criticality. *Phys. Rev. Lett.*, 77(25):5079, 1996.

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

21 Klaus Reinhardt. The simple reachability problem in switch graphs. In *Proc. of SOFSEM*, pages 461–472, 2009.

22 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Appl. Math.*, 8(1):85–89, 1984.

Costs and Rewards in Priced Timed Automata

Martin Fränzle
Department of Computing Science, University of Oldenburg, Germany
martin.fraenzle@informatik.uni-oldenburg.de

Mahsa Shirmohammadi
CNRS & LIS, France
mahsa.shirmohammadi@lis-lab.fr

Mani Swaminathan
Department of Computing Science, University of Oldenburg, Germany
mani.swaminathan@informatik.uni-oldenburg.de

James Worrell
Department of Computer Science, University of Oxford, UK
james.worrell@cs.ox.ac.uk

Abstract

We consider Pareto analysis of reachable states of multi-priced timed automata (MPTA): timed automata equipped with multiple observers that keep track of costs (to be minimised) and rewards (to be maximised) along a computation. Each observer has a constant non-negative derivative which may depend on the location of the MPTA.

We study the Pareto Domination Problem, which asks whether it is possible to reach a target location via a run in which the accumulated costs and rewards Pareto dominate a given objective vector. We show that this problem is undecidable in general, but decidable for MPTA with at most three observers. For MPTA whose observers are all costs or all rewards, we show that the Pareto Domination Problem is PSPACE-complete. We also consider an ε-approximate Pareto Domination Problem that is decidable without restricting the number and types of observers.

We develop connections between MPTA and Diophantine equations. Undecidability of the Pareto Domination Problem is shown by reduction from Hilbert's 10^{th} Problem, while decidability for three observers is shown by a translation to a fragment of arithmetic involving quadratic forms.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models

Keywords and phrases Priced Timed Automata, Pareto Domination, Diophantine Equations

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.125

Related Version A full version of this paper may be found at [10], `https://arxiv.org/abs/1803.01914`.

Acknowledgements to Dan Segal for discussions on [12] and the reviewers for their feedback.

1 Introduction

Multi Priced Timed Automata (MPTA) [5, 7, 8, 11, 17, 18, 19] extend priced timed automata [2, 3, 4, 6, 16] with *multiple observers* that capture the accumulation of costs and rewards along a computation. This extension allows to model multi-objective optimization problems beyond the scope of timed automata [1]. MPTA lie at the frontier between timed automata (for which reachability is decidable [1]) and linear hybrid automata (for which reachability is undecidable [14]). The observers exhibit richer dynamics than the clocks of timed automata

by not being confined to unit slope in locations, but may neither be queried nor reset while taking edges. This *observability restriction* has been exploited in [17] (under a cost-divergence assumption) for carrying out a *Pareto analysis* of reachable values of the observers.

In this paper we distinguish between observers that represent *costs* (to be minimised) and those that represent *rewards* (to be maximised). Formally, we partition the set $\mathcal{Y}$ of observers into cost and reward variables and say that $\gamma \in \mathbb{R}^{\mathcal{Y}}_{\geq 0}$ *Pareto dominates* $\gamma' \in \mathbb{R}^{\mathcal{Y}}_{\geq 0}$ if $\gamma(y) \leq \gamma'(y)$ for each cost variable y and $\gamma(y) \geq \gamma'(y)$ for each reward variable y. Then the *Pareto curve* corresponding to an MPTA consists of all undominated vectors γ that are reachable in an accepting location. While cost and reward variables are syntactically identical in the underlying automaton model, distinguishing between them changes the notion of Pareto domination and the associated decision problems.

We introduce in Section 3 a decision version of the problem of computing Pareto curves for MPTA, called the *Pareto Domination Problem*. Here, given a target vector $\gamma \in \mathbb{R}^{\mathcal{Y}}_{\geq 0}$, one asks to reach an accepting location with a valuation $\gamma' \in \mathbb{R}^{\mathcal{Y}}_{\geq 0}$ that Pareto dominates γ. This has not been addressed in prior work on Pareto analysis of MPTA [17], which considers only costs or only rewards. Other works on MPTA either do not address Pareto analysis [5, 8, 11, 18, 19], or have only discrete costs updated on edges [22], or are confined to a single clock [7].

Our first main result is that the Pareto Domination Problem is undecidable in general. The undecidability proof in Section 4 is by reduction from Hilbert's 10^{th} problem. Owing to the existence of so-called "universal Diophantine equations" (of degree 4 with 58 variables [15]), our proof shows undecidabililty of the Pareto Domination Problem for some fixed but large number of observers. Undecidability of the Pareto Domination Problem entails that one cannot compute an exact Pareto curve for an arbitrary MPTA.

We consider three different approaches to recover decidability of the Pareto Domination Problem, which all have a common foundation, namely a *monotone* VASS described in Sections 2 and 5, which simulates integer runs of a given MPTA. By analysing the semi-linear reachability set of this VASS we can reduce the Pareto Domination Problem to satisfiability of a class of bilinear mixed integer-real constraints. We then consider restrictions on MPTA and variants of the Pareto Domination Problem that allow us to solve this class of constraints.

We first show in Section 6 that restricting to MPTA with only costs or only rewards yields PSPACE-completeness of the Pareto Domination Problem. Here we are able to eliminate integer variables from our bilinear constraints, resulting in a formula of linear real arithmetic. This strengthens [17, Theorem 1 and Corollary 1], whose decision procedures (that exploit well-quasi-orders for termination) do not yield complexity bounds.

Next we confine the MPTA in Section 7 to at most three observers, but allow a mix of costs and rewards. Decidability is now achieved by eliminating real variables from the bilinear constraint system, thus reducing the Pareto Domination Problem to deciding the existence of positive integer zeros of a quadratic form, which is known to be decidable from [12].

We consider in Section 8 another method to restore decidability for general MPTA with arbitrarily many costs and rewards, by studying an approximate version of the Pareto Domination Problem, called the *Gap Domination Problem*. Similar to the setting of [9], the Gap Domination Problem represents the decision version of the problem of computing ε-Pareto curves. This problem, whose input includes a tolerance $\varepsilon > 0$ and a vector $\gamma \in \mathbb{R}^{\mathcal{Y}}_{\geq 0}$, places no requirement on the answer if γ is dominated and all solutions dominating γ do so with slack at most ε. We solve the Gap Domination Problem by relaxation and rounding applied to our bilinear system of constraints.

In this paper we consider only MPTA with non-negative rates. Our approach can be generalised to obtain decidability results also in the case of negative rates by extending our foundation in Sections 2 and 5 from monotone VASS to $\mathbb{Z}$-VASS [13].

2 Background

Quadratic Diophantine Equations. For later use we recall a decidable class of non-linear Diophantine problems. Consider the quadratic equation

$$\sum_{i,j=1}^{n} a_{ij} X_i X_j + \sum_{j=1}^{n} b_j X_j + c = 0 \tag{1}$$

whose coefficients a_{ij}, b_j, and c are rational numbers. Consider also the family of constraints

$$f_1(X_1, \ldots, X_n) \sim c_1 \wedge \ldots \wedge f_k(X_1, \ldots, X_n) \sim c_k\,, \tag{2}$$

where $f_1, \ldots, f_k$ are linear forms with rational coefficients, $c_1, \ldots, c_k \in \mathbb{Q}$, and $\sim \in \{<, \leq\}$.

▶ **Theorem 1** ([12]). *There is an algorithm that decides whether a given quadratic equation (1) and a family of linear inequalities (2) have a solution in $\mathbb{Z}^n$.*

Let us emphasize that in Theorem 1 at most one quadratic constraint is permitted. It is clear (e.g., by introducing a slack variable) that the theorem remains true if the equality symbol in (1) is replaced by any comparison operator in $\{<, \leq, >, \geq\}$.

Monotone VASS. A *monotone vector addition system with states* (monotone VASS) is a tuple $\mathcal{Z} = \langle n, Q, q_0, Q_f, \Sigma, \Delta \rangle$, where $n \in \mathbb{N}$ is the *dimension*, Q is a set of *states*, $q_0 \in Q$ is the *initial state*, $Q_f \subseteq Q$ is a set of *final states*, Σ is the set of *labels*, and $\Delta \subseteq Q \times \mathbb{N}^n \times \Sigma \times Q$ is the set of *transitions*.

Given such a monotone VASS $\mathcal{Z}$ as above, the family of sets $\mathrm{Reach}_{\mathcal{Z},q} \subseteq \mathbb{N}^n$, for $q \in Q$, is the minimal family (w.r.t. to set inclusion) of integer vectors such that $\mathbf{0} \in \mathrm{Reach}_{\mathcal{Z},q_0}$ and for all $q \in Q$, if $\boldsymbol{u} \in \mathrm{Reach}_{\mathcal{Z},q}$ and $(q, \boldsymbol{v}, \ell, p) \in \Delta$ for some $\ell \in L$, then $\boldsymbol{u} + \boldsymbol{v} \in \mathrm{Reach}_{\mathcal{Z},p}$. Finally we define the *reachability set* of $\mathcal{Z}$ to be $\mathrm{Reach}_{\mathcal{Z}} := \bigcup_{q \in Q_f} \mathrm{Reach}_{\mathcal{Z},q}$.

For every vector $\boldsymbol{v} \in \mathbb{N}^n$ and every finite set $P = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m\}$ of vectors in $\mathbb{N}^n$, we define the $\mathbb{N}$-*linear set* $S(\boldsymbol{v}, P) := \{\boldsymbol{v} + \sum_{i=1}^{m} a_i \boldsymbol{u}_i : a_1, \ldots, a_m \in \mathbb{N}\}$. We call $\boldsymbol{v}$ the *base vector* and $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m \in P$ the *period vectors* of the set.

The following proposition follows from [20, Proposition 4.3] (see [10, Appendix B.1]).

▶ **Proposition 2.** *Let $\mathcal{Z} = \langle n, Q, q_0, Q_f, \Sigma, \Delta \rangle$ be a monotone VASS. Then the set* $\mathrm{Reach}_{\mathcal{Z}}$ *can be written as a finite union of $\mathbb{N}$-linear sets $S(\boldsymbol{v}_1, P_1), \ldots, S(\boldsymbol{v}_k, P_k)$, where for $i = 1, \ldots, k$ the components of $\boldsymbol{v}_i$ and of each vector in P_i are bounded by $\mathrm{poly}(n, |Q|, M)^n$ in absolute value, where M is maximum absolute value of the entries of vectors in $\mathbb{N}^n$ occurring in Δ.*

3 Multi-Priced Timed Automata and Pareto Domination

Let $\mathbb{R}_{\geq 0}$ denote the set of non-negative real numbers. Given a set $\mathcal{X} = \{x_1, \ldots, x_n\}$ of *clocks*, the set $\Phi(\mathcal{X})$ of *clock constraints* is generated by the grammar $\varphi ::= \texttt{true} \mid x \leq k \mid x \geq k \mid \varphi \wedge \varphi$, where $k \in \mathbb{N}$ is a natural number and $x \in \mathcal{X}$. A *clock valuation* is a mapping $\nu : \mathcal{X} \to \mathbb{R}_{\geq 0}$ that assigns to each clock a non-negative real number. We denote by $\mathbf{0}$ the valuation such that $\mathbf{0}(x) = 0$ for all clocks $x \in \mathcal{X}$. We write $\nu \models \varphi$ to denote that ν satisfies the constraint φ. Given $t \in \mathbb{R}_{\geq 0}$, we let $\nu + t$ be the clock valuation such that $(\nu + t)(x) = \nu(x) + t$ for all clocks $x \in \mathcal{X}$. Given $\lambda \subseteq \mathcal{X}$, let $\nu[\lambda \leftarrow 0]$ be the clock valuation such that $\nu[\lambda \leftarrow 0](x) = 0$ if $x \in \lambda$, and $\nu[\lambda \leftarrow 0](x) = \nu(x)$ otherwise.

A *multi-priced timed automaton* (MPTA) is a tuple $\mathcal{A} = \langle L, \ell_0, L_f, \mathcal{X}, \mathcal{Y}, E, R \rangle$, where L is a finite set of *locations*, $\ell_0 \in L$ is an *initial location*, $L_f \subseteq L$ is a set of *accepting locations*,

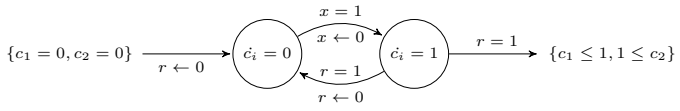


Figure 1 Predicates in curly brackets denote observer values enforced by initialisation, $c_i = 0$ with $i \in \{1, 2\}$, and by the Pareto constraint upon exit $\{c_1 \leq 1, 1 \leq c_2\}$. Denoting the initial value of clock x by x^*, the value of both c_1 and c_2 after n full traversals of the central cycle is nx^*. Meeting the final Pareto constraint from initial values thus requires that x^* be $\frac{1}{n}$ for some positive integer n.

$\mathcal{X}$ is a finite set of *clock variables*, $\mathcal{Y}$ is a finite set of *observers*, $E \subseteq L \times \Phi(\mathcal{X}) \times 2^{\mathcal{X}} \times L$ is the set of *edges*, $R : L \to \mathbb{N}^{\mathcal{Y}}$ is a *rate function*. Intuitively $R(\ell)$ is a vector that gives the rates of each observer in location ℓ.

A *state* of $\mathcal{A}$ is a triple (ℓ, ν, t) where ℓ is a location, ν a clock valuation, and $t \in \mathbb{R}_{\geq 0}$ is a *time stamp*. A *run* of $\mathcal{A}$ is an alternating sequence of states and edges $\rho = (\ell_0, \nu_0, t_0) \xrightarrow{e_1} (\ell_1, \nu_1, t_1) \xrightarrow{e_2} \ldots \xrightarrow{e_m} (\ell_m, \nu_m, t_m)$, where $t_0 = 0$, $\nu_0 = \mathbf{0}$, $t_{i-1} \leq t_i$ for all $i \in \{1, \ldots, m\}$, and $e_i = \langle \ell_{i-1}, \varphi, \lambda, \ell_i \rangle \in E$ is such that $\nu_{i-1} + (t_i - t_{i-1}) \models \varphi$ and $\nu_i = (\nu_{i-1} + (t_i - t_{i-1}))[\lambda \leftarrow 0]$ for $i = 1, \ldots, m$. The run is *accepting* if $\ell_m \in L_f$ and said to have *granularity* $\frac{1}{g}$ for a fixed $g \in \mathbb{N}$ if all $t_i \in \mathbb{Q}$ are positive integer multiples of $\frac{1}{g}$. The *cost* of such a run is a vector $\mathrm{cost}(\rho) \in \mathbb{R}^{\mathcal{Y}}$, defined by $\mathrm{cost}(\rho) = \sum_{j=0}^{m-1} (t_{i+1} - t_i) R(\ell_i)$.

Henceforth we will assume that the set $\mathcal{Y}$ of observers of a given MPTA is partitioned into a set $\mathcal{Y}_c$ of *cost variables* and a set $\mathcal{Y}_r$ of *reward variables*. With respect to this partition we define a *domination ordering* $\preccurlyeq$ on the set of valuations $\mathbb{R}^{\mathcal{Y}}$, where $\gamma \preccurlyeq \gamma'$ if $\gamma(y) \leq \gamma'(y)$ for all $y \in \mathcal{Y}_r$ and $\gamma'(y) \leq \gamma(y)$ for all $y \in \mathcal{Y}_c$. Intuitively $\gamma \preccurlyeq \gamma'$ (read γ' dominates γ) if γ' is at least as good as γ in all respects.

Given $\varepsilon > 0$ we define an *ε-domination ordering* $\preccurlyeq_\varepsilon$, where $\gamma \preccurlyeq_\varepsilon \gamma'$ (read γ' ε-dominates γ) if $\gamma(y) + \varepsilon \leq \gamma'(y)$ for all $y \in \mathcal{Y}_r$ and $\gamma'(y) + \varepsilon \leq \gamma(y)$ for all $y \in \mathcal{Y}_c$. We can think of $\gamma \preccurlyeq_\varepsilon \gamma'$ as denoting that γ' is better than γ by an additive factor of ε in all dimensions. In particular we clearly have that $\gamma \preccurlyeq_\varepsilon \gamma'$ implies $\gamma \preccurlyeq \gamma'$.

The *Pareto Domination Problem* is as follows. Given an MPTA $\mathcal{A}$ with a set $\mathcal{Y}$ of observers and a partition of $\mathcal{Y}$ into sets $\mathcal{Y}_c$ and $\mathcal{Y}_r$ of cost and reward variables, with a target $\gamma \in \mathbb{R}^{\mathcal{Y}}$, decide whether there is an accepting run ρ of $\mathcal{A}$ such that $\gamma \preccurlyeq \mathrm{cost}(\rho)$.

The *Gap Domination Problem* is a variant of the above problem in which the input additionally includes an accuracy parameter $\varepsilon > 0$. If there is some run ρ such that $\gamma \preccurlyeq_\varepsilon \mathrm{cost}(\rho)$ then the output should be "dominated" and if there is no run ρ such that $\gamma \preccurlyeq \mathrm{cost}(\rho)$ then the output should be "not dominated". In case neither of these alternatives hold (i.e., γ is dominated but not ε-dominated) then there is no requirement on the output.

In the (Pareto) Domination Problem the objective is to *reach* an accepting location while satisfying a family of upper-bound constraints on cost variables and lower-bound constraints on reward variables. We say that an instance of the problem is *pure* if all observers are cost variables or all are reward variables (and hence all constraints are upper bounds or all are lower bounds); otherwise we call the instance *mixed*. Our problem formulation involves only simple constraints on observers, i.e., those of the form $y \leq c$ or $y \geq c$ for $y \in \mathcal{Y}$. However such constraints can be used to encode more general linear constraints of the form $a_1 y_1 + \cdots + a_k y_k \sim c$, where $y_1, \ldots, y_k \in \mathcal{Y}$, $a_1, \ldots, a_k, c \in \mathbb{N}$ and $\sim \in \{\leq, \geq, =\}$. To do this one introduces a fresh observer to denote each linear term $a_1 y_1 + \cdots + a_k y_k$ (two fresh observers are needed for an equality constraint).

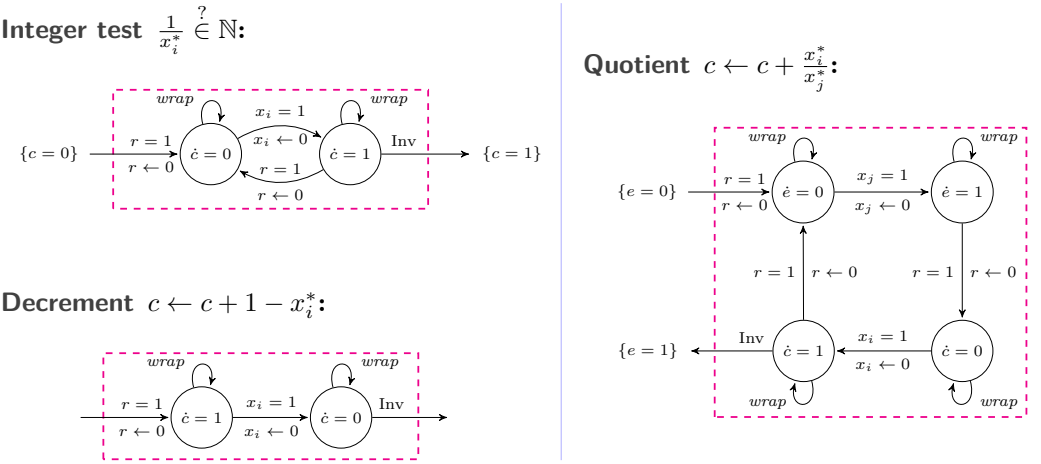


Figure 2 The *wrap* self-loop denotes a family of m wrapping edges, as in [14, Fig. 14], where the j-th edge has guard $x_j = 1$ and resets x_j. In the quotient gadget, e is a fresh observer, as is c in the integer test. The integer test and quotient gadgets are annotated with predicates in curly brackets indicating the initial values of observers on entering and their target values on exiting the gadget. Enforcing these target values through a corresponding Pareto constraint guarantees the desired behaviour of the gadget.

Note that we consider timed automata without *difference constraints* on clocks, i.e., without clock guards of the form $x_i - x_j \sim k$, for $k \in \mathbb{N}$. As discussed in [10, Appendix A] all our decidability and complexity results hold also in case of such constraints.

4 Undecidability of the Pareto Domination Problem

In this section we prove undecidability of the Pareto Domination Problem. To give some insight we first give in Figure 1 an MPTA, in which the Pareto constraint $c_1 \leq 1, c_2 \geq 1$ is used to enforce that when control enters the MPTA the value of clock x is $\frac{1}{n}$ for some positive integer n.

We prove undecidability of the Pareto Domination Problem by reduction from the satisfiability problem for a fragment of arithmetic given by a language $\mathcal{L}$ that is defined as follows. There is an infinite family of variables $X_1, X_2, X_3, \ldots$ and formulas are given by the grammar $\varphi ::= X = Y + Z \mid X = YZ \mid \varphi \wedge \varphi$, where X, Y, Z range over the set of variables. The satisfiability problem for $\mathcal{L}$ asks, given a formula φ, whether there is an assignment of positive integers to the variables that satisfies φ. In [10, Appendix B.2] we show that the satisfiability problem for $\mathcal{L}$ is undecidable by reduction from Hilbert's Tenth Problem.

▶ **Theorem 3.** *The Pareto Domination Problem is undecidable.*

Proof. Consider the following problem of reaching a single valuation in $\mathbb{R}^{\mathcal{Y}}_{\geq 0}$: given an MPTA $\mathcal{A} = \langle L, \ell_0, L_f, \mathcal{X}, \mathcal{Y}, E, R \rangle$, and target valuation $\gamma \in \mathbb{R}^{\mathcal{Y}}_{\geq 0}$, decide whether there is an accepting run ρ of $\mathcal{A}$ such that $\text{cost}(\rho) = \gamma$.

One can reduce the problem of reaching a given valuation to the Pareto Domination Problem as follows. Transform the MPTA $\mathcal{A}$ to an MPTA $\mathcal{A}'$ that has the same locations and edges as $\mathcal{A}$ but with two copies of each observer $y \in \mathcal{Y}$, with each copy having the same rate as y in each location. Formally $\mathcal{A}'$ has set of observers $\mathcal{Y}' = \{y_1, y_2 : y \in \mathcal{Y}\}$, where y_1 is a cost variable and y_2 is a reward variable. Then, defining $\gamma' \in \mathbb{R}^{\mathcal{Y}'}_{\geq 0}$ by $\gamma'(y_1) = \gamma'(y_2) = \gamma(y)$,

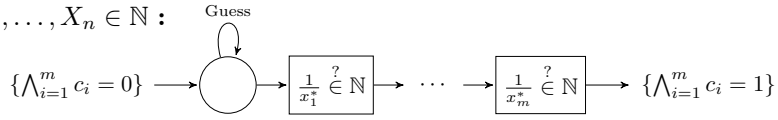
we have that $\mathcal{A}'$ has an accepting run ρ' such that $\mathrm{cost}(\rho')$ dominates γ' just in case A has an accepting run ρ such that $\mathrm{cost}(\rho) = \gamma$.

Now we give a reduction from the satisfiability problem for $\mathcal{L}$ to the problem of reaching a single valuation. Consider an $\mathcal{L}$-formula φ over variables $X_1, \ldots, X_m$. We define an MPTA $\mathcal{A}$ over the set of clocks $\mathcal{X} = \{x_1, \cdots, x_m, r\}$. Clock x_i corresponds to the variable X_i, for $i = 1, \ldots, m$, while r is a *reference clock*. The reference clock is reset whenever it reaches 1 and is not otherwise reset—thus it keeps track of global time modulo one. After an initialisation phase the remaining clocks $x_1, \ldots, x_m$ are likewise reset in a cyclic fashion, whenever they reach 1 and not otherwise. We denote by x_i^* the value of clock x_i whenever r is 1. During the initialisation phase the values x_i^* are established non-deterministically such that $0 < x_i^* \leq 1$. The idea is that $\frac{1}{x_i^*}$ represents the value of variable X_i in φ; in particular, x_i^* is the reciprocal of a positive integer. For each atomic sub-formula in φ the automaton $\mathcal{A}$ contains a gadget that checks that the guessed valuation satisfies the sub-formula.

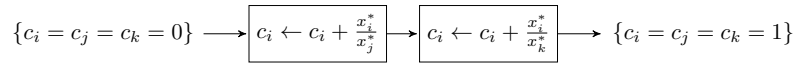
To present the reduction we first define three primitive gadgets. The first "integer test" gadget checks that the initial value x_i^* of clock x_i is a reciprocal of a positive integer, by adding wrapping edges on all clocks x_j other than x_i to the MPTA from Figure 1. The construction of each gadget is such that the precondition $r = 0$ holds when control enters the gadget and the postcondition $r = 1 \wedge \bigwedge_{j=1}^{m} x_j \leq 1$ holds on exiting the gadget. This last postcondition is abbreviated to Inv in the figures. For an observer c and $1 \leq i, j \leq m$, we define these three gadgets as in Figure 2.

In the following we show how to compose the three primitive operations in an MPTA to enforce the atomic constraints in the language $\mathcal{L}$. The initialisation automaton below is such that for $i = 1, \ldots, m$ the value x_i^* of clock x_i is such that $\frac{1}{x_i^*} \in \mathbb{N}$. Herein the Guess self-loop denotes a family of m edges, where the j-th edge non-deterministically resets clock x_j. Note that the incoming edge of the integer test gadget enforces $r = 1$ such that the initial guesses for the clocks x_i satisfy $x_i^* \in [0, 1]$. Of these, only reciprocals $\frac{1}{x_i^*} \in \mathbb{N}$ pass the subsequent series of integer tests.

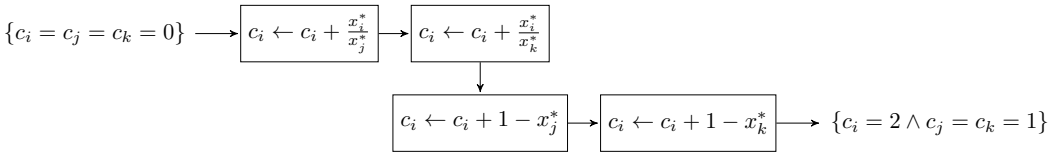
Initialisation $X_1, \ldots, X_n \in \mathbb{N}$ **:**



Sum $X_i = X_j + X_k$**:** According to the encoding of integer value X_n as clock value $x_n = \frac{1}{X_n}$, we have to enforce $\frac{1}{x_i^*} = \frac{1}{x_j^*} + \frac{1}{x_k^*}$, which is achieved by the following sequential combination of two quotient gadgets.



Product $X_i = X_j X_k$**:** The following gadget enforces $\frac{1}{x_i^*} = \frac{1}{x_j^*} \cdot \frac{1}{x_k^*}$:



The satisfiability problem for a given $\mathcal{L}$ formula φ can now directly be reduced to the problem of reaching a single valuation $\gamma \in \mathbb{R}_{\geq 0}^{\mathcal{Y}}$ by translating each of the conjuncts of φ into the corresponding above MPTA gadget. The valuation γ encodes the target costs of the respective gadgets. ◀

Let us remark that the proof of Theorem 3 shows that undecidability of the Pareto Domination Problem already holds in case all observers have rates in $\{0,1\}$. Separately we observe that undecidability also holds in the special case that exactly one observer is a cost variable and the others are reward variables, and likewise when exactly one observer is a reward variable and the others are cost variables, when allowing rates beyond $\{0,1\}$. The idea is to reduce the problem of reaching a particular valuation $\gamma \in \mathbb{R}^{\mathcal{Y}}_{\geq 0}$ in an MPTA $\mathcal{A}$ to that of dominating a valuation $\gamma' \in \mathbb{R}^{\mathcal{Y}'}_{\geq 0}$ in a derived MPTA $\mathcal{A}'$ with set of observers $\mathcal{Y}' = \mathcal{Y} \cup \{y_{\text{sum}}\}$, where y_{sum} is a fresh variable. In $\mathcal{A}'$ we designate all $y \in \mathcal{Y}$ as cost variables and y_{sum} as a reward variable, or vice versa. Valuation γ' is specified by $\gamma'(y) = \gamma(y)$ for all $y \in \mathcal{Y}$ and $\gamma'(y_{\text{sum}}) = \sum_{y \in \mathcal{Y}} \gamma(y)$. Automaton $\mathcal{A}'$ has the same locations, edges, and rate function as those of $\mathcal{A}$ except that $R'(y_{\text{sum}}) = \sum_{y \in \mathcal{Y}} R(y)$.

5 The Simplex Automaton

This section introduces the basic construction from which we derive our positive decidability results and complexity upper bounds.

Let $\mathcal{A} = \langle L, \ell_0, L_f, \mathcal{X}, \mathcal{Y}, E, R \rangle$ be an MPTA. For a sequence of edges $e_1, \ldots, e_m \in E$, define $\mathit{Runs}(e_1, \ldots, e_m) \subseteq \mathbb{R}^m_{\geq 0}$ to be the collection of sequences of timestamps $(t_1, \ldots, t_m) \in \mathbb{R}^m_{\geq 0}$ such that $\mathcal{A}$ has a run $\rho = (\ell_0, \nu_0, t_0) \xrightarrow{e_1} (\ell_1, \nu_1, t_1) \xrightarrow{e_2} \ldots \xrightarrow{e_m} (\ell_m, \nu_m, t_m)$. Recalling that by convention $t_0 = 0$ and $\nu_0 = \mathbf{0}$, once the edges $e_1, \ldots, e_m$ have been fixed then the run ρ is determined solely by the timestamps $t_1, \ldots, t_m$. When the sequence of edges $e_1, \ldots, e_m$ is understood, we call such a sequence of timestamps a run.

▶ **Proposition 4.** *$\mathit{Runs}(e_1, \ldots, e_m) \subseteq \mathbb{R}^m_{\geq 0}$ is defined by a conjunction of difference constraints.*

The proof of Proposition 4 is in [10, Appendix B.3].

▶ **Proposition 5.** *$\mathit{Runs}(e_1, \ldots, e_m)$ is equal to the convex hull of the set of its integer points.*

Proof. Fix a positive integer M. From Proposition 4 it immediately follows that the set $\mathit{Runs}(e_1, \ldots, e_m) \cap [0, M]^m$ can be written as a conjunction of closed difference constraints $A\boldsymbol{t} \leq \boldsymbol{b}$, where A is an integer matrix, $\boldsymbol{t}$ the vector of time-stamps $t_1 \ldots t_m$, and $\boldsymbol{b}$ an integer vector. Given this, it follows that $\mathit{Runs}(e_1, \ldots, e_m) \cap [0, M]^m$, being a closed and bounded polygon, is the convex hull of its vertices. Moreover each vertex is an integer point since the matrix A here, being by Proposition 4 the incidence matrix of a balanced signed graph with half edges, is totally unimodular [21, Proposition 8A.5]. ◀

Proposition 6 shows that for Pareto reachability on an MPTA $\mathcal{A}$ with $|\mathcal{Y}| = d$ observers, it suffices to look at $d+1$-simplices of integer runs.

▶ **Proposition 6.** *For any run ρ of $\mathcal{A}$ there exists a set of at most $d+1$ integer-time runs S, all over the same sequence of edges as ρ, such that $\mathrm{cost}(\rho)$ lies in the convex hull of $\mathrm{cost}(S)$.*

Proof. Let ρ be a run of $\mathcal{A}$ over an edge-sequence $e_1, \ldots, e_m$ with time stamps $t_0, \ldots, t_m$, given by $\rho = (\ell_0, \nu_0, t_0) \xrightarrow{e_1} (\ell_1, \nu_1, t_1) \xrightarrow{e_2} \ldots \xrightarrow{e_m} (\ell_m, \nu_m, t_m)$. By Proposition 5, $(t_1, \ldots, t_m)$ lies in the convex hull of the set I of integer points in $\mathit{Runs}(e_1, \ldots, e_m)$.

Since the map $\mathit{cost} : \mathit{Runs}(e_1, \ldots, e_m) \to \mathbb{R}^d$ is linear we have that $\mathrm{cost}(\rho)$ lies in the convex hull of $\mathrm{cost}(I)$. Moreover by Carathéodory's Theorem there exists a subset $S \subseteq I$ of cardinality at most $d+1$ such that $\mathrm{cost}(\rho)$ lies in the convex hull of $\mathrm{cost}(S)$. ◀

We now exploit Proposition 6 by introducing the so-called *simplex automaton* $\mathcal{S}(\mathcal{A})$, which is a monotone VASS obtained from a given MPTA $\mathcal{A}$. The automaton $\mathcal{S}(\mathcal{A})$ generates $(d+1)$-tuples of integer-time runs of $\mathcal{A}$, such that each run in the tuple executes the same sequence of edges in $\mathcal{A}$ and the runs differ only in the times at which the edges are taken. The basic component underlying the definition of the simplex automaton is the *integer-time automaton* $\mathcal{Z}(\mathcal{A})$. This automaton is a monotone VASS that generates the integer-time runs of $\mathcal{A}$, using its counters to keep track of the running cost for each observer.

The definition of $\mathcal{Z}(\mathcal{A})$ is as follows. Let $\mathcal{A} = \langle L, \ell_0, L_f, \mathcal{X}, \mathcal{Y}, E, R\rangle$ be an MPTA. Let also $M_{\mathcal{X}} \in \mathbb{N}$ be a positive constant greater than the maximum clock constant in $\mathcal{A}$. We define a monotone VASS $\mathcal{Z}(\mathcal{A}) = \langle d, Q, q_0, Q_f, E, \Delta\rangle$, in which the dimension $d = |\mathcal{Y}|$, the set of states is $Q = L \times \{0, 1, \ldots, M_{\mathcal{X}}\}^{\mathcal{X}}$, the initial state is $q_0 = (\ell_0, \mathbf{0})$, the set of accepting states is $Q_f = L_f \times \{0, 1, \ldots, M_{\mathcal{X}}\}^{\mathcal{X}}$, the set of labels is E (i.e., the set of edges of the MPTA), and the transition relation $\Delta \subseteq Q \times \mathbb{N}^d \times E \times Q$ includes a transition $((\ell, \nu), t \cdot R(\ell), e, (\ell', \nu'))$ for every $t \in \{0, 1, \ldots, M_{\mathcal{X}}\}$ and edge $e = (\ell, \varphi, \lambda, \ell')$ in $\mathcal{A}$ s.t. $\nu \oplus t \models \varphi$ and $\nu' = (\nu \oplus t)[\lambda \leftarrow 0]$. Here $(\nu \oplus t)(x) = \min(\nu(x) + t, M_{\mathcal{X}})$ for all $x \in \mathcal{X}$. We then have:

▶ **Proposition 7.** *Given a valuation $\gamma \in \mathbb{R}^{\mathcal{Y}}_{\geq 0}$, there exists an integer-time accepting run ρ of $\mathcal{A}$ with $\mathrm{cost}(\rho) = \gamma$ if and only if $\gamma \in \mathrm{Reach}_{\mathcal{Z}(\mathcal{A})}$.*

The simplex automaton $\mathcal{S}(\mathcal{A})$ is built by taking $d+1$ copies of $\mathcal{Z}(\mathcal{A}) = \langle d, Q, q_0, Q_f, E, \Delta\rangle$ that synchronize on transition labels. Formally, $\mathcal{S}(\mathcal{A}) = \langle d(d+1), Q^{d+1}, \boldsymbol{q}_0, Q_f^{d(d+1)}, E, \boldsymbol{\Delta}\rangle$, where $\boldsymbol{q}_0 = (q_0, \ldots, q_0)$ and $\boldsymbol{\Delta} \subseteq Q^{d+1} \times \mathbb{Z}^{d(d+1)} \times E \times Q^{d+1}$ comprises those tuples $((q_1, \ldots, q_{d+1}), (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{d+1}), e, (q'_1, \ldots, q'_{d+1}))$ s.t. $(q_i, \boldsymbol{v}_i, e, q'_i) \in \Delta$ for all $i \in \{1, \ldots, d+1\}$.

From Propositions 6 and 7 we have:

▶ **Proposition 8.** *Given $\gamma \in \mathbb{R}^{\mathcal{Y}}_{\geq 0}$, there exists an accepting run ρ of $\mathcal{A}$ with $\mathrm{cost}(\rho) = \gamma$ if and only if there exists $(\gamma_1, \ldots, \gamma_{d+1}) \in \mathrm{Reach}_{\mathcal{S}(\mathcal{A})}$ with γ in the convex hull of $\{\gamma_1, \ldots, \gamma_{d+1}\}$.*

We now introduce the following "master system" of bilinear inequalities that expresses whether $\gamma \preccurlyeq \mathrm{cost}(\rho)$ for some accepting run ρ of $\mathcal{A}$.

$$\begin{array}{lll} \gamma & \preccurlyeq & \lambda_1\gamma_1 + \cdots + \lambda_{d+1}\gamma_{d+1} \\ (\gamma_1, \ldots, \gamma_{d+1}) & \in & \mathrm{Reach}_{\mathcal{S}(\mathcal{A})} \end{array} \qquad \begin{array}{lll} 1 & = & \lambda_1 + \cdots + \lambda_{d+1} \\ 0 & \leq & \lambda_1, \ldots, \lambda_{d+1} \end{array} \tag{3}$$

The system has real variables $\lambda_1, \ldots, \lambda_{d+1} \in \mathbb{R}^{\mathcal{Y}}_{\geq 0}$ and integer variables $\gamma_1, \ldots, \gamma_{d+1} \in \mathbb{N}^{\mathcal{Y}}$. The key property of the master system is stated in the following Proposition 9, which follows immediately from Proposition 8.

▶ **Proposition 9.** *Given a valuation $\gamma \in \mathbb{R}^{\mathcal{Y}}_{\geq 0}$ there is an accepting run ρ of $\mathcal{A}$ such that $\gamma \preccurlyeq \mathrm{cost}(\rho)$ if and only if the system of inequalities (3) has a solution.*

Given Proposition 9, the results of Section 4 imply that satisfiability of the master system (3) is not decidable in general. In the rest of the paper we pursue different approaches to showing decidability of restrictions and variants of the Pareto Domination Problem by solving appropriately restricted versions of (3).

6 Pareto Domination Problem with Pure Constraints

In this section we show that the Pareto Domination Problem is decidable in polynomial space for the class of MPTA in which the observers are all costs. We prove this complexity

upper bound by exhibiting for such an MPTA $\mathcal{A}$ and target $\gamma \in \mathbb{R}^{\mathcal{Y}}_{\geq 0}$ a positive integer M, whose bit-length is polynomial in the size of $\mathcal{A}$ and γ, such that there exists a run ρ of $\mathcal{A}$ reaching the target location with $\gamma \preccurlyeq \text{cost}(\rho)$ iff there exists such a run of granularity $\frac{1}{M_1}$ for some $M_1 \leq M$. To show this we rewrite the bilinear system of inequalities (3) into an equisatisfiable disjunction of linear systems of inequalities. We thus obtain a bound on the bit-length of any satisfying assignment of (3) from which we obtain the above granularity bound. A similar bound in case of all reward variables is obtained in [10, Appendix C].

Consider an MPTA $\mathcal{A} = \langle L, \ell_0, L_f, \mathcal{X}, \mathcal{Y}, E, R\rangle$. Recall that the reachability set $\text{Reach}_{\mathcal{S}(\mathcal{A})}$ can be written as a union of linear sets $S(\boldsymbol{v}_i, P_i)$, $i \in I$. More precisely, let $M_{\mathcal{Y}}$ be the maximum rate occurring in the rate function R of the given MPTA $\mathcal{A}$. We then have the following, see [10, Appendix B.4] for the proof.

▶ **Proposition 10.** *The set* $\text{Reach}_{\mathcal{S}(\mathcal{A})}$ *can be written as a finite union of linear sets* $\bigcup_{i\in I} S(\boldsymbol{v}_i, P_i)$ *such that for each* $i \in I$ *the base vectors* $\boldsymbol{v}_i$ *and period vectors in* P_i *have entries of magnitude bounded by* $\text{poly}(d, |L|, M_{\mathcal{Y}}, M_{\mathcal{X}})^{d(d+1)|\mathcal{X}|}$.

Suppose that the set of observers $\mathcal{Y}$ with $|\mathcal{Y}| = d$ is comprised exclusively of cost variables. We will apply Proposition 10 to analyse the Pareto Domination Problem. The key observation is that in this case we can equivalently rewrite the bilinear system (3) as a disjunction of linear systems of inequalities.

As a first step we can rewrite the constraint $(\gamma_1, \ldots, \gamma_{d+1}) \in \text{Reach}_{\mathcal{S}(\mathcal{A})}$ in (3) as a disjunction of constraints $(\gamma_1, \ldots, \gamma_{d+1}) \in S(\boldsymbol{v}_i, P_i)$, for $i \in I$. But since the period vectors in P_i are non-negative we can further observe that in order to satisfy the upper bound constraints on cost variables, the optimal choice of $(\gamma_1, \ldots, \gamma_{d+1}) \in S(\boldsymbol{v}_i, P_i)$ is the base vector $\boldsymbol{v}_i$. Thus we can treat $\gamma_1, \ldots, \gamma_{d+1}$ as a constant in (3).

Thus we rewrite (3) as a finite disjunction of systems of linear inequalities—one such system for each $i \in I$. For a given $i \in I$ let $\boldsymbol{v}_i = (\gamma_1^{(i)}, \ldots, \gamma_{d+1}^{(i)})$ be the base vector of the linear set $S(\boldsymbol{v}_i, P_i)$. The corresponding system of inequalities specialising (3) is

$$\gamma \preccurlyeq \lambda_1\gamma_1^{(i)} + \ldots + \lambda_{d+1}\gamma_{d+1}^{(i)},\ 1 = \lambda_1 + \cdots + \lambda_{d+1},\ 0 \leq \lambda_1, \ldots, \lambda_{d+1} \tag{4}$$

Recall that if a set of linear inequalities $A\boldsymbol{x} \geq \boldsymbol{a}$, $B\boldsymbol{x} > \boldsymbol{b}$ is feasible then it is satisfied by some $\boldsymbol{x} \in \mathbb{Q}^n$ of bit-length $\text{poly}(n, b)$, where b is the total bit-length of the entries of A, B, $\boldsymbol{a}$, and $\boldsymbol{b}$. Applying this bound and Proposition 10 we see that a solution of (4) can be written in the form $\lambda_1 = \frac{p_1}{g}, \ldots, \lambda_{d+1} = \frac{p_{d+1}}{g}$ for integers $p_1, \ldots, p_{d+1}, g$ of bit-length at most $\text{poly}(d, |\mathcal{X}|, |L|, \log(M_{\mathcal{Y}}), \log(M_{\mathcal{X}}))$. This entails that the cost vector $\lambda_1\gamma_1^{(i)} + \ldots + \lambda_{d+1}\gamma_{d+1}^{(i)}$ arises from a run of $\mathcal{A}$ with granularity $\frac{1}{g}$, thus indirectly addressing the open problem stated in [17, Section 8] on the granularity of optimal runs in MPTA.

Together with Proposition 10, this yields PSPACE-membership for the Pareto Domination Problem. As reachability in timed automata is already PSPACE-hard [1] we have:

▶ **Theorem 11.** *The Pareto Domination Problem with pure constraints is PSPACE-complete.*

7 Pareto Domination Problem with Three Mixed Observers

In this section we consider the Pareto Domination Problem for MPTA with three observers. In the case of three cost variables or three reward variables the results of Section 6 apply. Below we show decidability for two cost variables and one reward variable. The similar case of two reward variables and one cost variable is handled in [10, Appendix E].

Consider an instance of the Pareto Domination Problem given by an MPTA $\mathcal{A}$ with $|\mathcal{Y}| = 3$ observers, and a target vector $\gamma \in \mathbb{R}^{\mathcal{Y}}_{\geq 0}$. Our starting point is again Proposition 9. To apply

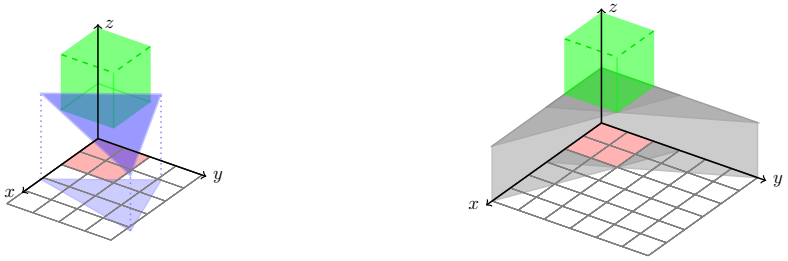


Figure 3 The target T is the green rectangular region and the blue region is S. The pink region is $\pi(T)$ and the light blue region $\pi(S)$. The grey region F is described in equation (5).

this proposition the idea is to eliminate the quantifiers over the real variables (the λ_i) in the system of equations (3) and thereby obtain a formula that lies in a decidable fragment of arithmetic (namely disjunctions of constraints of the form considered in Theorem 1).

To explain this quantifier-elimination step in more detail, let us identify $\mathbb{R}^{\mathcal{Y}}_{\geq 0}$ with $\mathbb{R}^3_{\geq 0}$. Denote by $T \subseteq \mathbb{R}^3_{\geq 0}$ the set of valuations that dominate a given fixed valuation $\gamma \in \mathbb{R}^3_{\geq 0}$. We can write $T = \{(x, y, z) \in \mathbb{R}^3_{\geq 0} : x \leq a \wedge y \leq b \wedge z \geq c\}$, where a, b, c are non-negative integer constants (see the left-hand side of Figure 3). We seek a quantifier-free formula of arithmetic that expresses that T meets a 4-simplex $S \subseteq \mathbb{R}^3_{\geq 0}$ given by the convex hull of $\{\gamma_1, \ldots, \gamma_4\}$, where $(\gamma_1, \ldots, \gamma_4) \in \mathrm{Reach}_{S(\mathcal{A})}$. However, since T is unbounded, it is clear that T meets a given 4-simplex S just in case it meets a face of S (which is a 3-simplex). Thus it will suffice to write a quantifier-free formula of arithmetic φ_T expressing that a 3-simplex in $\mathbb{R}^3_{\geq 0}$ meets T. Such a formula has nine free variables—one for each of the coordinates of the three vertices of S. We describe φ_T in the remainder of this section.

It is geometrically clear that S intersects T iff either S lies inside T, the boundary of S meets T, or the boundary of T meets S. More specifically we have the following proposition, whose proof is given in [10, Appendix B.5].

▶ **Proposition 12.** *Let $S \subseteq \mathbb{R}^3_{\geq 0}$ be a 3-simplex. Then $T \cap S$ is nonempty if and only if at least one of the following holds: (a) Some vertex of S lies in T; (b) Some bounding edge of S intersects either the face of T supported by the plane $x = a$ or the face of T supported by the plane $y = b$; (c) The bounding edge of T supported by the line $x = a \cap y = b$ intersects S.*

The following definition and proposition are key to expressing intersections of the form identified in Case (c) of Proposition 12 in terms of quadratic constraints. The idea is to identify a bounded region $F \subseteq \mathbb{R}^3_{\geq 0}$ such that in Case (c) one of the vertices of S lies in F. The proof of Proposition 13 can be found in [10, Appendix B.6].

Define a region $F \subseteq \mathbb{R}^3_{\geq 0}$ (depicted as the grey-shaded region on the right of Figure 3) by:

$$F = \{(x, y, z) \in \mathbb{R}^3_{\geq 0} \mid z < c \wedge (x + ay \leq a(b+1) \vee y + bx \leq b(a+1))\}. \tag{5}$$

Then we have:

▶ **Proposition 13.** *Let $S \subseteq \mathbb{R}^3_{\geq 0}$ be a 3-simplex such that $S \cap T$ is non-empty but none of the bounding edges of S meets T. Then some vertex of S lies in F.*

Denote by $\pi : \mathbb{R}^3 \to \mathbb{R}^2$ the projection of $\mathbb{R}^3$ onto the xy-plane, where $\pi(x, y, z) = (x, y)$ for all $x, y, z \in \mathbb{R}$. Write $\pi(T)$ and $\pi(S)$ for the respective images of T and S under π.

We write separate formulas $\varphi_T^{(1)}, \varphi_T^{(2)}, \varphi_T^{(3)}$, respectively expressing the three necessary and sufficient conditions for $T \cap S$ to be nonempty, as identified in Proposition 12. These are formulas of arithmetic whose free variables denote the coordinates of the three vertices of S.

Figure 4 Two cases for expressing that $\boldsymbol{c} \in \pi(S)$. The grey region is $\pi(F)$.

Some vertex of S lies in T. Denote the vertices of S by $\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}$. Formula $\varphi_T^{(1)}$ expresses that $\boldsymbol{p} \in T$ or $\boldsymbol{q} \in T$ or $\boldsymbol{r} \in T$. This is clearly a formula of linear arithmetic.

Some bounding edge of S meets a face of T. It is straightforward to obtain $\varphi_T^{(2)}$ given a formula ψ expressing that an arbitrary line segment $\boldsymbol{xy}$ in $\mathbb{R}^3_{\geq 0}$ meets a given fixed face of T. We outline such a formula in the rest of this sub-section. For concreteness we consider the face of T supported by the plane $x = a$, which maps under π to the line segment $L = \{(a, y) : 0 \leq y \leq b\}$. Formula ψ has six free variables, respectively denoting the coordinates of $\boldsymbol{x} = (x_1, x_2, x_3)$ and $\boldsymbol{y} = (y_1, y_2, y_3)$.

Formula ψ is a conjunction of two parts. The first part expresses that $\pi(\boldsymbol{x})\pi(\boldsymbol{y})$ meets L. Since the complement of $\pi(F)$ is a convex region in $\mathbb{R}^2_{\geq 0}$ that excludes $\pi(T)$ we have that either $\pi(\boldsymbol{x}) \in \pi(F)$ or $\pi(\boldsymbol{y}) \in \pi(F)$. Moreover since $\pi(F)$ contains finitely many integer points, we can write separate sub-formulas expressing that $\pi(\boldsymbol{x})\pi(\boldsymbol{y})$ meets L for each fixed value of $\pi(\boldsymbol{x}) \in \pi(F)$ and each fixed value of $\pi(\boldsymbol{y}) \in \pi(F)$. Each of these sub-formulas can then be written in linear arithmetic [10, Appendix D].

Suppose now that $\pi(\boldsymbol{x})\pi(\boldsymbol{y})$ meets L. Then the line $\boldsymbol{xy}$ meets the face of T supported by the plane $x = a$ iff the line in xz-plane connecting (x_1, x_3) and (y_1, y_3) passes above (a, c). This requirement is expressed by the quadratic constraint (8) in [10, Appendix D].

A bounding edge of T meets S. We proceed to describe the formula $\varphi_T^{(3)}$ expressing that the bounding edge E of T, supported by the line $x = a \cap y = b$, meets S. Note that image of E under the projection π is the single point $\boldsymbol{c} = (a, b)$. Thus E meets S just in case $\boldsymbol{c} \in \pi(S)$ and the point (a, b, c) lies below the plane affinely spanned by S. We describe two formulas that respectively express these requirements.

Denote the vertices of S by $\boldsymbol{p}$, $\boldsymbol{q}$, and $\boldsymbol{r}$. We first give a formula of linear arithmetic expressing that $\boldsymbol{c} \in \pi(S)$. Notice that if $\boldsymbol{c} \in \pi(S)$ then at least one vertex of $\pi(S)$ must lie in $\pi(F)$. We now consider two cases. The first case is that exactly one vertex of $\pi(S)$ (say $\pi(\boldsymbol{p})$) lies in $\pi(F)$. The second case is that at least two vertices of of $\pi(S)$ (say $\pi(\boldsymbol{p})$ and $\pi(\boldsymbol{q})$) lie in $\pi(F)$. The two cases are respectively denoted in Figure 4, that we refer to in the following.

In the first case we can express that $\boldsymbol{c} \in \pi(S)$ by requiring that the line segment $\pi(\boldsymbol{p})\pi(\boldsymbol{q})$ crosses the edge $\boldsymbol{f}_2\boldsymbol{c}$ and $\pi(\boldsymbol{p})\pi(\boldsymbol{r})$ crosses the edge $\boldsymbol{f}_1\boldsymbol{c}$. By writing a separate constraint for each fixed value of $\pi(\boldsymbol{p}) \in \pi(F)$ the above requirements can be expressed in linear arithmetic.

In the second case we can express that $\boldsymbol{c} \in \pi(S)$ by requiring that $\boldsymbol{c}$ lies on the left of each of the directed line segments $\pi(\boldsymbol{p})\pi(\boldsymbol{q})$, $\pi(\boldsymbol{q})\pi(\boldsymbol{r})$, and $\pi(\boldsymbol{r})\pi(\boldsymbol{p})$. By writing such a constraint for each fixed value of $\pi(\boldsymbol{p})$ and $\pi(\boldsymbol{q})$ in $\pi(F)$ we obtain, again, a formula of linear arithmetic [10, Appendix D].

It remains to give a formula expressing that (a, b, c) lies below the plane affinely spanned by $\boldsymbol{p}$, $\boldsymbol{q}$, and $\boldsymbol{r}$ under the assumption that $\boldsymbol{c} \in \pi(S)$. Note here that the above-described

formula expressing that $\pi(\boldsymbol{c}) \in \pi(S)$ specifies *inter alia* that $\pi(\boldsymbol{p})$, $\pi(\boldsymbol{q})$, and $\pi(\boldsymbol{r})$ are oriented counter-clockwise. Thus (a, b, c) lies below the plane affinely spanned by $\boldsymbol{p}$, $\boldsymbol{q}$, and $\boldsymbol{r}$ iff

$$\begin{vmatrix} q_1 - p_1 & r_1 - p_1 & a - p_1 \\ q_2 - p_2 & r_2 - p_2 & b - p_2 \\ q_3 - p_3 & r_3 - p_3 & c - p_3 \end{vmatrix} < 0$$

The above expression is cubic, but by Proposition 13 we may assume that $\boldsymbol{p}$ lies in the set F, which has finitely many integer points. Thus by a case analysis we may regard $\boldsymbol{p}$ as being fixed and so write the desired formula as a disjunction of atoms, each with a single quadratic term, whose satisfiability is known to be decidable from Theorem 1. This leads us to:

▶ **Theorem 14.** *The Pareto Domination Problem is decidable for at most three observers.*

Theorem 14 was proven by reduction to satisfiability of a system of arithmetic constraints with a *single* quadratic term. For the case of four observers this technique does not appear to yield arithmetic constraints in a known decidable class. Note that satisfiability of systems of constraints featuring two distinct quadratic terms is not known to be decidable in general.

In [10, Appendix F] we consider (a generalisation of) the Pareto Domination Problem for MPTA with at most two observers. In contrast to the case of three observers, we are able to show decidability for two observers by reduction to satisfiability in linear arithmetic.

8 Gap Domination Problem

In this section we give a decision procedure for the Gap Domination Problem. Given an MPTA $\mathcal{A}$, valuation $\gamma \in \mathbb{R}^{\mathcal{Y}}_{\geq 0}$, and a rational tolerance $\varepsilon > 0$, our procedure is such that

- if there is an accepting run ρ of $\mathcal{A}$ such that $\gamma \preccurlyeq_\varepsilon \mathrm{cost}(\rho)$ then we output "dominated";
- if there is no accepting run ρ of A such that $\gamma \preccurlyeq \mathrm{cost}(\rho)$ then we output "not dominated".

To do this, our approach is to find approximate solutions of the bilinear system (3) by relaxation and rounding.

Recall from Proposition 9 that (3) is satisfiable iff $\mathcal{A}$ has an accepting run ρ such that $\gamma \preccurlyeq \mathrm{cost}(\rho)$. Now we use the semi-linear decomposition of $\mathrm{Reach}_{\mathcal{S}(\mathcal{A})}$ to eliminate the constraints on integer variables from (3). In more detail, fix a decomposition of $\mathrm{Reach}_{\mathcal{S}(\mathcal{A})}$ as a union of linear sets and let $S := S(\boldsymbol{v}, P)$ be one such linear set, where $P = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\}$. Then we replace the constraint $(\gamma_1, \ldots, \gamma_{d+1}) \in \mathrm{Reach}_{\mathcal{S}(\mathcal{A})}$ in (3) with

$$(\gamma_1, \ldots, \gamma_{d+1}) = \boldsymbol{v} + n_1\boldsymbol{u}_1 + \cdots + n_k\boldsymbol{u}_k\,,$$

where $n_1, \ldots, n_k$ are variables ranging over $\mathbb{N}$. We thus obtain for each choice of S a bilinear system of inequalities φ_S of the form (6), where I and J are finite sets and for each $i \in I$ and $j \in J$, it holds that f_i, g_j are linear forms (i.e., polynomials of degree one with no constant terms) with non-negative integer coefficients and c_i and d_j are rational constants.

$$\begin{array}{rcll} f_i(n_1\lambda_1, n_1\lambda_2, \ldots, n_k\lambda_{d+1}) & \leq & c_i & (i \in I) \\ g_j(n_1\lambda_1, n_1\lambda_2, \ldots, n_k\lambda_{d+1}) & \geq & d_j & (j \in J) \\ n_1, \ldots, n_k \in \mathbb{N} & & & \end{array} \qquad \begin{array}{rcl} \lambda_1, \ldots, \lambda_{d+1} & \geq & 0 \\ \lambda_1 + \cdots + \lambda_{d+1} & = & 1 \end{array} \tag{6}$$

Fix a particular system φ_S, as depicted in (6). Let μ be the maximum coefficient of the f_i, $i \in I$. Given $T \subseteq \{1, \ldots, d+1\}$, we define the following constraint ψ_T on $\lambda_1, \ldots, \lambda_{d+1}$:

$$\psi_T := \bigwedge_{i \in T} \lambda_i \leq \tfrac{\varepsilon}{(d+1)k\mu} \wedge \bigwedge_{i \notin T} \lambda_i \geq \tfrac{\varepsilon}{(d+1)k\mu}\,.$$

Intuitively, ψ_T expresses that λ_i is "small" for $i \in T$ and "large" for $i \notin T$. Given any satisfying assignment of φ_S it is clear that $\lambda_1, \ldots, \lambda_{d+1}$ must satisfy φ_T for some $T \subseteq \{1, \ldots, d+1\}$.

Now fix a set $T \subseteq \{1, \ldots, d+1\}$ and consider the satisfiability of $\varphi_S \wedge \psi_T$. If $i \notin T$ then for any term $\lambda_i n_j$ that appears in an upper-bound constraint with right-hand side c in φ_S, we must have $n_j \leq \lceil \frac{c(d+1)\mu}{\varepsilon} \rceil$ in order for the constraint to be satisfied. Thus by enumerating all values of n_j we can eliminate this variable. By doing this we may assume that in $\varphi_S \wedge \psi_T$, for any term $\lambda_i n_j$ that appears on the left-hand side of an upper-bound constraint we have $i \in T$ and hence that λ_i must be "small" in any satisfying assignment.

The next step is relaxation—try to solve $\varphi_S \wedge \psi_T$ (after the above described elimination step), letting the variables $n_1, \ldots, n_k$ range over the non-negative reals. Recall here that the existential theory of real closed fields is decidable in polynomial space. If there is no real solution of $\varphi_S \wedge \psi_T$ for any S and T then there is certainly no solution over the naturals. and we can output "not dominated". On the other hand, if there is a run ρ with $\gamma \preccurlyeq_\varepsilon \mathrm{cost}(\rho)$ then for some S and T, the system $\varphi_S \wedge \psi_T$ will have a real solution in which moreover the inequalities $f_i(n_1\lambda_1, \ldots, n_k\lambda_{d+1}) \leq c_i$ for $i \in I$ all hold with slack at least ε. Given such a solution, replace n_j with $\lceil n_j \rceil$ for $j = 1, \ldots, k$. Consider the left-hand side $f_i(n_1\lambda_1, \ldots, n_k\lambda_{d+1})$ of an upper bound constraint in φ_S. Since the variables λ_i mentioned in such a linear form are small, the effect of rounding is to increase this term by at most ε. Hence the rounded valuation still satisfies φ_S thanks to the slack in the original solution. This then leads to Theorem 15 below:

▶ **Theorem 15.** *The Gap Domination Problem is decidable.*

References

1. R. Alur and D. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.
2. R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In M.-D. Di Benedetto and A. S-Vincentelli, editors, *HSCC*, volume 2034 of *LNCS*, pages 49–62. Springer, 2001.
3. G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. W. Vaandrager. Minimum-cost reachability for priced timed automata. In M.-D. Di Benedetto and A. S-Vincentelli, editors, *HSCC*, volume 2034 of *LNCS*, pages 147–161. Springer, 2001.
4. P. Bouyer, T. Brihaye, V. Bruyère, and J.-F. Raskin. On the optimal reachability problem of weighted timed automata. *Formal Methods in System Design*, 31(2):135–175, 2007.
5. P. Bouyer, E. Brinksma, and K. G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 32(1):3–23, 2008.
6. P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In F. Cassez and C. Jard, editors, *FORMATS*, volume 5215 of *LNCS*, pages 33–47. Springer, 2008.
7. P. Bouyer, K. G. Larsen, and N. Markey. Model checking one-clock priced timed automata. *Logical Methods in Computer Science*, 4:1–28, 2008.
8. T. Brihaye, V. Bruyère, and J.-F. Raskin. On model-checking timed automata with stopwatch observers. *Inf. Comput.*, 204(3):408–433, 2006.
9. I. Diakonikolas and M. Yannakakis. Small approximate pareto sets for biobjective shortest paths and other problems. *SIAM J. Comput.*, 39(4):1340–1371, 2009.
10. M. Fränzle, M. Shirmohammadi, M. Swaminathan, and J. Worrell. Costs and rewards in priced timed automata. *CoRR*, 2018. URL: `https://arxiv.org/abs/1803.01914`.
11. M. Fränzle and M. Swaminathan. Revisiting decidability and optimum reachability for multi-priced timed automata. In J. Ouaknine and F. W. Vaandrager, editors, *FORMATS*, volume 5813 of *LNCS*, pages 149–163. Springer, 2009.

12 F. Grunewald and D. Segal. On the integer solutions of quadratic equations. *Journal für die reine und angewandte Mathematik*, 569:13–45, 2004.

13 C. Haase and S. Halfon. Integer vector addition systems with states. In J. Ouaknine, I. Potapov, and J. Worrell, editors, *RP*, volume 8762 of *LNCS*, pages 112–124. Springer, 2014.

14 T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *J. Comput. Syst. Sci.*, 57(1):94–124, 1998.

15 J. P. Jones. Undecidable diophantine equations. *Bull. Amer. Math. Soc.*, 3:859–862, 1980.

16 K. G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In G. Berry, H. Comon, and A. Finkel, editors, *CAV*, volume 2102 of *LNCS*, pages 493–505. Springer, 2001.

17 K. G. Larsen and J. I. Rasmussen. Optimal reachability for multi-priced timed automata. *TCS*, 390(2-3):197–213, 2008.

18 V. Perevoshchikov. *Multi-weighted automata models and quantitative logics.* PhD thesis, University of Leipzig, 2015.

19 K. Quaas. *Kleene-Schützenberger and Büchi theorems for weighted timed automata.* PhD thesis, University of Leipzig, 2010.

20 A. W. To. Parikh images of regular languages: Complexity and applications. *CoRR*, 2010. URL: `http://arxiv.org/abs/1002.1464`.

21 T. Zaslavsky. Signed graphs. *Discrete Applied Mathematics*, 4(1):47–74, 1982.

22 Z. Zhang, B. Nielsen, K. G. Larsen, G. Nies, M. Stenger, and H. Hermanns. Pareto optimal reachability analysis for simple priced timed automata. In Z. Duan and L. Ong, editors, *ICFEM*, volume 10610 of *LNCS*, pages 481–495. Springer, 2017.

First-Order Interpretations of Bounded Expansion Classes

Jakub Gajarský[1]
Technical University Berlin, Germany

Stephan Kreutzer[1]
Technical University Berlin, Germany

Jaroslav Nešetřil[2]
Charles University, Prague, Czech Republic

Patrice Ossona de Mendez[2]
CAMS (CNRS, UMR 8557), Paris, France

Michał Pilipczuk[3]
University of Warsaw, Warsaw, Poland

Sebastian Siebertz[3]
University of Warsaw, Warsaw, Poland

Szymon Toruńczyk[4]
University of Warsaw, Warsaw, Poland

Abstract

The notion of bounded expansion captures uniform sparsity of graph classes and renders various algorithmic problems that are hard in general tractable. In particular, the model-checking problem for first-order logic is fixed-parameter tractable over such graph classes. With the aim of generalizing such results to dense graphs, we introduce classes of graphs with *structurally bounded expansion*, defined as first-order interpretations of classes of bounded expansion. As a first step towards their algorithmic treatment, we provide their characterization analogous to the characterization of classes of bounded expansion via low treedepth decompositions, replacing treedepth by its dense analogue called shrubdepth.

2012 ACM Subject Classification Theory of computation → Logic, Theory of computation → Finite Model Theory

Keywords and phrases Logical interpretations/transductions, structurally sparse graphs, bounded expansion

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.126

[1] J. Gajarský and S. Kreutzer are supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC Consolidator Grant DISTRUCT, grant agreement No 648527).

[2] J. Nešetřil and P. Ossona de Mendez are supported by CE-ITI P202/12/G061 of GACR and European Associated Laboratory (LEA STRUCO).

[3] M. Pilipczuk and S. Siebertz are supported by the National Science Centre of Poland (NCN) via POLONEZ grant agreement UMO-2015/19/P/ST6/03998, which has received funding from the European Union's Horizon 2020 research and innovation programme (Marie Skłodowska-Curie grant agreement No. 665778).

[4] Sz. Toruńczyk is supported by the NCN grant 2016/21/D/ST6/01485.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 126; pp. 126:1–126:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The interplay of methods from logic and graph theory has led to many important results in theoretical computer science, notably in algorithmics and complexity theory. The combination of logic and algorithmic graph theory is particularly fruitful in the area of *algorithmic meta-theorems*. Algorithmic meta-theorems are results of the form: *every computational problem definable in a logic $\mathcal{L}$ can be solved efficiently on any class of structures satisfying a property $\mathcal{P}$*. In other words, these theorems show that the *model-checking problem* for the logic $\mathcal{L}$ on any class $\mathcal{C}$ satisfying $\mathcal{P}$ can be solved efficiently, where *efficiency* usually means *fixed-parameter tractability*.

The archetypal example of an algorithmic meta-theorem is Courcelle's theorem [1, 2], which states that model-checking a formula φ of *monadic second-order logic* can be solved in time $f(\varphi) \cdot n$ on any graph with n vertices which comes from a fixed class of graphs of *bounded treewidth*, for some computable function f. Seese [27] proved an analogue of Courcelle's result for the model-checking problem of first-order logic on any class of graphs of bounded degree. Following this result, the complexity of first-order model-checking on specific classes of graphs has been studied extensively in the literature. See e.g. [17, 7, 19, 21, 4, 9, 10, 27, 8, 22, 28, 16, 5, 6, 13, 20]. One of the main goals of this line of research is to find a structural property $\mathcal{P}$ which precisely defines those graph classes $\mathcal{C}$ for which model checking of first-order logic is tractable.

So far, research on algorithmic meta-theorems has focused predominantly on *sparse* classes of graphs, such as classes of bounded *treewidth*, *excluding a minor* or which have *bounded expansion* or are *nowhere dense*. The concepts of *bounded expansion* and *nowhere denseness* were introduced by Nešetřil and Ossona de Mendez with the goal of capturing the intuitive notion of *sparseness*. See [25] for an extensive cover of these notions. The large number of equivalent ways in which they can be defined using either notions from combinatorics, theoretical computer science or logic, indicate that these two concepts capture some very natural limits of "well-behavedness" and algorithmic tractability. For instance, Grohe et al. [19] proved that if $\mathcal{C}$ is a class of graphs closed under taking subgraphs then model checking first-order logic on $\mathcal{C}$ is tractable if, and only if, $\mathcal{C}$ is nowhere dense (the lower bound was proved in [7]). As far as algorithmic meta-theorems for fixed-parameter tractability of first-order model-checking are concerned, this result completely solves the case for graph classes which are *closed under taking subgraphs*, which is a reasonable requirement for sparse but not for dense graph classes.

Consequently, research in this area has shifted towards studying the dense case, which is much less understood. While there are several examples of algorithmic meta-theorems on dense classes, such as for monadic second-order logic on classes of bounded *cliquewidth* [3] or for first-order logic on *interval graphs*, *partial orders*, classes of bounded *shrubdepth* and other classes, see e.g. [13, 11, 14, 12], a general theory of meta-theorems for dense classes is still missing. Moreover, unlike the sparse case, there is no canonical hierarchy of dense graph classes similar to the sparse case which could guide research on algorithmic meta-theorems in the dense world.

Hence, the main research challenge for dense model-checking is not only to prove tractability results and to develop the necessary logical and algorithmic tools. It is at least as important to define and analyze promising candidates for "structurally simple" classes of graph classes which are not necessarily sparse. This is the main motivation for the research in this paper. Since bounded expansion and nowhere denseness form the limits for tractability of certain problems in the sparse case, any extension of the theory should provide notions

which collapse to bounded expansion or nowhere denseness, under the additional assumption that the classes are closed under taking subgraphs. Therefore, a natural way of seeking such notions is to base them on the existing notions of bounded expansion or nowhere denseness.

In this paper, we take bounded expansion classes as a starting point and study two different ways of generalizing them towards dense graph classes preserving their good properties. In particular, we define and analyze classes of graphs obtained from bounded expansion classes by means of first-order interpretations and classes of graphs obtained by generalizing another, more combinatorial characterization of bounded expansion in terms of low treedepth colorings into the dense world. Our main structural result shows that these two very different ways of generalizing bounded expansion into the dense setting lead to the same classes of graphs. This is explained in greater detail below.

Interpretations and transductions. One possible way of constructing "well-behaved" and "structurally simple" classes of graphs is to use logical *interpretations*, or the related concept of *transductions* studied in formal language and automata theory. For our purpose, transductions are more convenient and we will use them in this paper. Intuitively, a *transduction* is a logically defined operation I which takes a structure as input and nondeterministically produces as output a target structure. In this paper we use *first-order* transductions, which involve first-order formulas (see Section 2 for details). Two examples of such transductions are graph complementation, and the squaring operation which, given a graph G, adds an edge between every pair of vertices at distance 2 from each other.

We postulate that if we start with a "structurally simple" class $\mathcal{C}$ of graphs, e.g. a class of bounded expansion or a nowhere dense class, and then study the graph classes $\mathcal{D}$ which can be obtained from $\mathcal{C}$ by first-order transductions, then the resulting classes should still have a simple structure and thus be well-behaved algorithmically as well as in terms of logic. In other words, the resulting classes are interesting graph classes with good algorithmic and logical properties, and which are certainly not sparse in general. For instance, a useful feature of transductions is that they provide a canonical way of reducing model-checking problems from the generated classes $\mathcal{D}$ to the original class $\mathcal{C}$, provided that given a graph $H \in \mathcal{D}$, we can effectively compute some graph $G \in \mathcal{C}$ that is mapped to H by the transduction. In general, this is a hard problem, requiring a *combinatorial* understanding of the structure of the resulting classes $\mathcal{D}$.

The above principle has so far been successfully applied in the setting of graph classes of bounded treewidth and monadic second-order transductions: it was shown by Courcelle, Makowsky and Rotics [3] that transductions of classes of bounded treewidth can be combinatorially characterized as classes of bounded cliquewidth. This, combined with Oum's result [26] gives a fixed-parameter algorithm for model-checking monadic second-order logic on classes of bounded cliquewidth. More recently, the same principle, but for first-order logic, has been applied to graphs of bounded degree [12], leading to a combinatorial characterization of first-order transductions of such classes, and to a model-checking algorithm.

Applying our postulate to bounded expansion classes yields the central notion of this paper: a class of graphs has *structurally bounded expansion* if it is the image of a class of bounded expansion under some fixed first-order transduction. This paper is a step towards a combinatorial, algorithmic, and logical understanding of such graph classes.

Low Shrubdepth Covers. The method of transductions is one way of constructing complex graphs out of simple graphs. A more combinatorial approach is the method of *decompositions* (or *colorings*) [25], which we reformulate below in terms of *covers*. This method can be used

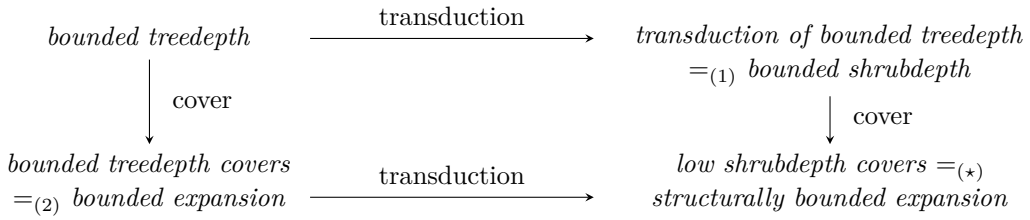


Figure 1 The nodes in the diagram depict properties of graph classes, and the arrows depict operations on properties of graph classes. Equality (1) is by [14]. Equality (2) is by [24]. Equality ($\star$) is the main result of this paper, Theorem 5.1.

to provide a characterization of bounded expansion classes in terms of very simple graph classes, namely classes of *bounded treedepth*. A class of graphs has bounded treedepth if there is a bound on the length of simple paths in the graphs in the class (see Section 2 for a different but equivalent definition). A class $\mathcal{C}$ has *low treedepth covers* if for every number $p \in \mathbb{N}$ there is a number N and a class of bounded treedepth $\mathcal{T}$ such that for every $G \in \mathcal{C}$, the vertex set $V(G)$ can be covered by N sets $U_1, \ldots, U_N$ so that every set $X \subseteq V(G)$ of at most p vertices is contained in some U_i, and for each $i = 1, \ldots, N$, the subgraph of G induced by U_i belongs to $\mathcal{T}$. A consequence of a result by Nešetřil and Ossona de Mendez [24] on a related notion of *low treedepth colorings* is that a graph class has bounded expansion if, and only if, it has low treedepth covers.

The decomposition method allows to lift algorithmic, logical, and structural properties from classes of bounded treedepth to classes of bounded expansion. For instance, this was used to show tractability of first-order model-checking on bounded expansion classes [7, 18].

An analogue of treedepth in the dense world is the concept of *shrubdepth*, introduced in [14]. Shrubdepth shares many of the good algorithmic and logical properties of treedepth. This notion is defined combinatorially, in the spirit of the definition of cliquewidth, but can be also characterized by logical means, as first-order transductions of classes of bounded treedepth. Applying the method of decompositions to the notion of shrubdepth leads to the following definition. A class $\mathcal{C}$ of graphs has *low shrubdepth covers* if for every number $p \in \mathbb{N}$ there is a number N and a class $\mathcal{B}$ of bounded shrubdepth such that for every $G \in \mathcal{C}$, there is a *p-cover* of G consisting of N sets $U_1, \ldots, U_N \subseteq V(G)$, so that every set $X \subseteq V(G)$ of at most p vertices is contained in some U_i and for each $i = 1, \ldots, N$, the subgraph of G induced by U_i belongs to $\mathcal{B}$. Shrubdepth properly generalizes treedepth and consequently classes admitting low shrubdepth covers properly extend bounded expansion classes.

It was observed earlier [23] that for every fixed $r \in \mathbb{N}$ and every class $\mathcal{C}$ of bounded expansion, the class of rth power graphs G^r of graphs from $\mathcal{C}$ (the rth power of a graph is a simple first-order transduction) admits low shrubdepth colorings.

Our contributions. Our main result, Theorem 5.1, states that the two notions introduced above are the same: a class of graphs $\mathcal{C}$ has structurally bounded expansion if, and only if, it has bounded shrubdepth covers. That is, transductions of classes of bounded expansion are the same as classes with low shrubdepth covers (cf. Figure 1). This gives a combinatorial characterization of structurally bounded expansion classes, which is an important step towards their algorithmic treatment.

One of the key ingredients of our proof is a quantifier-elimination result (Lemma 5.5) for transductions on classes of structurally bounded expansion. This result strengthens

in several ways similar results for bounded expansion classes due to Dvořák, Kráľ, and Thomas [7] and Grohe and Kreutzer [18]. Our assumption is more general, as they assume that $\mathcal{C}$ has bounded expansion, and here $\mathcal{C}$ is only required to have low shrubdepth covers. Also, our conclusion is stronger, as their results provide quantifier-free formulas involving some unary functions and unary predicates which are computable algorithmically, whereas our result shows that these functions can be defined using very restricted transductions. Quantifier-elimination results of this type proved to be useful for the model-checking problem on bounded expansion classes [7, 18], and this is also the case here.

As explained earlier, the transduction method allows to reduce the model-checking problem to the problem of finding inverse images under transductions, which is a hard problem in general and depends very much on the specific transduction. On the other hand, as we show, the cover method allows to reduce the model-checking problem for classes with low shrubdepth covers to the problem of computing a bounded shrubdepth cover of a given graph. In fact, as a consequence of our proof, in Theorem 6.1 we show that it is enough to compute a 2-cover of a given graph G from a structurally bounded expansion class, in order to obtain an algorithm for the model-checking problem for such classes. We strongly conjecture that such an algorithm exists and that therefore first-order model-checking is fixed-parameter tractable on any class of graphs of structurally bounded expansion. We leave this problem for future work.

2 Transductions

In this section, apart from recalling some background notions from logic and graph theory, we introduce the notion of transductions which we use in this paper.

Structures. We use standard logical notation and terminology with the following exceptions. A *signature* Σ is a finite set of relation and function symbols. We allow relations of any finite arity but only unary functions. We use boldface letters $\mathbf{A}$ for logical structures and denote their domains by $V(\mathbf{A})$. A structure $\mathbf{A}$ over Σ is defined as usual with the exception that each function symbol $f \in \Sigma$ is interpreted as a *partial* function $f^{\mathbf{A}} \colon V(\mathbf{A}) \rightharpoonup V(\mathbf{A})$. If $\mathbf{A}$ is a structure and $X \subseteq V(\mathbf{A})$ then we define the *substructure* of $\mathbf{A}$ induced by X in the usual way except that a unary function $f(x)$ in $\mathbf{A}$ becomes undefined on all $x \in X$ for which $f(x) \notin X$. The semantics of first-order logic is defined as usual, with the proviso that an atomic formula evaluates to false if any of the terms involved in it is undefined.

Graphs and colored graphs. We consider finite, simple and undirected graphs. These can be viewed as finite structures over the signature consisting of a binary relation symbol E, interpreted as the edge relation, in the usual way. For a finite label set Λ, by a Λ-*colored* graph we mean a graph enriched by a unary predicate U_λ for every $\lambda \in \Lambda$. We will follow the convention that if $\mathcal{C}$ is a class of colored graphs, then we implicitly assume that all graphs in $\mathcal{C}$ are over the same fixed finite signature. A rooted forest is an acyclic graph F together with a unary predicate $R \subseteq V(F)$ selecting one root in each connected component of F.

Transductions. We now define the notion of transduction used in the sequel. A *transduction* is a special type of first-order interpretation with set parameters, which we see here (from a computational point of view) as a nondeterministic operation that maps input structures to output structures. Transductions are defined as compositions of *atomic operations* listed below.

An **extension** operation is parameterized by a first-order formula $\varphi(x_1, \ldots, x_k)$ and a relation symbol R. Given an input structure $\mathbf{A}$, it outputs the structure $\mathbf{A}$ extended by the relation R interpreted as the set of k-tuples of elements satisfying φ in $\mathbf{A}$. A **restriction** operation is parameterized by a unary formula $\psi(x)$. Applied to a structure $\mathbf{A}$ it outputs the substructure of $\mathbf{A}$ induced by all elements satisfying ψ. A **reduct** operation is parameterized by a relation symbol R, and results in removing the relation R from the input structure. **Copying** is an operation which, given a structure $\mathbf{A}$ outputs a disjoint union of two copies of $\mathbf{A}$ extended with a new unary function which maps both copies of a vertex v back to v. A **function extension** operation is parameterized by a binary formula $\varphi(x, y)$ and a function symbol f, and extends a given input structure by a partial function f defined as follows: $f(x) = y$ if y is the unique vertex such that $\varphi(x, y)$ holds. Note that if there is no such y or more than one such y, then $f(x)$ is undefined. Finally, suppose σ is function that maps each structure $\mathbf{A}$ to a nonempty family $\sigma(\mathbf{A})$ of subsets of its universe. A **unary lift** operation, parametrized by σ, takes as input a structure $\mathbf{A}$ and outputs the structure $\mathbf{A}$ enriched by a unary predicate X interpreted by a nondeterministically chosen set $U \in \sigma(\mathbf{A})$.

We remark that a function extension operations can be simulated by extension operations, defining the graphs of the functions in the obvious way. They are, however, useful as a means of extending the expressive power of transductions in which only quantifier-free formulas are allowed, as defined below.

▶ **Definition 2.1.** *Transductions* are defined inductively: every atomic transduction is a transduction, and the composition of two transductions I and J is the transduction $\mathsf{I};\mathsf{J}$ that, given a structure $\mathbf{A}$, first applies J to $\mathbf{A}$ and then I to the output $\mathsf{J}(\mathbf{A})$. A transduction is *deterministic* if it does not use unary lifts. In this case, for every input structure there is exactly one output structure. A transduction is *almost quantifier-free* if all formulas that parameterize atomic operations comprising it are quantifier-free (observe that such transductions still can access elements that are not among its free variables via functions, hence, to avoid confusion we do not speak of quantifier-free transductions), and is *deterministic almost quantifier-free* if it additionally does not use unary lifts.

If $\mathcal{C}$ is a class of structures, we write $\mathsf{I}(\mathcal{C})$ for the class which contains all possible outputs $\mathsf{I}(\mathbf{A})$ for $\mathbf{A} \in \mathcal{C}$. We say that two transductions I and J are *equivalent* on a class $\mathcal{C}$ of structures if every possible output of $\mathsf{I}(\mathbf{A})$ is also a possible output of $\mathsf{J}(\mathbf{A})$, and vice versa, for every $\mathbf{A} \in \mathcal{C}$.

▶ **Example 2.2.** Let $\mathcal{C}$ be the class of rooted forests of depth at most d, for some fixed $d \in \mathbb{N}$. We describe an almost quantifier-free transduction which defines the *parent function* in $\mathcal{C}$. First, using unary lifts introduce $d+1$ unary predicates $D_0, ..., D_d$, where D_i marks the vertices of the input tree which are at distance i from a root. Next, using a function extension, define a function f which maps a vertex v in the input tree to its parent, or is undefined in case of a root. This can be done by a quantifier-free formula, which selects those pairs x, y such that x and y are adjacent and $D_i(x)$ implies $D_{i-1}(y)$.

It will sometimes be convenient to work with the encoding of bounded-depth trees and forests as node sets endowed with the parent function, rather than graphs with prescribed roots. As seen in Example 2.2, these two encodings can be translated to each other by means of almost quantifier-free transductions, which render them essentially equivalent.

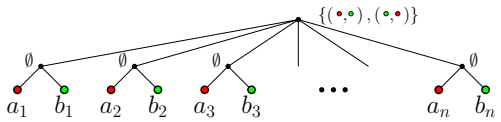
3 Treedepth, shrubdepth and bounded expansion

Treedepth. The *depth* of a node x in a rooted forest F is the number of edges on the root-to-x path in F. The *depth* of F is the maximal depth of any of its nodes. The *treedepth* of a graph G is the minimal depth of a rooted forest F with the same vertex set as G, such that for every edge vw of G, v is an ancestor of w, or w is an ancestor of v in F. A class $\mathcal{C}$ of graphs has *bounded treedepth* if there is a bound $d \in \mathbb{N}$ such that every graph in $\mathcal{C}$ has treedepth at most d. Equivalently, there is some number k such that no graph in $\mathcal{C}$ has a simple path of length k [25].

Shrubdepth. The following notion of *shrubdepth* has been proposed in [14] as a dense analogue of treedepth. Originally, shrubdepth was defined using the notion of *tree-models*. We present an equivalent definition based on the notion of *connection models*, introduced in [14] under the name of *m-partite cographs* of bounded depth.

▶ **Definition 3.1.** A *connection model* with labels from Λ is a rooted labeled tree T in which every leaf x is assigned a label $\lambda(x) \in \Lambda$, and every non-leaf node v is labeled by a (symmetric) binary relation $C(v) \subseteq \Lambda \times \Lambda$. Such a model defines a graph G on the leaves of T, in which two distinct leaves x and y are connected by an edge if and only if $(\lambda(x), \lambda(y)) \in C(v)$, where v is the common ancestor of x and y of largest depth. We say that T is a *connection model* of the resulting graph G.

▶ **Example 3.2.** Fix $n \in \mathbb{N}$, and let G_n be the bi-complement of a matching of order n, i.e., the bipartite graph with nodes $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$, such that a_i is adjacent to b_j if, and only if, $i \neq j$. A connection model for G_n is shown below:



▶ **Definition 3.3.** A class $\mathcal{C}$ of graphs has *bounded shrubdepth* if there is an $h \in \mathbb{N}$ and a finite set Λ such that every $G \in \mathcal{C}$ has a connection model of depth at most h using labels from Λ. A class of colored graphs has bounded shrubdepth if the class of underlying graphs has bounded shrubdepth.

Note that contrary to other graph parameters, it is meaningless to speak about the shrubdepth of a single graph. Rather, shrubdepth measures directly the complexity of a class.

Shrubdepth can be equivalently defined in terms of another graph parameter, as follows.

▶ **Definition 3.4.** Given a graph G and a set of vertices $W \subseteq V(G)$, the graph obtained by *flipping the adjacency within* W is the graph G' with vertices $V(G)$ and edge set which is the symmetric difference of the edge set of G and the edge set of the clique on W. The *subset-complementation depth*, or *SC-depth*, of a graph is defined inductively as follows: (1) a graph with one vertex has SC-depth 0, and (2) for $d \geqslant 1$ a graph G has SC-depth at most d if there is a set $W \subseteq V(G)$ of vertices such that in the graph obtained from G by flipping the adjacency within W all connected components have SC-depth at most $d-1$.

The notion of SC-depth leads to a natural notion of decompositions. An *SC-decomposition* of a graph G of SC-depth at most d is a rooted tree T of depth d with leaf set $V(G)$, equipped with unary predicates $W_0, \ldots, W_d$ on the leaves. Each child s of the root in T corresponds to a connected component C_s of the graph G' obtained from G by flipping the adjacency within W_0, such that the subtree of T rooted at s, together with the unary predicates $W_1, \ldots, W_d$ restricted to $V(C_s)$, form an SC-decomposition of C_s.

We will make use of the following properties, where the first one follows from the definition of shrubdepth, and the remaining ones follow from [14].

▶ **Proposition 3.5.** *Let $\mathcal{C}$ be a class of graphs. Then:*

1. *If $\mathcal{C}$ has bounded shrubdepth, then the class of all induced subgraphs of graphs from $\mathcal{C}$ also has bounded shrubdepth.*
2. *$\mathcal{C}$ has bounded shrubdepth if and only if for some $d \in \mathbb{N}$ all graphs in $\mathcal{C}$ have SC-depth at most d.*
3. *If $\mathcal{C}$ has bounded treedepth, then $\mathcal{C}$ has bounded shrubdepth.*
4. *If $\mathcal{C}$ has bounded shrubdepth and I is a transduction that outputs colored graphs, then $\mathsf{I}(\mathcal{C})$ has bounded shrubdepth.*

Bounded expansion. A graph H is a *depth-r minor* of a graph G if H can be obtained from a subgraph of G by contracting mutually disjoint connected subgraphs of radius at most r. A class $\mathcal{C}$ of graphs has *bounded expansion* if there is a function $f : \mathbb{N} \to \mathbb{N}$ such that $\frac{|E(H)|}{|V(H)|} \leqslant f(r)$ for every $r \in \mathbb{N}$ and every depth-r minor H of a graph from $\mathcal{C}$. Examples include the class of planar graphs, or any class of graphs with bounded maximum degree.

The connection between treedepth and graph classes of bounded expansion can be established via *treedepth-p colorings*. For an integer p, a function $c : V(G) \to C$ is a treedepth-p coloring if, for every $i \leqslant p$ and set $X \subseteq V(G)$ with $|c(X)| = i$, the induced graph $G[X]$ has treedepth at most i. A graph class $\mathcal{C}$ has *low treedepth colorings* if for every $p \in \mathbb{N}$ there is a number N_p such that for every $G \in \mathcal{C}$ there exists a treedepth-p coloring $c\colon V(G) \to C$ with $|C| \leqslant N_p$.

▶ **Theorem 3.6** ([24])**.** *A class $\mathcal{C}$ of graphs has bounded expansion if, and only if, it has low treedepth colorings.*

4 Structurally bounded expansion and low shrubdepth covers

In this section we introduce two notions which generalize the concept of bounded expansion. First, we introduce classes of *structurally bounded expansion*. This notion arises from closing bounded expansion graph classes under transductions.

▶ **Definition 4.1.** A class $\mathcal{C}$ of graphs has *structurally bounded expansion* if there exists a class of graphs $\mathcal{D}$ of bounded expansion and a transduction I such that $\mathcal{C} \subseteq \mathsf{I}(\mathcal{D})$.

The second notion, *low shrubdepth covers*, arises from the low treedepth coloring characterization of bounded expansion (see Theorem 3.6) by replacing treedepth by its dense counterpart, shrubdepth. For convenience, we formally define this in terms of *covers*.

▶ **Definition 4.2.** A *cover* of a graph G is a family $\mathcal{U}_G$ of subsets of $V(G)$ such that $\bigcup \mathcal{U}_G = V(G)$. A cover $\mathcal{U}_G$ is a *p-cover*, where $p \in \mathbb{N}$, if every set of at most p vertices is contained in some $U \in \mathcal{U}_G$. If $\mathcal{C}$ is a class of graphs, then a (p-)cover of $\mathcal{C}$ is a family $\mathcal{U} = (\mathcal{U}_G)_{G \in \mathcal{C}}$, where $\mathcal{U}_G$ is a (p-)cover of G. The cover $\mathcal{U}$ is *finite* if $\sup\{|\mathcal{U}_G| \colon G \in \mathcal{C}\}$ is finite. Let $\mathcal{C}[\mathcal{U}]$ denote the class of graphs $\{G[U] \colon G \in \mathcal{C}, U \in \mathcal{U}_G\}$. We say that the cover $\mathcal{U}$ has *bounded treedepth* (respectively, bounded shrubdepth) if the class $\mathcal{C}[\mathcal{U}]$ has bounded treedepth (respectively, shrubdepth).

▶ **Example 4.3.** Let $\mathcal{T}$ be the class of trees and $p \in \mathbb{N}$ be a number. We construct a finite p-cover $\mathcal{U}$ of $\mathcal{T}$ which has bounded treedepth. Given a rooted tree T, let $\mathcal{U}_T = \{U_0, \ldots, U_p\}$, where U_i is the set of vertices of T whose depth is not congruent to i modulo $p+1$. Note that $T[U_i]$ is a forest of height p, and that $\mathcal{U}_T$ is a p-cover of T. Hence $\mathcal{U} = (\mathcal{U}_T)_{T \in \mathcal{T}}$ is a finite p-cover of $\mathcal{T}$ of bounded treedepth.

In analogy to low treedepth colorings, we can now characterize graph classes of bounded expansion using covers. We say that a class $\mathcal{C}$ of graphs has *low treedepth covers* if for every $p \in \mathbb{N}$ there is a finite p-cover $\mathcal{U}$ of $\mathcal{C}$ with bounded treedepth. The following lemma follows easily from Theorem 3.6.

▶ **Lemma 4.4.** *A class of graphs has bounded expansion if, and only if, it has low treedepth covers.*

We now define the second notion generalizing the concept of bounded expansion. The idea is to use low shrubdepth covers instead of low treedepth covers.

▶ **Definition 4.5.** A class $\mathcal{C}$ of structures has *low shrubdepth covers* if, and only if, for every $p \in \mathbb{N}$ there is a finite p-cover $\mathcal{U}$ of $\mathcal{C}$ with bounded shrubdepth.

It is easily seen that Lemma 4.4 together with Proposition 3.5 (3) imply that every class of bounded expansion has low shrubdepth covers.

5 Main Result

The main result of our paper is the followiong.

▶ **Theorem 5.1.** *A class of graphs has structurally bounded expansion if, and only if, it has low shrubdepth covers.*

We prove the left-to-right direction of Theorem 5.1 in Section 5.1 and the right-to-left direction in Section 5.2. But before we need some lemmas on classes of bounded shrubdepth.

The first lemma is an analogue of a known fact that every class of bounded cliquewidth which excludes a fixed bi-clique as a subgraph has bounded treewidth. The lemma is proved by an easy induction on the depth of the connection models.

▶ **Lemma 5.2.** *A class $\mathcal{C}$ of graphs has bounded treedepth if, and only if, $\mathcal{C}$ has bounded shrubdepth and all $G \in \mathcal{C}$ exclude some fixed bi-clique as a subgraph.*

The second lemma, Lemma 5.3 below, is substantially more involved, and is the combinatorial cornerstone of our approach. Essentially, it says that if $\mathcal{C}$ is a class of bounded shrubdepth, then a bounded-depth SC-decomposition of a graph $G \in \mathcal{C}$ can be computed using an *almost quantifier-free* transduction B. In other words, an SC-decomposition of a graph $G \in \mathcal{C}$ can be encoded in G using a finite number of unary predicates and reconstructed using function extension operations involving deterministic almost quantifier-free formulas.

▶ **Lemma 5.3.** *Let $\mathcal{B}$ be a class of graphs of bounded shrubdepth. Then there is a class $\mathcal{T}$ of colored trees of bounded height and a pair of transductions T and B such that T is almost quantifier-free, B is deterministic almost quantifier-free, $\mathsf{T}(\mathcal{B}) \subseteq \mathcal{T}$, $\mathsf{B}(\mathcal{T}) \subseteq \mathcal{B}$, and*

$$\mathsf{B}(\mathsf{T}(G)) = \{G\} \quad \text{for all} \quad G \in \mathcal{B} \quad \text{and} \quad \mathsf{T}(\mathsf{B}(t)) \ni t \quad \text{for all} \quad t \in \mathcal{T}.$$

Moreover, for any $G \in \mathcal{B}$, every $t \in \mathsf{T}(G)$ is an SC-decomposition of G.

The key ingredient in the proof of Lemma 5.3 is that we can provide an almost quantifier-free transduction that defines connected components in graphs of bounded shrubdepth, as formulated in the following lemma.

▶ **Lemma 5.4.** *Let $\mathcal{C}$ be a class of graphs of bounded shrubdepth. There is an almost quantifier-free transduction F such that for a given $G \in \mathcal{C}$, every output of F on G is equal to G enriched by a function $g\colon V(G) \to V(G)$ such that $g(v) = g(w)$ if, and only if, v and w are in the same connected component of G.*

In the proof of Lemma 5.4 we use the fact that bounded shrubdepth implies bounded diameter [14], and the connection between bounded shrubdepth and (bi-)cographs [15].

5.1 Structurally bounded expansion implies low shrubdepth covers

The left-to-right implication of Theorem 5.1 – that every graph class with structurally bounded expansion has low shrub-depth covers – follows from the two lemmas below, whose proofs are outlined later in this section.

▶ **Lemma 5.5.** *Let $\mathcal{C}$ be a class of colored graphs which has low shrubdepth covers. Then every transduction I is equivalent on $\mathcal{C}$ to an almost quantifier-free transduction J.*

▶ **Lemma 5.6.** *If a class of graphs $\mathcal{C}$ has low shrubdepth covers and J is an almost quantifier-free transduction which outputs graphs, then $\mathsf{J}(\mathcal{C})$ also has low shrubdepth covers.*

We first show how to conclude the left-to-right implication of Theorem 5.1 from Lemma 5.5 and Lemma 5.6. Let $\mathcal{C}$ be a class of bounded expansion and I be a transduction which outputs graphs. By Lemma 4.4, $\mathcal{C}$ has low treedepth covers and hence, by Lemma 3.5, also has low shrubdepth covers. Applying Lemma 5.5, we obtain an almost quantifier-free transduction J such that $\mathsf{I}(\mathcal{C}) = \mathsf{J}(\mathcal{C})$. By Lemma 5.6, we deduce that $\mathsf{J}(\mathcal{C}) = \mathsf{I}(\mathcal{C})$ has low shrubdepth covers, proving the left-to-right implication of Theorem 5.1.

In the remainder of this section we sketch the proofs of Lemma 5.5 and Lemma 5.6.

Proof outline for Lemma 5.5. Our proof of Lemma 5.5 is a quantifier elimination procedure similar to that of Dvořák et al. [7] and Grohe and Kreutzer [18]. First, we prove the statement for classes of colored trees of bounded depth, just as in [7, 18]. Lemma 5.3 allows us to immediately lift the statement to classes of bounded shrubdepth. Finally, this is lifted to classes with low shrubdepth covers. ◀

Proof outline for Lemma 5.6. To prove the lemma, we first observe that every almost quantifier-free transduction is equivalent to a transduction which consists of a sequence of unary lifts followed by a deterministic almost quantifier-free transduction. Since, by Proposition 3.5, adding colors (the result of applying unary lifts) to a graph class of bounded shrubdepth will again result in a graph class of bounded shrubdepth, it suffices to prove the lemma for a deterministic almost quantifier-free transduction and a class of colored graphs $\mathcal{C}$.

Consider a p-cover $\mathcal{U}$ of $\mathcal{C}$ of bounded shrubdepth. As a very special case, suppose that I is a single quantifier-free extension operation that does not use any functions in the formula governing it. Then $\mathsf{I}(G[W]) = \mathsf{I}(G)[W]$ for $G \in \mathcal{C}$ and any $W \subseteq V(G)$. As a consequence, $\mathcal{U}_G$ is a p-cover of $\mathsf{I}(G)$ and hence $\mathsf{I}(\mathcal{C}[\mathcal{U}]) = \mathsf{I}(\mathcal{C})[\mathcal{U}]$ is a p-cover of $\mathsf{I}(\mathcal{C})$. Moreover, by Proposition 3.5, $\mathsf{I}(\mathcal{C})[\mathcal{U}]$ has bounded shrubdepth, so $\mathcal{U}$ is a finite cover of $\mathcal{C}$ of bounded shrubdepth. As p is arbitrary, this proves that $\mathsf{I}(\mathcal{C})$ has bounded shrubdepth covers.

For a general deterministic almost quantifier-free transduction I, the equality $\mathsf{I}(G[W]) = \mathsf{I}(G)[W]$ may fail, due to the fact that I may involve functions which reach outside of W. However, the value of any term involved in I depends only on a bounded number of vertices of G. This leads to the following lemma.

▶ **Lemma 5.7.** *For every deterministic almost quantifier-free transduction I there is a number c such that the following holds. For every graph G and vertex v of $\mathsf{I}(G)$ there is a set $S_v \subseteq V(G)$ of size at most c such that for any sets U, W with $W \subseteq V(\mathsf{I}(G))$ and $U \subseteq V(G)$, if $U \supseteq \bigcup_{v \in W} S_v$, then*

$$\mathsf{I}(G)[W] = \mathsf{I}(G[U])[W].$$

We now proceed to the general case. Choose any $p \in \mathbb{N}$. Let c be as in Lemma 5.7. Let $\mathcal{U}$ be a finite $(p \cdot c)$-cover of $\mathcal{U}$ of bounded shrubdepth. For a graph $G \in \mathcal{C}$ and a set $U \in \mathcal{U}_G$, define $W_U \subseteq \mathsf{I}(G)$ as the set of all vertices $v \in \mathsf{I}(G)$ such that $S_v \subseteq U$, where S_v is as in Lemma 5.7. From the lemma and the fact that classes of bounded shrubdepth are closed under transductions and induced subgraphs it follows easily that letting $\mathcal{W}_{\mathsf{I}(G)} = \{W_U : U \in \mathcal{U}_G\}$ yields a p-cover of $\mathsf{I}(G)$, which collectively, for all $G \in \mathcal{C}$, form a p-cover of $\mathsf{I}(\mathcal{C})$ which has bounded shrubdepth. Since p is arbitrary, this proves the lemma. ◀

5.2 Low shrubdepth covers imply structurally bounded expansion

For the right-to-left implication of Theorem 5.1, we prove the following statement.

▶ **Lemma 5.8.** *Let $\mathcal{C}$ be a class of graphs with low shrubdepth covers. There is a pair of transductions S and I, where S is almost quantifier-free and I is deterministic almost quantifier-free, such that $\mathsf{S}(\mathcal{C})$ is a class of colored graphs of bounded expansion and $\mathsf{I}(\mathsf{S}(G)) = \{G\}$ for each $G \in \mathcal{C}$.*

Clearly, the lemma implies that $\mathcal{C}$ has structurally bounded expansion, since it can be obtained as a result of a transduction I to a class $\mathsf{S}(\mathcal{C})$ of bounded expansion. Thus, the right-to-left implication of Theorem 5.1 is a consequence of the lemma.

Proof outline. Let $\mathcal{U}$ be a finite 2-cover of $\mathcal{C}$, and let $N = \sup\{|\mathcal{U}_G| : G \in \mathcal{C}\}$. We define a transduction S as follows. For a given $G \in \mathcal{C}$, let $\{U_1, \ldots, U_N\} = \mathcal{U}_G$. Given G, the transduction S introduces unary predicates $U_1, \ldots, U_N$, and then, using the transduction T from Lemma 5.3, computes the union $\bigcup_{i=1}^{N} \mathsf{T}(G[U_i])$, which is a union of N trees glued along the leaves. It is easy to see that the resulting graph excludes the bi-clique $K_{N+1,N+1}$ as a subgraph. By Lemma 5.6, the class $\mathsf{S}(\mathcal{C})$ has low shrubdepth covers. Any p-cover of $\mathsf{S}(\mathcal{C})$ of bounded shrubdepth has bounded treedepth by Lemma 5.2. Therefore, $\mathsf{S}(\mathcal{C})$ has low treedepth covers, so has bounded expansion.

The transduction I is easily constructed from the transduction B from Lemma 5.3, allowing to reconstruct the graph G from the colored union of the trees produced by S. ◀

6 Algorithmic aspects

In this section we give a partial result about efficient computability of transductions on classes with structurally bounded expansion. When we refer to the size of a structure in the algorithmic context, we refer to its total size, i.e., the sum of its universe size and the total sum of sizes of tuples in its relations.

Call a class $\mathcal{C}$ of graphs of structurally bounded expansion *efficiently decomposable* if there is a finite 2-cover $\mathcal{U}$ of $\mathcal{C}$ and an algorithm that, given a graph $G \in \mathcal{C}$, in linear time computes the cover $\mathcal{U}_G$ and for each $U \in \mathcal{U}_G$, an SC-decomposition S_U of depth at most d of the graph $G[U]$, for some constant d depending only on $\mathcal{C}$. Our result is as follows.

▶ **Theorem 6.1.** *Suppose J is a deterministic transduction and $\mathcal{C}$ is a class of graphs that has structurally bounded expansion and is efficiently decomposable. Then given a graph $G \in \mathcal{C}$, one may compute $\mathsf{J}(G)$ in time linear in the size of the input plus the size of the output.*

We remark that instead of efficient decomposability we could assume that the 2-cover $\mathcal{U}_G$ of a graph G and corresponding SC-decompositions for all $U \in \mathcal{U}_G$ is given together with G as input. If only the cover is given but not the SC-decompositions, we would obtain cubic running time because bounded shrubdepth implies bounded cliquewidth and we can compute

an approximate clique decomposition in cubic time [26]. Then, SC-decompositions of small height are definable in monadic second-order logic, and hence they can be computed in linear time using the result of Courcelle, Makowski and Rotics [3].

Observe that the theorem implies that we can efficiently evaluate a first-order sentence and enumerate all tuples satisfying a formula $\varphi(x_1, \ldots, x_k)$ on the given input graph, since this amounts to applying the theorem to a transduction consisting of a single extension operation. This strengthens the analogous result of Kazana and Segoufin [21] for classes of bounded expansion.

Proof outline. We will make use of transductions S and I constructed in the proof of Lemma 5.8. Recall that $\mathsf{S}(\mathcal{C})$ is a class of colored graphs of bounded expansion, I is deterministic, and $\mathsf{I}(\mathsf{S}(G)) = \{G\}$ for each $G \in \mathcal{C}$. Observe that J is equivalent to $\mathsf{S}; \mathsf{I}; \mathsf{J}$ on $\mathcal{C}$. Defining K as $\mathsf{I}; \mathsf{J}$, we get that $\mathsf{J}(G) = \mathsf{K}(\mathsf{S}(G))$ for $G \in \mathcal{C}$. Moreover, since I is deterministic, it follows that K is deterministic.

Let $G \in \mathcal{C}$ be an input graph. By efficient decomposability of $\mathcal{C}$, in linear time we can compute a cover $\mathcal{U}_G$ of G together with an SC-decomposition S_U of depth at most d of $G[U]$, for $U \in \mathcal{U}_G$. Each S_U is a colored tree, and by the construction described in the proof of Lemma 5.8, the trees S_U for $U \in \mathcal{U}_G$, glued along the leaves form a structure belonging to $\mathsf{S}(G)$. As $\mathsf{J}(G) = \mathsf{K}(\mathsf{S}(G))$, it suffices to apply the enumeration result of Kazana and Segoufin for classes of bounded expansion [21] to the colored graph $\mathsf{S}(G)$ and to all formulas occurring in the transduction K. ◀

7 Conclusion

In this paper we have provided a natural combinatorial characterization of graph classes that are first-order transductions of bounded expansion classes of graphs. Our characterization parallels the known characterization of bounded expansion classes by the existence of low treedepth decompositions, by replacing the notion of treedepth by shrubdepth. We believe that we have thereby taken a big step towards solving the model-checking problem for first-order logic on classes of structurally bounded expansion.

On the structural side we remark that transductions of bounded expansion graph classes are just the same as transductions of classes of structures of bounded expansion (i.e., classes whose Gaifman graphs or whose incidence encodings have bounded expansion). On the other hand, it remains an open question to characterize classes of relational structures, rather than just graphs, which are transductions of bounded expansion classes. We are lacking the analogue of Lemma 5.3; the problem is that within the proof we crucially use the characterization of shrubdepth via SC-depth, which works well for graphs but is unclear for structures of higher arity.

Finally, observe that classes of bounded expansion can be characterized among classes with structurally bounded expansion as those which are bi-clique free. It follows, that every monotone (i.e., subgraph closed) class of structurally bounded expansion has bounded expansion. Exactly the same statement holds characterizing bounded treedepth among bounded shrubdepth, and the second item holds for treewidth vs cliquewidth. In particular, for monotone graph classes all pairs of notions collapse.

We do not know how to extend our results to nowhere dense classes of graphs, mainly due to the fact that we do not know whether there exists a robust quantifier-elimination procedure for these graph classes. Obtaining such a procedure remains an open problem of prime importance in this field of research.

References

1 B. Courcelle. Graph rewriting: an algebraic and logic approach. In *Handbook of Theoretical Computer Science*, volume 2, chapter 5, pages 142–193. Elsevier, Amsterdam, 1990.

2 B. Courcelle. The monadic second-order logic of graphs I: recognizable sets of finite graphs. *Inform. and Comput.*, 85:12–75, 1990.

3 B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33:125–150, 2000.

4 A. Dawar, M. Grohe, and S. Kreutzer. Locally excluding a minor. In *22nd Annual IEEE Symposium on Logic in Computer Science*, pages 270–279, 2007.

5 A. Durand and E. Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Transactions on Computational Logic (TOCL)*, 8(4):21, 2007.

6 A. Durand, N. Schweikardt, and L. Segoufin. Enumerating answers to first-order queries over databases of low degree. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 121–131. ACM, 2014.

7 Z. Dvořák, D. Kráľ, and R. Thomas. Testing first-order properties for subclasses of sparse graphs. *Journal of the ACM*, 60:5 Article 36, 2013.

8 K. Eickmeyer and K. Kawarabayashi. FO model checking on map graphs. In *Proceedings of the 21st International Symposium on Fundamentals of Computation Theory, FCT 2017*, pages 204–216, 2017.

9 J. Flum and M. Grohe. Fixed-parameter tractability, definability, and model checking. *SIAM Journal of Computing*, 31:113–145, 2001.

10 M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *Journal of the ACM*, 48:1148–1206, 2001.

11 J. Gajarský, P. Hliněný, D. Lokshtanov, J. Obdržálek, S. Ordyniak, M.S. Ramanujan, and S. Saurabh. Fo model checking on posets of bounded width. In *56th Annual Symposium on Foundations of Computer Science (FOCS), 2015*, pages 963–974. IEEE, 2015.

12 J. Gajarský, P. Hliněný, J. Obdržálek, D. Lokshtanov, and M. S. Ramanujan. A new perspective on fo model checking of dense graph classes. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 176–184. ACM, 2016.

13 R. Ganian, P. Hliněný, J. Obdržálek, J. Schwartz, J. Teska, and D. Kráľ. Fo model checking of interval graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 250–262. Springer, 2013.

14 R. Ganian, P. Hliněný, J. Nešetřil, J. Obdržálek, P. Ossona de Mendez, and R. Ramadurai. When trees grow low: Shrubs and fast MSO_1. In *MFCS 2012*, volume 7464 of *Lecture Notes in Computer Science*, pages 419–430. Springer-Verlag, 2012.

15 V. Giakoumakis and J.-M. Vanherpe. Bi-complement reducible graphs. *Adv. Appl. Math.*, 18:389–402, 1997.

16 M. Grohe. Generalized model-checking problems for first-order logic. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 12–26. Springer, 2001.

17 M. Grohe and S. Kreutzer. Methods for algorithmic meta-theorems. *Contemporary Mathematics*, 588, American Mathematical Society 2011.

18 M. Grohe and S. Kreutzer. Methods for algorithmic meta theorems. In *Model Theoretic Methods in Finite Combinatorics*, Contemporary mathematics, pages 181–206, 2011.

19 M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. *Journal of the ACM*, 64(3):17:1–17:32, 2017.

20 Petr Hlinený, Filip Pokrývka, and Bodhayan Roy. FO model checking of geometric graphs. In *12th International Symposium on Parameterized and Exact Computation, IPEC 2017*, pages 19:1–19:12, 2017. doi:10.4230/LIPIcs.IPEC.2017.19.

21 W. Kazana and L. Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *Proceedings of the 16th International Conference on Database Theory*, pages 10–20, 2013.

22 W. Kazana and L. Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 297–308. ACM, 2013.

23 O. Kwon, Mi. Pilipczuk, and S. Siebertz. On low rank-width colorings. In *43rd International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2017*, pages 372–385, 2017.

24 J. Nešetřil and P. Ossona de Mendez. Grad and classes with bounded expansion I. decompositions. *European Journal of Combinatorics*, 29(3):760–776, 2008.

25 J. Nešetřil and P. Ossona de Mendez. *Sparsity (Graphs, Structures, and Algorithms)*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012. 465 pages.

26 S. Oum. Approximating rank-width and clique-width quickly. *ACM Transactions on Algorithms (TALG)*, 5(1):10, 2008.

27 D. Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 5:505–526, 1996.

28 L. Segoufin and W. Kazana. First-order query evaluation on structures of bounded degree. *Logical Methods in Computer Science*, 7, 2011.

Randomized Sliding Window Algorithms for Regular Languages

Moses Ganardi
Universität Siegen, Germany
ganardi@eti.uni-siegen.de

Danny Hucke
Universität Siegen, Germany
hucke@eti.uni-siegen.de

Markus Lohrey
Universität Siegen, Germany
lohrey@eti.uni-siegen.de

Abstract

A sliding window algorithm receives a stream of symbols and has to output at each time instant a certain value which only depends on the last n symbols. If the algorithm is randomized, then at each time instant it produces an incorrect output with probability at most ϵ, which is a constant error bound. This work proposes a more relaxed definition of correctness which is parameterized by the error bound ϵ and the failure ratio ϕ: a randomized sliding window algorithm is required to err with probability at most ϵ at a portion of $1 - \phi$ of all time instants of an input stream. This work continues the investigation of sliding window algorithms for regular languages. In previous works a trichotomy theorem was shown for deterministic algorithms: the optimal space complexity is either constant, logarithmic or linear in the window size. The main results of this paper concerns three natural settings (randomized algorithms with failure ratio zero and randomized/deterministic algorithms with bounded failure ratio) and provide natural language theoretic characterizations of the space complexity classes.

2012 ACM Subject Classification Theory of computation → Streaming models

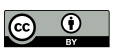
Keywords and phrases sliding windows, regular languages, randomized complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.127

Related Version A full version (with some additional results) can be found in [9], `https://arxiv.org/abs/1802.07600`.

1 Introduction

Sliding window algorithms process an input sequence $a_1 a_2 \cdots a_m$ from left to right and have at time t only direct access to the current symbol a_t. Moreover, at each time instant t the algorithm is required to compute a value that depends on the last n symbols. The value n is called the *window size* and the last n symbols form the *active window* at time t. In many streaming applications, data items are outdated after a certain time and the sliding window model is a simple way to model this. A general goal in the area of sliding window algorithms is to avoid the explicit storage of the window content (which requires $\Omega(n)$ bits), and, instead, to work in considerably smaller space, e.g. polylogarithmic space with respect to the window size n. A detailed introduction into the sliding window model can be found in [1, Chapter 8].



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 127; pp. 127:1–127:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Regular languages over sliding windows. In our recent papers [7, 8] we initiated the study of sliding window algorithms for regular languages. In general, a sliding window algorithm for a language $L \subseteq \Sigma^*$ decides at every time instant whether the active window belongs to L. In [8] we proved that for every regular language L the optimal space bound for a sliding window algorithm for L is either constant, logarithmic or linear in the window size. This trichotomy result resembles the well-known fact that every regular (even context-free) language has either polynomial or exponential growth. In [7] we also gave several characterizations for these space classes: A regular language has a sliding window algorithm with space $\mathcal{O}(\log n)$ if and only if it belongs to $\langle \mathbf{LI}, \mathbf{Len} \rangle$, which denotes the Boolean closure of the class **LI** of regular left ideals and the class **Len** of regular length languages. Moreover, a regular language has a sliding window algorithm that uses space $\mathcal{O}(1)$ if and only if it belongs to $\langle \mathbf{ST}, \mathbf{Len} \rangle$, where **ST** is the class of suffix-testable languages. The formal definitions of these and other language classes can be found in Section 2. The goal of this work is to extend the results from [7, 8] to randomized algorithms.

Main results. Consider a Monte-Carlo sliding window algorithm which can produce incorrect outputs. Abstracting away from the actual computation, we will view such a randomized sliding window algorithm (SWA for short) as a family $\mathcal{R} = (R_n)_{n \geq 0}$ of probabilistic automata, where R_n is the algorithm for window size n. We denote by $f(\mathcal{R}, n)$ the number of bits stored by R_n, which is the logarithm of the number of states. There are different ways to define correctness of $\mathcal{R}$ for a certain language L. The maybe most natural choice is to require that after reading an arbitrary input word $a_1 a_2 \cdots a_m$, the algorithm R_n correctly decides whether $a_{m-n+1} \cdots a_m \in L$ with probability at least $2/3$. This ensures that for every input stream and every time instant, one can be sure to get a correct answer with probability at least $2/3$. By a standard probability amplification argument, $2/3$ can be replaced by any probability strictly between $1/2$ and 1. With this definition (formal details can be found in Section 3) our first main result is the following, where **SF** denotes the class of all regular suffix-free languages (point (1) and (5) are from [8]).

▶ **Theorem 1.1.** *Let $L \subseteq \Sigma^*$ be a regular language.*

1. *If $L \in \langle \mathbf{ST}, \mathbf{Len} \rangle$, then L has a deterministic SWA $\mathcal{R}$ with $f(\mathcal{R}, n) = \mathcal{O}(1)$.*
2. *If $L \notin \langle \mathbf{ST}, \mathbf{Len} \rangle$, then $f(\mathcal{R}, n) \notin o(\log \log n)$ for every randomized SWA $\mathcal{R}$ for L.*
3. *If $L \in \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$, then L has a randomized SWA $\mathcal{R}$ with $f(\mathcal{R}, n) = \mathcal{O}(\log \log n)$.*
4. *If $L \notin \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$, then $f(\mathcal{R}, n) \notin o(\log n)$ for every randomized SWA $\mathcal{R}$ for L.*
5. *If $L \in \langle \mathbf{LI}, \mathbf{Len} \rangle$, then L has a deterministic SWA $\mathcal{R}$ with $f(\mathcal{R}, n) = \mathcal{O}(\log n)$.*
6. *If $L \notin \langle \mathbf{LI}, \mathbf{Len} \rangle$, then $f(\mathcal{R}, n) \notin o(n)$ for every randomized SWA $\mathcal{R}$ for L.*

One may argue that an algorithm which occasionally produces a wrong answer with probability $> 1/3$ is acceptable as well. This motivates the following definition: We say that a randomized algorithm for a certain language L and a window size n has *failure ratio* ϕ if for every input stream, the portion of all time instants where the algorithm gives a wrong answer with probability $> 1/3$ is bounded by ϕ. The second main result concerns algorithms with a bounded failure ratio. If we ask for an arbitrarily small non-zero failure ratio we get the following space dichotomy, where **PF** denotes the class of regular prefix-free languages:

▶ **Theorem 1.2.** *Let $L \subseteq \Sigma^*$ be a regular language.*

1. *If $L \in \langle \mathbf{LI}, \mathbf{PF}, \mathbf{Len} \rangle$ and $0 < \phi \leq 1$, then L has a randomized SWA with $f(\mathcal{R}, n) = \mathcal{O}(1)$ and failure ratio ϕ.*
2. *If $L \notin \langle \mathbf{LI}, \mathbf{PF}, \mathbf{Len} \rangle$, then there exists a failure ratio $0 < \phi \leq 1$ such that $f(\mathcal{R}, n) \notin o(n)$ for every randomized SWA $\mathcal{R}$ for L with failure ratio ϕ.*

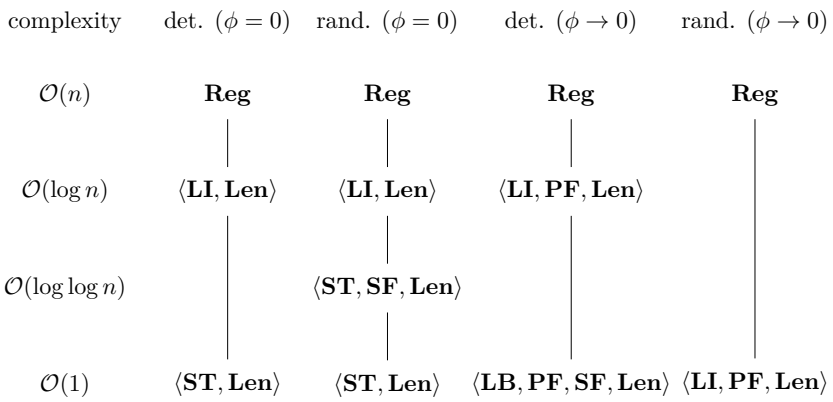


Figure 1 All language classes are defined in Section 2.

The notion of failure ratio makes sense for deterministic sliding window algorithms as well. Our third main result is a space trichotomy for deterministic sliding window algorithms having a bounded failure ratio. Let **LB** denote the class of left ideals generated by bifix-free (i.e., prefix- and suffix-free) regular languages. We then show:

▶ **Theorem 1.3.** *Let $L \subseteq \Sigma^*$ be a regular language.*

1. *If $L \in \langle \mathbf{LB}, \mathbf{PF}, \mathbf{SF}, \mathbf{Len} \rangle$ and $0 < \phi \leq 1$, then L has a deterministic SWA with $f(\mathcal{R}, n) = \mathcal{O}(1)$ and failure ratio ϕ.*
2. *If $L \notin \langle \mathbf{LB}, \mathbf{PF}, \mathbf{SF}, \mathbf{Len} \rangle$, then there exists a failure ratio $0 < \phi \leq 1$ such that $f(\mathcal{R}, n) \notin o(\log n)$ for every deterministic SWA $\mathcal{R}$ for L with failure ratio ϕ.*
3. *If $L \in \langle \mathbf{LI}, \mathbf{PF}, \mathbf{Len} \rangle$ and $0 < \phi \leq 1$, then L has a deterministic SWA with $f(\mathcal{R}, n) = \mathcal{O}(\log n)$ and failure ratio ϕ.*
4. *If $L \notin \langle \mathbf{LI}, \mathbf{PF}, \mathbf{Len} \rangle$, then there exists a failure ratio $0 < \phi \leq 1$ such that $f(\mathcal{R}, n) \notin o(n)$ for every deterministic SWA $\mathcal{R}$ for L with failure ratio ϕ.*

Note that Theorem 1.3(4) is an immediate corollary of Theorem 1.2(2), and that Theorem 1.3(3) follows from Theorem 1.3(1) and Theorem 1.1(5). Figure 1 summarizes the resulting space classes for each setting; the left column shows the three classes for the deterministic setting [7, 8]. Figure 2 shows an inclusion diagram for the language classes in Figure 1. One can show that the example languages in Figure 2 witness the strictness of the inclusions.

Technical contributions. Our randomized algorithms are based on a simulation of a counter by a Bernoulli random variable. Albeit simple, this *Bernoulli algorithm* can be utilized for sliding window algorithms with a bounded failure ratio. We present automata-theoretic descriptions of several Boolean closed language classes, to derive certain witness words for proving the lower bounds. Concerning lower bounds for sliding window algorithms with a bounded failure ratio, we consider a promise variant of the communication problem IDX_n, which has randomized one-way communication complexity $\Omega(n)$. Also we prove that a DFA which can count up to n, allowing a bounded failure ratio, needs $\Omega(n)$ states.

Related work. Let us emphasize related results which prove bounds on randomized sliding window algorithms. In the seminal paper of Datar et al. [6], where the sliding window model was introduced, the authors prove that the number of 1's in a 0/1-sliding window of

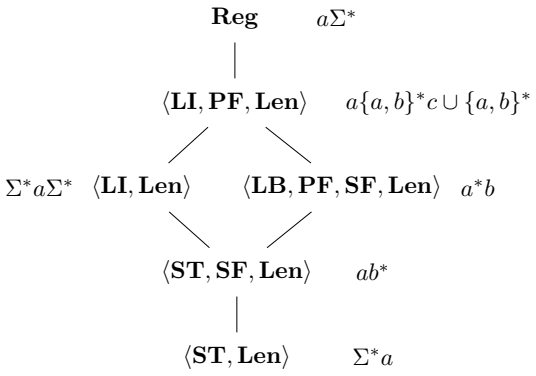


Figure 2 Examples for all occurring language classes where $\Sigma = \{a, b, c\}$.

size n can be maintained in space $\mathcal{O}(\frac{1}{\epsilon} \cdot \log^2 n)$ if one allows a multiplicative error of $1 \pm \epsilon$. Furthermore, they proved a matching lower bound for both deterministic and randomized algorithms. Ben-Basat et al. [3] present a sliding window algorithm for the related problem of approximating counts with an additive error, and present a deterministic and randomized lower space bound. We remark that both papers [3, 6] use a stronger notion of correctness: A randomized sliding window algorithm must have the property that for every input w, the probability that the algorithm produces some incorrect output while reading w is bounded by $1/3$. The authors utilize Yao's minimax principle to lift the deterministic lower bound to a randomized lower bound [13, Chapter 2]. Arasu et al. [2] study the problem of maintaining approximate frequency counts and quantiles over sliding windows. They present a deterministic algorithm and a randomized algorithm with improved space complexity using a simple sampling technique. Chan et al. [5] present a randomized sliding window algorithm for a certain measure of monotonicity. Furthermore, they present randomized lower bounds using communication complexity. The randomized algorithm analyzed in both papers comply with our standard definition of correctness with failure ratio 0. Further references can be found in [1, 4].

Finally let us mention the study of the communication complexity of regular languages by Tesson and Thérien [17], where a trichotomy (resp., quatrochotomy) for the deterministic (resp., randomized) communication complexity was shown. These results resemble our results, but the language classes that appear in [17] are different from the classes in our results.

2 Preliminaries

For integers $i, j \in \mathbb{N}$ let $[i, j] = \{k \in \mathbb{N} \colon i \leq k \leq j\}$. The set of all words over a finite alphabet Σ is denoted by Σ^*. The empty word is denoted by ε whereas error probabilities are denoted by the lunate epsilon ϵ. The sets of words over Σ of length exactly, at most and at least n are denoted by Σ^n, $\Sigma^{\leq n}$ and $\Sigma^{\geq n}$, respectively. Consider a word $w = a_1 a_2 \cdots a_m$. The *reversal* of w is defined as $w^{\mathsf{R}} = a_m \cdots a_2 a_1$, and for a language L we set $L^{\mathsf{R}} = \{w^{\mathsf{R}} \colon w \in L\}$. For a non-empty interval $[i, j] \subseteq [1, m]$ we define $w[i, j] = a_i a_{i+1} \cdots a_j$. If $i > j$ we set $w[i, j] = \varepsilon$. A *prefix* of w is a word of the form $w[1, i]$ for some $0 \leq i \leq m$; a *suffix* of w is a word of the form $w[i, m]$ for some $1 \leq i \leq m + 1$. A language $L \subseteq \Sigma^*$ is *prefix-free* (resp., *suffix-free*) if there are no two words $x, y \in L$ with $x \neq y$ and x is a prefix (resp., suffix) of y. A language is *bifix-free* if it is both prefix- and suffix-free.

Automata and regular languages. For general background in automata theory see [10]. A *deterministic finite automaton* (DFA) $A = (Q, \Sigma, q_0, \delta, F)$ consists of a finite set of states Q, a finite alphabet Σ, an initial state $q_0 \in Q$, a transition function $\delta\colon Q \times \Sigma \to Q$ and a set of final states $F \subseteq Q$. We inductively extend δ to a function $\delta\colon Q \times \Sigma^* \to Q$ as usual. If $P \subseteq Q$ is a set of states, then $L(A, P) = \{w \in \Sigma^*\colon \delta(q_0, w) \in P\}$. The language accepted by A is $L(A) = L(A, F)$. A language is *regular* if it is accepted by a DFA. Classes of languages are denoted by boldfaced letters. In this paper we will deal with the following language classes:

- **Reg**: the class of all *regular languages*.
- **Len**: the class of *regular length languages*, i.e., regular languages L such that for all $n \in \mathbb{N}$ we have $\Sigma^n \subseteq L$ or $\Sigma^n \cap L = \emptyset$.
- **LI** (resp., **RI**): the class of *regular left (resp., right) ideals*, i.e., languages of the form $\Sigma^* L$ (resp., $L\Sigma^*$) where L is regular.
- **ST** (resp., **PT**): the class of *suffix testable (resp., prefix testable) languages*, i.e., finite Boolean combinations of languages $\Sigma^* w$ (resp., $w\Sigma^*$) where $w \in \Sigma^*$.
- **SF** (resp., **PF**): the class of *regular suffix-free (resp., prefix-free) languages*
- **LB** (resp., **RB**): the class of *left ideals (resp., right ideals) generated by regular bifix-free languages*, i.e., languages of the form $\Sigma^* L$ (resp., $L\Sigma^*$) where $L \in \mathbf{PF} \cap \mathbf{SF}$.[1]

It is easy to see that every finite language is prefix and suffix testable. Moreover, prefix testable and suffix testable languages are regular. If $\mathbf{A}_1, \ldots, \mathbf{A}_n$ are language classes, then $\langle \mathbf{A}_1, \ldots, \mathbf{A}_n \rangle$ denotes the smallest boolean-closed class which contains $\bigcup_{i=1}^n \mathbf{A}_i$.

Probabilistic automata. In the following we introduce probabilistic automata [14, 15] as a model for randomized streaming algorithms. A *probabilistic automaton* $R = (Q, \Sigma, \iota, \rho, F)$ consists of a (possibly infinite) set of states Q, an alphabet Σ, an initial state distribution $\iota\colon Q \to \{r \in \mathbb{R}\colon 0 \leq r \leq 1\}$, a transition probability function $\rho\colon Q \times \Sigma \times Q \to \{r \in \mathbb{R}\colon 0 \leq r \leq 1\}$ and set of final states $F \subseteq Q$ such that $\sum_{q \in Q} \iota(q) = 1$ and $\sum_{q \in Q} \rho(p, a, q) = 1$ for all $p \in Q$, $a \in \Sigma$. If ι and ρ map into $\{0, 1\}$, then R is a *deterministic automaton*. A *run* on a word $a_1 \cdots a_m \in \Sigma^*$ in R is a sequence $\pi = (q_0, a_1, q_1, a_2, \ldots, a_m, q_m)$ where $q_0, \ldots, q_m \in Q$ and $\rho(q_{i-1}, a_i, q_i) > 0$ for all $1 \leq i \leq m$. Given such a run π in R we define $\rho_\iota(\pi) = \iota(q_0) \cdot \prod_{i=1}^n \rho(q_{i-1}, a_i, q_i)$. For each $w \in \Sigma^*$ the function ρ_ι is a probability distribution on the set $\mathrm{Runs}(w)$ of all runs of $\mathcal{R}$ on w.

3 Randomized streaming and sliding window algorithms

A *randomized streaming algorithm* (R, enc) consists of a probabilistic automaton $R = (Q, \Sigma, \iota, \rho, F)$ as above and an injective function $\mathrm{enc}\colon Q \to \{0, 1\}^*$. Usually, we will only refer to the underlying automaton R. If R is deterministic, we speak of a *deterministic streaming algorithm*. The maximum number of bits stored during a run $\pi = (q_0, a_1, \ldots, a_m, q_m)$ is denoted by $\mathrm{space}(R, \pi)$, i.e., $\mathrm{space}(R, \pi) = \max\{|\mathrm{enc}(q_i)| : 0 \leq i \leq m\}$. The worst case space complexity of R on w is $\mathrm{space}(R, w) = \max\{\mathrm{space}(R, \pi) : \pi \in \mathrm{Runs}(w), \rho_\iota(\pi) > 0\}$. A run $\pi = (q_0, a_1, \ldots, a_m, q_m)$ is *correct* with respect to a language $K \subseteq \Sigma^*$ if $q_m \in F \Leftrightarrow a_1 \cdots a_m \in K$ holds. The *error probability* of R on w for K is

$$\epsilon(R, w, K) = \sum \{\rho_\iota(\pi) : \pi \in \mathrm{Runs}(w) \text{ is not correct with respect to } K\}.$$

[1] or equivalently, the class of all left ideals generated by regular prefix-free languages. Since our proofs related to **LB** yield decompositions of the form $\Sigma^* L$ with L bifix-free, we decided to define **LB** as above.

Given an error bound $0 \leq \epsilon \leq 1$, the *failure ratio* of R on w is defined as

$$\phi(R, w, K, \epsilon) = \frac{1}{m+1} |\{t \in [0, m] : \epsilon(R, a_1 a_2 \cdots a_t, K) > \epsilon\}|.$$

For a window length $n \geq 0$ and a stream $x \in \Sigma^*$ we define $\text{last}_n(x)$ to be the suffix of $\square^n x$ of length n where $\square \in \Sigma$ is a fixed alphabet symbol. The word $\text{last}_n(\varepsilon) = \square^n$ is the initial window content. Given a language $L \subseteq \Sigma^*$ and a window size $n \geq 0$ we define the language

$$L_n = \{x \in \Sigma^* \colon \text{last}_n(x) \in L\}. \tag{1}$$

A *randomized sliding window algorithm* (*randomized SWA* for short) is a sequence $\mathcal{R} = (R_n)_{n \geq 0}$ of randomized streaming algorithms R_n over the same alphabet Σ. If every R_n is deterministic, we speak of a *deterministic SWA*. The *space complexity* of the randomized SWA $\mathcal{R} = (R_n)_{n \geq 0}$ is the function $f(\mathcal{R}, n) = \sup\{\text{space}(R_n, w) : w \in \Sigma^*\} \in \mathbb{N} \cup \{\infty\}$. Clearly, if R_n is finite, then one can always find a state encoding such that $f(\mathcal{R}, n) = \lfloor \log_2 |R_n| \rfloor$.

▶ **Definition 3.1.** Let $0 \leq \epsilon \leq 1$ and $0 \leq \phi \leq 1$. A randomized SWA $\mathcal{R} = (R_n)_{n \geq 0}$ is (ϵ, ϕ)-*correct* for a language $L \subseteq \Sigma^*$ if $\phi(R_n, w, L_n, \epsilon) \leq \phi$ for all $n \geq 0$ and $w \in \Sigma^{\geq n}$. The number ϵ is the *error probability* and ϕ is the *failure ratio* of $\mathcal{R}$ (with respect to ϵ).

Definition 3.1 also makes sense in the special case that $\mathcal{R}$ is deterministic and $\epsilon = 0$. A $(0, \phi)$-correct deterministic SWA $\mathcal{R} = (R_n)_{n \geq 0}$ for L has the property that R_n produces at most $\phi \cdot (m+1)$ many incorrect answers when running on any input word of length $m \geq n$.

We will set the error probability to $\epsilon = 1/3$, which is justified by the following lemma (that follows from a standard Chernoff bound).

▶ **Lemma 3.2.** *Let $L \subseteq \Sigma^*$, $0 < \epsilon' < \epsilon < \frac{1}{2}$ and $0 \leq \phi \leq 1$. Given a randomized SWA $\mathcal{R}$ which is (ϵ, ϕ)-correct for L, one can construct a randomized SWA $\mathcal{R}'$ which is (ϵ', ϕ)-correct for L such that $f(\mathcal{R}', n) \leq \ln(\frac{1}{\epsilon'}) \cdot \frac{1}{\text{poly}(\epsilon)} \cdot f(\mathcal{R}, n)$.*

▶ **Definition 3.3.** Let $L \subseteq \Sigma^*$ be a language and $0 \leq \phi \leq 1$.

- A *randomized SWA for L* with failure ratio ϕ is a randomized SWA which is $(1/3, \phi)$-correct for L. If moreover $\phi = 0$, then we speak of a *randomized SWA for L*.
- A *deterministic SWA for L* with failure ratio ϕ is a deterministic SWA which is $(0, \phi)$-correct for L. If moreover $\phi = 0$, then we speak of a *deterministic SWA for L*.

We only consider randomized SWAs $\mathcal{R} = (R_n)_{n \geq 0}$ where every R_n has a finite state set Q_n. This is justified by the fact that for every language L and every n the language L_n from (1) is regular and hence can be accepted by a DFA. The space-optimal deterministic SWA for a language L therefore consists of the minimal DFA for L_n for every $n \geq 0$. For a fixed error probability $\epsilon < 1/2$ a space-optimal randomized SWA for L consists of a minimal probabilistic finite automaton for L_n which accepts a word w with probability at least $1 - \epsilon$ if $w \in L_n$ and accepts w with probability at most ϵ if $w \notin L_n$. By a result of Rabin [15] such a probabilistic finite automaton can be transformed into an equivalent DFA with an exponential blow-up. Hence, we get:

▶ **Lemma 3.4.** *Let $\mathcal{R}$ be a randomized SWA for the language L. Then, there exists a deterministic SWA $\mathcal{D}$ for L such that $f(\mathcal{D}, n) \in \mathcal{O}(2^{f(\mathcal{R}, n)})$.*

4 Upper bounds

In this section we prove the upper bounds in Theorem 1.1, 1.2 and 1.3. We use the simple fact that space complexity classes in the sliding window model are Boolean-closed:

▶ **Lemma 4.1.** *Let L be a Boolean combination of languages $L_1, \dots, L_k$. For each $i \in [1,k]$ let $\mathcal{R}_i$ be a randomized SWA for L_i with failure ratio ϕ_i. Then L has a SWA $\mathcal{R}$ with failure ratio $\sum_{i=1}^k \phi_i$ and $f(\mathcal{R},n) = \mathcal{O}(\sum_{i=1}^k f(\mathcal{R}_i,n))$.*

4.1 The Bernoulli algorithm

In this section, we introduce a randomized SWA that will be used for the proof of the upper bounds (3) from Theorem 1.1 and (1) from Theorem 1.2. The idea is based on the algorithm from [7], which stores *path summaries* in the reversal DFA. Consider a regular language $L \subseteq \Sigma^*$ and let $A = (Q, \Sigma, q_0, \delta, F)$ be a DFA for L^{R}. Observe that if L is a left-ideal then any run in A switches from $Q \setminus F$ to F at most once; if L is suffix-free then any run in A visits F at most once. For a stream $w \in \Sigma^*$ define the function $\ell_w \colon Q \to \mathbb{N} \cup \{\infty\}$ by

$$\ell_w(q) = \inf\{k \in \mathbb{N} : \delta(q, \mathrm{last}_k(w)^{\mathsf{R}}) \in F\}, \tag{2}$$

where we set $\inf(\emptyset) = \infty$. By the observations above we know:

- If L is a left ideal, then $\mathrm{last}_n(w) \in L$ if and only if $\ell_w(q_0) \leq n$.
- If L is suffix-free, then $\mathrm{last}_n(w) \in L$ if and only if $\ell_w(q_0) = n$.

One can define a deterministic SWA which stores the function ℓ_w on input stream w. If a symbol $a \in \Sigma$ is read, we can determine ℓ_{wa} from ℓ_w: If $q \in F$, then $\ell_{wa}(q) = 0$. Otherwise $\ell_{wa}(q) = 1 + \ell_w(\delta(q,a))$ where $1 + \infty = \infty$.

Using a Bernoulli random variable, we define a randomized approximation of the above deterministic SWA to reduce the space complexity to $\mathcal{O}(1)$. Let $\beta \colon \mathbb{N} \to \mathbb{R}$ be a function such that for some n_0, $0 \leq \beta(n) \leq 1$ for all $n \geq n_0$, which controls the Bernoulli random variable and will later be instantiated by concrete functions. We define the following constant-space randomized SWA $\mathcal{B} = (B_n)_{n \geq 0}$ (which depends on the language L, the DFA A and the function β), which we call the Bernoulli algorithm. If $n < n_0$ let B_n be the trivial deterministic streaming algorithm for L_n. For $n \geq n_0$ the algorithm B_n stores a Boolean flag for each state in form of a function $b \colon Q \to \{0,1\}$. All flags $b(q)$ for $q \in F$ are fixed to 1 forever. For all other states $q \in Q \setminus F$ we define the initial value of the flag $b(q)$ as follows, where $\ell = \ell_\varepsilon(q) \in \mathbb{N} \cup \{\infty\}$:

$$b(q) := \begin{cases} 0 \text{ with probability } 1 - (1-\beta(n))^{\ell} \\ 1 \text{ with probability } (1-\beta(n))^{\ell}. \end{cases} \tag{3}$$

Here we set $x^\infty = 0$ for $0 \leq x < 1$. For all states $q \in Q \setminus F$ we do the following upon arrival of a symbol $a \in \Sigma$:

$$b(q) := \begin{cases} 0 & \text{with probability } \beta(n) \\ b(\delta(q,a)) & \text{with probability } 1 - \beta(n) \end{cases} \tag{4}$$

The algorithm accepts if and only if $b(q_0) = 1$. An induction on $|w|$ shows:

▶ **Lemma 4.2.** *For all $n \geq n_0$ and $w \in \Sigma^*$ we have $\Pr[B_n \text{ accepts } w] = (1-\beta(n))^{\ell_w(q_0)}$.*

4.2 Randomized SWAs with failure ratio zero

In this section we present a $\mathcal{O}(\log\log n)$ space algorithm for suffix-free regular languages L. Since languages in **ST** and **Len** have constant space deterministic SWAs (Theorem 1.1(1)) and the space complexity classes are closed under Boolean combinations by Lemma 4.1, this implies Theorem 1.1(3).

Let $L \in \mathbf{SF}$ and $A = (Q, \Sigma, q_0, \delta, F)$ be a DFA for $L^{\mathsf{R}} \in \mathbf{PF}$. Since the case $L = \emptyset$ is trivial, we can assume that A contains at least one final state which is reachable from q_0. Furthermore, since L^{R} is prefix-free, any run in A from q_0 contains at most one final state. Therefore, we can assume that F contains exactly one final state q_F, and all outgoing transitions from q_F lead to a sink state.

Recall the function $\ell_w \colon Q \to \mathbb{N} \cup \{\infty\}$ defined in (2). Notice that $\mathrm{last}_n(w) \in L$ if and only if $\ell_w(q_0) = n$ for all $w \in \Sigma^*$. Our randomized SWA $\mathcal{R} = (R_n)_{n \geq 0}$ consists of two parts: Firstly, we take the Bernoulli algorithm $\mathcal{B} = (B_n)_{n \geq 0}$ from Section 4.1 for the function $\beta(n) = 1/(2n)$ and $n_0 = 1$. From Lemma 4.2 we know that B_n accepts a word $w \in \Sigma^*$ with probability $(1 - 1/(2n))^{\ell_w(q_0)}$. Secondly, we simultaneously run a modulo-counting algorithm M_n. Let p_i be the i-th prime number and let $s(m)$ be the product of all prime numbers $\leq m$. It is known that $\ln(s(m)) > m \cdot (1 - 1/\ln m)$ for $m \geq 41$ [16, 3.16] and $p_i < i \cdot (\ln i + \ln \ln i)$ for $i \geq 6$ [16, 3.13]. Let k be the first natural number such that $\prod_{i=1}^{k} p_i \geq n$. By the above bounds we get $k \in \mathcal{O}(\log n)$ and $p_{3k} \in \mathcal{O}(\log n \cdot \log \log n)$. The algorithm M_n initially picks a random prime $p \in \{p_1, \ldots, p_{3k}\}$, which is stored throughout the run using $\mathcal{O}(\log \log n)$ bits. Then, after reading $w \in \Sigma^*$, M_n stores for every $q \in Q$ a bit telling whether $\ell_w(q) < \infty$ and, if the latter holds, the values $\ell_w(q) \bmod p$ using $\mathcal{O}(|Q| \cdot \log \log n)$ bits. The algorithm accepts if and only if $\ell_w(q_0) \equiv n \bmod p$.

The combined algorithm R_n accepts if and only if both B_n and T_n accept. Let us bound the error probability on an input stream $w \in \Sigma^*$ with $\ell = \ell_w(q_0)$.

Case 1. $\ell = n$, i.e., $\mathrm{last}_n(w) \in L$. Then M_n accepts w with probability 1. Moreover, B_n accepts w with probability $(1 - 1/(2n))^n \geq 0.6$ for $n \geq 12$ (note that $(1 - 1/(2n))^n$ converges to $1/\sqrt{e} \approx 0.60653$ from below). Hence, R_n accepts with probability at least 0.6.

Case 2. $\ell \geq 2n$ and hence $\mathrm{last}_n(w) \notin L$. Then B_n rejects with probability $1 - (1 - 1/(2n))^{\ell} \geq 1 - (1 - (1/2n))^{2n} \geq 1 - 1/e \geq 0.6$. Here, we use the well-known inequality $(1 - 1/y)^y \leq e^{-1}$ for all $y \geq 1$. Hence, R_n also rejects with probability at least 0.6.

Case 3. $\ell < 2n$ and $\ell \neq n$, and thus $\mathrm{last}_n(w) \notin L$. Since $\ell - n \in [-n, n]$ and any product of at least $k + 1$ pairwise distinct primes exceeds n, the number $\ell - n \neq 0$ has at most k prime factors. Therefore, M_n (and thus R_n) rejects with probability at least $2/3$.

4.3 SWAs with arbitrarily small non-zero failure ratio

In this section we sketch the proofs of Theorem 1.2(1) and Theorem 1.3(1). We focus on the main base case $L \in \mathbf{LI}$ from Theorem 1.2(1).

Let $L \subseteq \Sigma^*$ be a regular left ideal. Let $A = (Q, \Sigma, q_0, \delta, F)$ be the minimal DFA for L^{R}. Since the case $L = \emptyset$ is trivial, we can assume that $L \neq \emptyset$. It is easy to see that F contains a single state q_F from which all outgoing transitions lead back to q_F. Recall the function $\ell_w \colon Q \to \mathbb{N} \cup \{\infty\}$ defined in (2). Since L is a left ideal, we have: $\mathrm{last}_n(w) \in L$ if and only if $\ell_w(q_0) \leq n$: The following lemma says that the portion of prefixes of an input stream, where $\ell_w(q_0)$ is close to n, is small:

▶ **Lemma 4.3.** *Let $0 < \xi < 1$. Let $n \geq 0$ be a window size and $w \in \Sigma^{\geq n}$ be an input stream. Then the number of prefixes v of w such that $\lceil \xi n \rceil \leq \ell_v(q_0) \leq n$ is at most*

$$\frac{(1 - \xi + \frac{1}{n}) \cdot |Q|}{\xi} \cdot (|w| + 1 + \xi n). \tag{5}$$

Proof. Let us say that a prefix v of w is a *hit*, if $\lceil \xi n \rceil \leq \ell_v(q_0) \leq n$. Consider an interval $I = [i, i']$ with $0 \leq i \leq i' \leq |w|$ and $i' - i \leq \lceil \xi n \rceil$. With each position $j \in I$ we associate the prefix $w[1, j]$. We claim that $V := \{w[1, j] \colon j \in I\}$ contains at most $|Q| \cdot (n - \lceil \xi n \rceil + 1)$ hits. Let us assume the contrary. Since $\lceil \xi n \rceil \leq \ell_v(q_0) \leq n$ for every hit v, and the interval $[\lceil \xi n \rceil, n]$ contains $n - \lceil \xi n \rceil + 1$ many different values, there is a subset $U \subseteq V$ and some $\ell \in [\lceil \xi n \rceil, n]$ such that (i) $|U| > |Q|$ and (ii) $\ell_v(q_0) = \ell$ for all $v \in U$. Let $U = \{w[1, j_1], w[1, j_2], \ldots, w[1, j_k]\}$, where $k > |Q|$. Consider the words $u_1 = \mathrm{last}_\ell(w[1, j_1]), u_2 = \mathrm{last}_\ell(w[1, j_2]), \ldots, u_k = \mathrm{last}_\ell(w[1, j_k])$. Since $j_k - j_1 \leq \lceil \xi n \rceil$ and $\ell \geq \lceil \xi n \rceil$, we have $\ell - j_k + j_1 \geq 0$. Hence, we can consider the words $v_1 = \mathrm{last}_{\ell - j_k + j_1}(w[1, j_1]), v_2 = \mathrm{last}_{\ell - j_k + j_2}(w[1, j_2]), \ldots, v_k = \mathrm{last}_\ell(w[1, j_k])$. Clearly, v_j is a suffix of u_j. Moreover, the words v_j all start in the same position of $\square^n w$, i.e., every v_j is a prefix of $v_{j'}$ for $j \leq j'$. Consider now the state $q_j = \xi(q_0, v_j^{\mathsf{R}})$ for $1 \leq j \leq k$. Since $k > |Q|$ there exist $j < j'$ such that $q_j = q_{j'}$. But this would imply that $\ell_{v_j}(q_0) < \ell_{v_{j'}}(q_0)$, which contradicts $\ell_{v_j}(q_0) = \ell = \ell_{v_{j'}}(q_0)$. This shows the above claim.

Now we can finish the proof of the lemma: We divide the interval $[0, |w|]$ into intervals of size $\lceil \xi n \rceil + 1$ and one last interval of possibly shorter length. This yields $\lceil (|w|+1)/(\lceil \xi n \rceil + 1) \rceil$ many intervals. In each of these intervals we find at most $|Q| \cdot (n - \lceil \xi n \rceil + 1)$ many hits by the above claim. Hence, the total number of hits is bounded by

$$\begin{aligned} \left\lceil \frac{|w|+1}{\lceil \xi n \rceil + 1} \right\rceil \cdot |Q| \cdot (n - \lceil \xi n \rceil + 1) &\leq \left(\frac{|w|+1}{\xi n} + 1 \right) \cdot |Q| \cdot (n - \xi n + 1) \\ &= (|w| + 1 + \xi n) \cdot |Q| \cdot \frac{1 - \xi + \frac{1}{n}}{\xi}. \end{aligned}$$

This concludes the proof of the lemma. ◀

We now consider the Bernoulli algorithm $\mathcal{B}^\epsilon = (B_n^\epsilon)_{n \geq 0}$ for $\beta_\epsilon \colon \mathbb{N} \to \mathbb{R}$ with $\beta_\epsilon(n) = \ln(1/\epsilon)/n$. Note that $0 < \beta_\epsilon(n) \leq 1$ for $n \geq \ln(1/\epsilon)$. With Lemma 4.2 one can show:

▶ **Lemma 4.4.** *For every $0 < \xi < 1$ there exists $0 < \epsilon < 1/2$ and $n_0 \geq 1$ such that for all $n \geq n_0$ the following holds: If $w \in \Sigma^*$ and $\ell_w(q_0) \notin [\lceil \xi n \rceil, n]$, then $\epsilon(B_n^\epsilon, w, L_n) \leq \epsilon$.*

▶ **Theorem 4.5.** *Let L be a regular left ideal and $0 < \phi < 1$. Then L has a randomized SWA $\mathcal{R}$ with $f(\mathcal{R}, n) = \mathcal{O}(1)$ and failure ratio ϕ.*

Proof. Let us fix a failure ratio $0 < \phi < 1$ and let $0 < \xi < 1$, which will be defined later (depending on ϕ). Let ϵ and n_0 be the numbers from Lemma 4.4. Let $\mathcal{B}^\epsilon = (B_n^\epsilon)_{n \geq 0}$ be the randomized SWA described above. Let $n \geq n_0$ be a window size and $w \in \Sigma^{\geq n}$ be an input stream. Consider the set $P(w)$ of all prefixes v of w such that $\ell_v(q_0) \in [\lceil \xi n \rceil, n]$. By Lemma 4.4 the algorithm B_n^ϵ errs on each prefix $v \notin P(w)$ with probability at most ϵ, i.e.,

$$\begin{aligned} \phi(B_n^\epsilon, w, L_n, \epsilon) &\leq \frac{|P(w)|}{|w|+1} \overset{\text{Lemma 4.3}}{\leq} \frac{(1 - \xi + \frac{1}{n}) \cdot |Q|}{\xi} \cdot \left(1 + \frac{\xi n}{|w|+1} \right) \\ &\leq \left(1 - \xi + \frac{1}{n} \right) \cdot |Q| \cdot \left(1 + \frac{1}{\xi} \right). \end{aligned} \tag{6}$$

Note that if ξ converges to 1, then the probability (6) tends towards $2|Q|/n$. Hence we can choose numbers $n_1 \geq n_0$ and $0 < \xi < 1$ such that for all $n \geq n_1$ the probability (6) is smaller than our fixed failure ratio ϕ.

Finally for window sizes $n < n_1$ we can use the optimal deterministic sliding-window algorithms for L and window size n. The space complexity of the resulting algorithm is a constant that depends only on ϕ. ◀

The base case $L \in \mathbf{Len}$ in Theorem 1.2(1) is trivial (there is a constant-space deterministic SWA). Finally, for the case $L \in \mathbf{PF}$, one can show that the constant-space deterministic SWA that always rejects has a failure ratio of $\mathcal{O}(1/n)$ for window length n. This fact also covers the base case $L \in \mathbf{PF}$ from Theorem 1.3(1), and a similar argument covers the case $L \in \mathbf{SF}$. For the remaining base case $L \in \mathbf{LB}$ in Theorem 1.3(1), one can basically use a DFA for L itself as a sliding window algorithm. One can prove that this algorithm gives only $\mathcal{O}(1)$ incorrect answers in n consecutive windows.

5 Lower bounds

To prove the claimed lower bounds, we apply the same proof strategy in all cases (with one exception). We first show that if a regular language does not belong to the language class under consideration then there exist certain witness words. These words can then be used to apply known lower bounds from randomized one-way communication complexity [11, 12] by deriving a randomized communication protocol from a randomized SWA. This is a standard technique for showing lower bounds for streaming algorithms.

The setting in (one-way) communication complexity is as follows: Alice holds an element $x \in X$ and Bob holds an element $y \in Y$ and they aim to compute $f(x, y)$ according to a randomized one-way protocol P. Alice computes from her input $x \in X$ and from a random string a message and sends it to Bob. Using this message, his input $y \in Y$ and a random string, Bob computes an output bit. The cost of the protocol P is the maximal number of bits sent from Alice to Bob. We say that P computes f if the protocol computes $f(x, y)$ with probability at least $2/3$ for all inputs (x, y). The minimal cost of a protocol which computes f is denoted by $C(f)$.

Due to space constraints we only prove Theorem 1.1(4) in detail and discuss the proofs of the remaining lower bounds only briefly in Section 5.2.

5.1 Proof of point 4 from Theorem 1.1

Let $A = (Q, \Sigma, q_0, \delta, F)$ be a DFA. A state q is *trivial* if $\delta(q, x) \neq q$ for all $x \in \Sigma^+$, otherwise it is *non-trivial*. A pair $(p, q) \in Q \times Q$ of states is called *synchronized* if there exist words $x, y, z \in \Sigma^*$ with $|x| = |y| = |z| \geq 1$ such that $\delta(p, x) = p$, $\delta(p, y) = q$ and $\delta(q, z) = q$. A pair (p, q) is called *reachable* from a state r if p is reachable from r. A state pair (p, q) is called *F-consistent* if either $\{p, q\} \cap F = \emptyset$ or $\{p, q\} \subseteq F$. We remark that synchronized state pairs have no connection to the notion of synchronizing words. A simple pumping argument shows:

▶ **Lemma 5.1.** *A state pair (p, q) is synchronized if and only if p and q are non-trivial and there exists $y \in \Sigma^+$ such that $|Q|!$ divides $|y|$ and $\delta(p, y) = q$.*

Let $Q = T \cup N$ be the partition of the state set into the set T of trivial states and the set N of non-trivial states. A function $\beta\colon \mathbb{N} \to \{0, 1\}$ is *k-periodic* if $\beta(i) = \beta(i + k)$ for all $i \in \mathbb{N}$.

▶ **Lemma 5.2.** *Assume that every synchronized pair in A which is reachable from q_0 is F-consistent. Then for every word $v \in \Sigma^*$ of length at least $|Q|! \cdot (|T| + 1)$ there exists a $|Q|!$-periodic function $\beta_v\colon \mathbb{N} \to \{0, 1\}$ such that the following holds: If $w \in v\Sigma^*$ and $\delta(q_0, w) \in N$, then we have $w \in L$ iff $\beta(|w|) = 1$.*

Proof. Let $v = a_1 a_2 \cdots a_k$ with $k \geq |Q|! \cdot (|T| + 1)$, and consider the run $q_0 \xrightarrow{a_1} \cdots \xrightarrow{a_k} q_k$ of A on v. Clearly, each trivial state can occur at most once in the run. First notice that for each $0 \leq i \leq |Q|! - 1$ at least one of the states in $Q_i = \{q_{i+j|Q|!} : 0 \leq j \leq |T|\}$ is non-trivial because otherwise the set would contain $|T| + 1$ pairwise distinct trivial states.

Furthermore, we claim that the non-trivial states in Q_i are either all final or all non-final: Take two non-trivial states $q_{i+j_1|Q|!}$ and $q_{i+j_2|Q|!}$ with $j_1 < j_2$. Since we have a run of length $(j_2 - j_1)|Q|!$ from $q_{i+j_1|Q|!}$ to $q_{i+j_2|Q|!}$, the states form a synchronized pair by Lemma 5.1. Hence, by assumption the two states are F-consistent. Now define $\beta_v \colon \mathbb{N} \to \{0,1\}$ by

$$\beta_v(m) = \begin{cases} 1 & \text{if the states in } Q_{m \bmod |Q|!} \cap N \text{ are final,} \\ 0 & \text{if the states in } Q_{m \bmod |Q|!} \cap N \text{ are non-final,} \end{cases}$$

which is well-defined by the remarks above. Clearly β_v is $|Q|!$-periodic.

Let $w = a_1 \cdots a_m \in v\Sigma^*$ be a word of length $m \geq k$. The run of A on w prolongs the run on v, say $q_0 \xrightarrow{a_1} \cdots \xrightarrow{a_k} q_k \xrightarrow{a_{k+1}} \cdots \xrightarrow{a_m} q_m$. Assume that $q_m \in N$. As argued above, there is a position $0 \leq i \leq k$ such that $i \equiv m \pmod{|Q|!}$ and $q_i \in N$. Hence (q_i, q_m) is a synchronized pair by Lemma 5.1 which is F-consistent by assumption. Therefore $w \in L$ iff $q_m \in F$ iff $q_i \in F$ iff $\beta_v(|w|) = 1$. ◀

▶ **Lemma 5.3.** *Assume that every synchronized pair in A which is reachable from q_0 is F-consistent. Then $L(A)$ belongs to $\langle \mathbf{PT}, \mathbf{PF}, \mathbf{Len} \rangle$.*

Proof. Let $F_N = N \cap F$ and $F_T = T \cap F$. We decompose L into

$$L = L(A, F_N) \cup \bigcup_{q \in F_T} L(A, \{q\}).$$

First observe that $L(A, \{q\}) \in \mathbf{PF}$ for all $q \in F_T$ because a trivial state q can occur at most once in a run of A. It remains to show that $L(A, F_N)$ belongs to $\langle \mathbf{PT}, \mathbf{PF}, \mathbf{Len} \rangle$. Using the threshold $k = |Q|! \cdot (|T| + 1)$, we distinguish between words of length at most $k-1$ and words of length at least k, and group the latter set by their prefix of length k, i.e.,

$$L(A, F_N) = (L(A, F_N) \cap \Sigma^{\leq k-1}) \cup \bigcup_{v \in \Sigma^k} (L(A, F_N) \cap v\Sigma^*).$$

The first part $L(A, F_N) \cap \Sigma^{\leq k-1}$ is finite and thus prefix testable. To finish the proof, we will show that $L(A, F_N) \cap v\Sigma^* \in \langle \mathbf{PT}, \mathbf{PF}, \mathbf{Len} \rangle$ for each $v \in \Sigma^k$. Let $v \in \Sigma^k$ and let $\beta_v \colon \mathbb{N} \to \{0,1\}$ be the $|Q|!$-periodic function from Lemma 5.2. We know

$$L(A, F_N) \cap v\Sigma^* = (v\Sigma^* \cap \{w \in \Sigma^* : \beta(|w|) = 1\}) \setminus L(A, T).$$

Note that $\{w \in \Sigma^* : \beta(|w|) = 1\} \in \mathbf{Len}$, $v\Sigma^* \in \mathbf{PT}$ and $L(A, T) \in \langle \mathbf{PF} \rangle$. ◀

By applying Lemma 5.3 to the language L^{R}, we obtain:

▶ **Lemma 5.4.** *If $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$, then there exist $u, x, y, z \in \Sigma^*$ with $|x| = |y| = |z| \geq 1$ such that one of the following cases holds:*

- $x^*u \subseteq L$ *and* $z^*yx^*u \cap L = \emptyset$
- $x^*u \cap L = \emptyset$ *and* $z^*yx^*u \subseteq L$.

We can now conclude the proof of Theorem 1.1(4). Let $L \in \mathbf{Reg} \setminus \langle \mathbf{ST}, \mathbf{SF}, \mathbf{Len} \rangle$. We reduce from the communication problem $\mathrm{GT}_m \colon [1,m]^2 \to \{0,1\}$ where $\mathrm{GT}_m(i,j) = 1$ iff $i > j$. It is known that $C(\mathrm{GT}_m) \in \Theta(\log m)$ [11, Theorem 3.8].

Consider the words $u, x, y, z \in \Sigma^*$ described in Lemma 5.4. Let $\mathcal{R} = (R_n)_{n \geq 0}$ be a randomized SWA for L. Let $m \geq 0$. We describe a randomized one-way protocol P_m for GT_m: Let $i \in [1,m]$ be the input of Alice and $j \in [1,m]$ be the input of Bob. Let $n = |x| \cdot m + |u| \in \Theta(m)$. Alice starts by running the probabilistic automaton R_n on

$z^m y x^{m-i}$ using her random bits in order to simulate the random choices of R_n. Afterwards, she sends the encoding of the reached state to Bob. Bob then continues the run of R_n from the transmitted state with the word $x^j u$. Hence, R_n is simulated on the word $w := z^m y x^{m-i} x^j u = z^m y x^{m-i+j} u$. We have

$$\text{last}_n(w) = \begin{cases} z^{i-1-j} y x^{m-i+j} u, & \text{if } i > j, \\ x^m u, & \text{if } i \leq j. \end{cases}$$

By Lemma 5.4, $\text{last}_n(w)$ belongs to L in exactly one of the two cases $i > j$ and $i \leq j$. Hence Bob can distinguish these two cases with probability at least $2/3$. It follows that the protocol computes GT_m and its cost is bounded by $f(\mathcal{R}, n)$. We have $f(\mathcal{R}, |x| \cdot m + |u|) = f(\mathcal{R}, n) \geq \text{cost}(P_m) \in \Omega(\log m)$ and therefore $f(\mathcal{R}, n) \notin o(\log n)$.

5.2 Remaining lower bounds

The remaining lower bounds in Theorem 1.1–1.3 are shown by reductions from communication problems as well, with one exception:

- For Theorem 1.1(2) we reduce from the communication problem EQ_m, where Alice holds a number $i \in [1, m]$, Bob holds a number $j \in [1, m]$ and Bob has to verify whether $i = j$. It is known that $C(\text{EQ}_m) \in \Theta(\log \log m)$ [12].
- For Theorem 1.1(6) we reduce from the communication problem IDX_m, where Alice holds a bitstring $a_1 \cdots a_m$, Bob holds an index $i \in [1, m]$ and Bob has to output the bit a_i. By [11, Theorem 3.7] we know that $C(\text{IDX}_m) \in \Theta(m)$.
- For Theorem 1.2(2) we reduce from a promise variant of IDX_m. Let $D \subseteq \{0,1\}^m \times [1, m]$ such that for each $x \in \{0,1\}^m$ there exist at least $7/8 \cdot m$ pairs $(x, i) \in D$. We prove that IDX_m still has communication complexity $\Omega(m)$ if the inputs for Alice and Bob are restricted to D. This extends [11, Theorem 3.7] and allows us to incorporate the notion of failure ratio into a communication protocol.
- Theorem 1.3(2) is not shown via a reduction to a communication problem. Instead, we utilize a combinatorial result on the ability of DFAs to count up to a threshold n, modulo a failure ratio ϕ.

6 Further results

The technical report [9] contains several further results that we briefly want to discuss.

In this paper, we only considered randomized SWAs with a two sided error (analogously to the complexity class BPP). Randomized SWAs with a one-sided error (analogously to the class RP) can be motivated by applications, where all "yes" outputs have to be correct, but a small probability for a false negative answer is acceptable. We prove that for every regular language the optimal space bound with respect to randomized SWAs with one-sided error coincides (up to constant factors) with the optimal space bound in the deterministic setting [7, 8] (which was discussed in the introduction).

In the introduction (related work) we remarked that some authors use a stronger correctness notion for randomized SWAs, where for every input w, the probability that the algorithm produces some incorrect output while reading w is bounded by $1/3$. We show that for every approximation problem (where approximation problems are defined by specifying for every input string a set of possible output values) every randomized SWA that fulfills this stronger notion of correctness can be completely derandomized without a space increase.

References

1 Charu C. Aggarwal. *Data Streams - Models and Algorithms*. Springer, 2007.

2 Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of PODS 2004*, pages 286–296. ACM, 2004.

3 Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Efficient summing over sliding windows. In *Proceedings of SWAT 2016*, volume 53 of *LIPIcs*, pages 11:1–11:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

4 Vladimir Braverman. Sliding window algorithms. In *Encyclopedia of Algorithms*, pages 2006–2011. Springer, 2016.

5 Ho-Leung Chan, Tak Wah Lam, Lap-Kei Lee, Jiangwei Pan, Hing-Fung Ting, and Qin Zhang. Edit distance to monotonicity in sliding windows. In *Proceedings of ISAAC 2011*, volume 7074 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2011.

6 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

7 Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey, and Konstantinos Mamouras. Automata theory on sliding windows. In *Proceedings of STACS 2018*, volume 96 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

8 Moses Ganardi, Danny Hucke, and Markus Lohrey. Querying regular languages over sliding windows. In *Proceedings of FSTTCS 2016*, volume 65 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

9 Moses Ganardi, Danny Hucke, and Markus Lohrey. Randomized sliding window algorithms for regular languages. Technical report, arXiv.org, 2018. URL: `https://arxiv.org/abs/1802.07600`.

10 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison–Wesley, Reading, MA, 1979.

11 Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999.

12 Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.

13 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

14 Azaria Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.

15 Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.

16 J. Barkley Rosser and Lowell Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6(1):64–94, 1962.

17 Pascal Tesson and Denis Thérien. Complete classifications for the communication complexity of regular languages. *Theory of Computing Systems*, 38(2):135–159, 2005. `doi:10.1007/s00224-004-1190-2`.

Aperiodic points in $\mathbb{Z}^2$-subshifts

Anael Grandjean[1]
Laboratoire d'Algorithmique, Complexité et Logique
Université Paris-Est Créteil, France
anael.grandjean@u-pec.fr

Benjamin Hellouin de Menibus
Laboratoire de Recherche en Informatique,
Université Paris-Sud, CNRS, CentraleSupélec, Université Paris-Saclay, France
hellouin@lri.fr
https://orcid.org/0000-0001-5194-929X

Pascal Vanier[2]
Laboratoire d'Algorithmique, Complexité et Logique
Université Paris-Est Créteil, France
pascal.vanier@lacl.fr

Abstract

We consider the structure of aperiodic points in $\mathbb{Z}^2$-subshifts, and in particular the positions at which they fail to be periodic. We prove that if a $\mathbb{Z}^2$-subshift contains points whose smallest period is arbitrarily large, then it contains an aperiodic point. This lets us characterise the computational difficulty of deciding if an $\mathbb{Z}^2$-subshift of finite type contains an aperiodic point. Another consequence is that $\mathbb{Z}^2$-subshifts with no aperiodic point have a very strong dynamical structure and are almost topologically conjugate to some $\mathbb{Z}$-subshift. Finally, we use this result to characterize sets of possible slopes of periodicity for $\mathbb{Z}^3$-subshifts of finite type.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Subshifts of finite type, Wang tiles, periodicity, aperiodicity, computability, tilings

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.128

Related Version `https://hal.archives-ouvertes.fr/hal-01722008v1`

Acknowledgements The authors wish to thank anonymous reviewers for many helpful remarks and improvements.

A subshift on $\mathbb{Z}^d$ is a set of colorings of $\mathbb{Z}^d$ by a finite set of colors avoiding some family of forbidden patterns. When this family is finite, the subshift is called a *subshift of finite type* (SFT). In dimension 2, SFTs are equivalent to sets of tilings by Wang tiles: Wang tiles are unit squares with colored borders that cannot be rotated and may be placed next to each other only if the borders match.

Wang tiles were introduced by Wang in order to study the decidability of some fragments of logic [18, 19]. He thus introduced the Domino Problem: given a set of Wang tiles, do they tile the plane? (in other words, is the corresponding subshift nonempty?) Wang first conjectured that whenever a tileset tiles the plane, it can do so in a periodic manner, which would have implied the decidability of the Domino Problem.

[1] Sponsored by grant TARMAC ANR 12 BS02 007 01.
[2] Sponsored by grant TARMAC ANR 12 BS02 007 01.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 128; pp. 128:1–128:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In dimension 1 the problem is decidable. A $\mathbb{Z}$-SFT corresponds to the set of biinfinite walks on some automaton and it tiles the line if and only if the automaton contains a cycle. Such a cycle provides a periodic point of the SFT, so non-empty $\mathbb{Z}$-SFTs always contain a periodic point. The situation is dramatically different in higher dimension. Berger [3] proved that there exists tilesets in dimension 2 that tile the plane only aperiodically, and that the Domino Problem was therefore undecidable.

Thus, from the start, periodicity and aperiodicity have been at the heart of the study of Wang tiles and SFTs, and the main tool in understanding their structural properties and the answer to various decision problems. To give a few examples:

- The presence of a dense set of periodic points is related to the decidability of the problem of deciding whether a given pattern appears in some point of an SFT [12].
- The finite subshifts on $\mathbb{Z}^d$ are exactly the subshifts containing only periodic configurations with d non-colinear vectors of periodicity [1, Theorem 3.8]. These configurations can be seen as finite configurations. This result has recently been extended to subshifts on groups [15].
- Countable SFTs always contain a finite configuration and if they are not finite, then they contain a configuration with exactly one vector of periodicity [1, Theorem 3.11].
- A subshift always contains a quasiperiodic configuration [4, 6], a configuration in which every finite pattern appears in any window of sufficiently large size depending only on the size of the pattern.

In this article we study the structure of aperiodic points in $\mathbb{Z}^2$-SFTs, and in particular the repartition of the coordinates where it avoids to be periodic. Our main result is:

▶ **Theorem 1.** *There exists a computable function f that satisfies the following. Assume X is a $\mathbb{Z}^2$-subshift such that for any finite set of periods P, X contains a configuration that avoids all periods of P. Then X contains an aperiodic point that avoids every period p at distance at most $f(\|p\|)$ from 0.*

This means that aperiodicity can be "organised" in concentric balls around a common center, in such a way that a proof of aperiodicity for any vector may be found near this center. As a consequence, when a subshift does not contain any aperiodic point, it must have a finite number of directions of periodicity:

▶ **Corollary 2.** *For any subshift X with no aperiodic point, there is a finite set of periods P such that any configuration of X is periodic for some period $p \in P$.*

This will lead to a further characterization of subshifts containing no aperiodic points in Section 3.2.

These results have a variety of consequences. Gurevich and Koryakov [8] proved that for $d \geq 2$ it is undecidable to know whether an SFT contains a periodic, resp. aperiodic configuration. While it is easy to see that checking whether an SFT contains a periodic configuration is a recursively enumerable problem (Σ_1^0 in the arithmetical hierarchy), it remained an open problem whether deciding if an SFT contains an aperiodic configuration was even in the arithmetical hierarchy. One of the consequences of Theorem 1 is that it is Π_1^0.

Periodicity is also a central topic of symbolic dynamics since sets of periods and directions of periodicity constitute conjugacy invariants. For example, we prove that a $\mathbb{Z}^2$-subshift with no aperiodic point has a very strong dynamical structure and is essentially equivalent (almost topologically conjugate) to some $\mathbb{Z}$-subshift, and this is true for SFTs as well. In particular, various classical decision problems are decidable for this class, its topological entropy is 0 and

its entropy dimension is at most 1. Sets of periods have also been studied and characterized through computability and complexity theory [9]. [16] recently proved that any Σ_2^0 set of $(\mathbb{Q} \cup \{\infty\})^2$ can be realized as a set of slopes of a $\mathbb{Z}^3$-SFT. Another consequence of Theorem 1 is that this becomes a characterization.

The article is organized as follows: Section 1 recalls some definitions and notations, Section 2 is devoted to the proof of Theorem 1, Section 3 is devoted to its consequences and Section 4 shows a counter example for $\mathbb{Z}^d$ subshifts when $d \geq 3$.

1 Definitions

Throughout the paper, we consider the distance on $\mathbb{Z}^d$ defined by the uniform norm $d(i,j) = ||i-j||_\infty$.

1.1 Subshifts

We provide here standard definitions about subshifts, which may be found in greater detail in [13].

The d-dimensional full shift is the set $\Sigma^{\mathbb{Z}^d}$ where Σ is a finite alphabet whose elements are called *letters* or *symbols*. Each element of $\Sigma^{\mathbb{Z}^d}$ is called a *configuration* or *point*. A configuration may be seen as a coloring of $\mathbb{Z}^d$ with the letters of Σ. For $v \in \mathbb{Z}^d$, the *shift function* $\sigma_v : \Sigma^{\mathbb{Z}^d} \to \Sigma^{\mathbb{Z}^d}$ is defined by $\sigma_v(x)_z = x_{z+v}$. The *full shift* equipped with the distance $d(x,y) = 2^{-\min\{\|v\| \mid v \in \mathbb{Z}^d, x_v \neq y_v\}}$ forms a compact metric space on which the shift functions act as homeomorphisms. A closed shift invariant subset X of $\Sigma^{\mathbb{Z}^d}$ is called a *subshift* or *shift*.

A *pattern* of shape Γ, where Γ is a finite subset of $\mathbb{Z}^d$, is an element of Σ^Γ or alternatively a function $p : \Gamma \to \Sigma$. A configuration x *avoids a pattern* γ of shape Γ if $\forall z \in \mathbb{Z}^d$, $\sigma_v(x)_{|\Gamma} \neq \gamma$ and *contains* γ if it does not avoid it.

For a family of forbidden patterns $\mathcal{F}$, denote $X_\mathcal{F}$ the set of configurations that avoid $\mathcal{F}$. Then $X_\mathcal{F}$ is a subshift, and every subshift can be defined in this way. When a subshift can be defined this way by a finite family, it is called a *subshift of finite type*. When a subshift can be defined by a recursively enumerable family of forbidden patterns, it is called an *effectively closed subshift*.

If X is a subshift, we denote by $\mathcal{L}(X)$ its *language*, *i.e.* the set of patterns that appear somewhere in one of its points.

▶ **Definition 3** (Periodicity). A configuration x is *periodic* of period v if there exists $v \in \mathbb{Z}^d \setminus \{(0,0)\}$ such that $\forall z \in \mathbb{Z}^d, x_z = x_{z+v}$. More precisely, a configuration is *k-periodic* if it has exactly k linearly independent periods. If a configuration has no period, then it is said to be *aperiodic*. A subshift is *aperiodic* if all its points are aperiodic.

Denote by $B(z,n)$ the ball of radius n centered in $z \in \mathbb{Z}^d$.

Let $x \in \Sigma^{\mathbb{Z}^d}$ and $p \in \mathbb{Z}^2$. If there exists $z \in \mathbb{Z}^2$ such that $x_z \neq x_{z+p}$, we say that x *avoids* period p. The pair $(z, z+p)$ is called an *avoidance* of period p in configuration x. We say that a configuration *avoids a set of periods* P if it avoids every period in P.

Let P be a set of periods. We denote P' the set obtained from P by replacing each period p by the least commun multiple of all periods of P that are colinear to p. More formally : $P' = \{\text{lcm}(q \mid q \in P \text{ and } q \text{ and } p \text{ are colinear}) \mid p \in P\}$. Observe that P' is a set of pairwise non-colinear periods.

Except in the last section, the subshifts we will be considering will implicitely be $\mathbb{Z}^2$-subshifts.

1.2 Arithmetical hierarchy

We give some basic definitions used in computability theory and in particular about the arithmetical hierarchy. More details may be found in [17].

Usually the arithmetical hierarchy is seen as a classification of sets according to their logical characterization. For our purpose we use an equivalent definition in terms of computability classes and Turing machines with oracles:

- $\Delta_0^0 = \Sigma_0^0 = \Pi_0^0$ is the class of recursive (or computable) problems.
- Σ_n^0 is the class of recursively enumerable (RE) problems with an oracle Π_{n-1}^0.
- Π_n^0 the complementary of Σ_n^0, or the class of co-recursively enumerable (coRE) problems with an oracle Σ_{n-1}^0.
- $\Delta_n^0 = \Sigma_n^0 \cap \Pi_n^0$ is the class of recursive (R) problems with an oracle Π_{n-1}^0.

In particular, Σ_1^0 is the class of recursively enumerable problems and Π_1^0 is the class of co-recursively enumerable problems.

2 Main result

This whole section is dedicated to the proof of the Theorem 1 and Corollary 2. Given a subshift that contains an aperiodic point, we prove that it contains some aperiodic point where all period avoidances are organised in concentric balls around a common center, in such a way that each period p is in a ball whose radius only depends on $\|p\|$. This result is used in a compactness argument to prove that, if a subshift contains configurations whose smallest period is arbitrarily large, then it contains an aperiodic point.

Actually, our algorithm can only gather avoidances in a small ball if all the periods are non-colinear. Fortunately we can easily build a set P'.

▶ **Lemma 4.** *Let P be a set of periods. Any configuration avoiding P' also avoids P.*

Proof. Each period p in P has an integer multiple $p' \in P'$. Each avoidance of p' induces an avoidance of p. ◀

▶ **Lemma 5.** *Let P be a set of pairwise non-colinear periods. Let x be a configuration avoiding P. Then x avoids every period of P in some ball of radius $\sum_{p\in P} \|p\|$.*

Proof. We prove the result by induction on the number of periods in P. When P is a singleton the case is trivial. Now suppose P is not a singleton. Denote $p_0, p_1, \dots, p_n$ the periods in P. By induction hypothesis, we can find a ball B_{n-1} of radius $\sum_{i<n} \|p_i\|$ centered in b_{n-1} containing avoidances for every period in P except p_n. Similarly, we find a ball B'_{n-1} of radius $\sum_{i>0} \|p_i\|$, centered in b'_{n-1} that contains avoidances of every period in P except p_0. We now show that either an avoidance of p_0 exists near a copy of B'_{n-1} or an avoidance of p_n exists near a copy of B_{n-1}.

Consider the ball $B(b_{n-1} + p_n, \sum_{i<n} \|p_i\|)$, the translated image of B_{n-1} by the vector p_n. Either $x_z = x_{z+p_n}$ for all $z \in B_{n-1}$, *i.e.* the two balls are filled the same way, or we found an avoidance $(z, z + p_n)$ with $z \in B_{n-1}$. In the latter case, the result is proved.

As depicted in Figure 1, this process can be iterated for both B'_{n-1} and B_{n-1} until either we find the necessary avoidance or the centers of the balls are close to each other: Since p_0 and p_n are not colinear, and assuming $\|p_n\| \geq \|p_0\|$, there exists $i, j \in \mathbb{Z}$ such that $\left\|b_{n-1} + ip_n - b'_{n-1} + jp_0\right\| \leq \left\|\frac{p_n}{2}\right\| + \left\|\frac{p_0}{2}\right\| < \|p_n\|$. We thus found a ball centered in $b'_{n-1} + jp_0$ and of radius $\sum_i \|p_i\|$ containing the two balls we translated, and therefore an avoidance of each period in P. Denote B_n this new ball and b_n its center. ◀

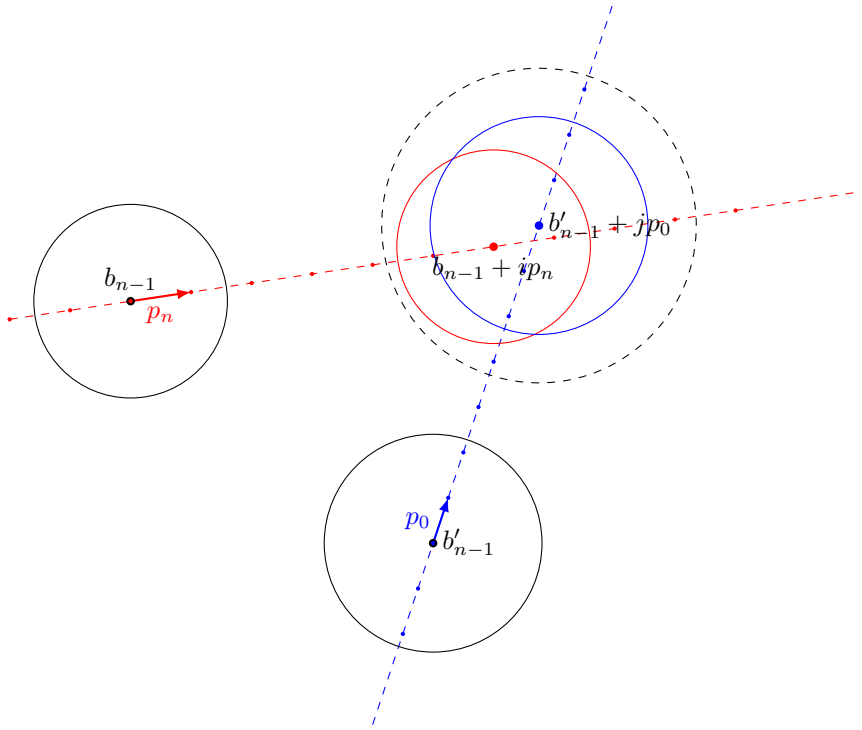


Figure 1 The process of translating B_{n-1} and B'_{n-1} close to each other: each translation may uncover the desired avoidance and if not, the two balls next to each other necessarily do so.

In the previous proof, the distance between b_{n-1} and b_n only depends on p_0, p_n and the distance between b_{n-1} and b'_{n-1}.

Therefore there is a computable function $f(p_0, p_n, r)$ such that, if b_{n-1} and b'_{n-1} belong to a common ball $B(z, r)$, then $\|b_n - z\| \leq f(p_0, p_n, r)$.

▶ **Lemma 6.** *Let $P = \{p_0, \dots, p_n\}$ be a set of non-colinear periods. Define $f'(P, r)$ recursively as:*

- *if $n = 0$, $f'(P, r) = r$;*
- *if $n > 1$, $f'(P, r) = f\big(p_0, p_n, \max\big[f'(P\backslash\{p_0\}, r), f'(P\backslash\{p_n\}, r)\big]\big)$*

Take $x \in X$, and assume that x avoids every period $p \in P$ in some ball $B(z, r)$. Then x avoids every period $p \in P$ in some ball $B(z', \sum_P \|p\|)$, with $\|z' - z\| \leq f'(P, r)$.

Proof. We prove the lemma by induction. If $n = 0$, the result is obvious.

Now assume $n > 1$. By applying the induction hypothesis twice on $p_0, \dots, p_{n-1}$ and $p_1, \dots, p_n$, we find two balls $B_{n-1} = B(b_{n-1}, \sum_{i<n} \|p_i\|)$ and $B'_{n-1} = B(b'_{n-1}, \sum_{i>0} \|p_i\|)$ such that $\|b_n - z\| \leq f'(P\backslash\{p_n\}, r)$ and $\|b'_n - z\| \leq f'(P\backslash\{p_0\}, r)$. Applying Lemma 5 on these balls, we obtain the desired ball with $\|b_n - z\| \leq f'(P, r)$. ◀

The next lemma states that in a configuration avoiding periods $p_1, \dots, p_n$, we can organise the avoidances in concentric balls around a common center, so that the distance of the avoidance of any given period from the center does not depend on n but only on the period itself.

▶ **Lemma 7.** *Let $P_n = \{p_0, \dots, p_n\}$ be a set of periods. Denote $P_i = \{p_0, \dots, p_i\}$ for $i \leq n$. Define recursively a function g such that*

$$g(\{p\}) = \|p\| \quad \text{and} \quad g(P_n) = g(P_{n-1}) + f'\left({P_{n-1}}', \sum_{{P_n}'} \|p\|\right) + \sum_{{P_n}'} \|p\|$$

Take x a point that avoids ${P_n}'$ in a ball $B(z, \sum_{{P_n}'} \|p\|)$. There exist $z' \in \mathbb{Z}^2$ such that:

- $\|z' - z\| \leq g(P_n)$
- *x avoids P_i in the ball $B(z', g(P_i))$ for any $i \leq n$.*

Proof. We proceed by induction on n.

If $n = 0$, then since $B(z, \|p_0\|)$ contains an avoidance of $P_0 = \{p_0\}$, taking z' to be this avoidance satisfies the requisite.

Assume $n > 0$. Since $B(z, \sum_{{P_n}'} \|p\|)$ contains avoidances of every period in ${P_n}'$, it contains avoidances of every period in ${P_{n-1}}'$. Indeed, if some period q is in ${P_{n-1}}'$ but not in ${P_n}'$, then by construction a multiple of q, say Mq, is in ${P_n}'$. Now if $(z_q, z_q + Mq)$ is an avoidance of Mq, at least one of $(z_q + mq, z_q + (m+1)q)$ for $m < M$ is an avoidance of q and is contained in the same ball.

Applying Lemma 6 on ${P_{n-1}}'$, we find a ball $B(z_0, \sum_{{P_{n-1}}'} \|p\|)$ that contains avoidances for all periods in ${P_{n-1}}'$ and such that $\|z_0 - z\| \leq f'({P_{n-1}}', \sum_{{P_n}'} \|p\|)$.

Now apply the induction hypothesis on this ball, obtaining z' such that $\|z' - z_0\| \leq g(P_{n-1})$ and for any $i \leq n-1$, the ball $B(z', g(P_i))$ contains avoidances of every period in P_i. This inductive process is depicted in Figure 2.

By the triangular inequality, $\|z' - z\| \leq g(P_{n-1}) + f'({P_n}', \sum_{{P_{n-1}}'} \|p\|)$. Since $g(P_n) \geq \|z - z'\| + \sum_{{P_n}'} \|p\|$, $B(z', g(P_n))$ contains entirely the ball $B(z, \sum_{{P_n}'} \|p\|)$. Therefore $B(z', g(P_n))$ avoids P_n and $\|z' - z\| \leq g(P_n)$, proving the lemma. ◀

▶ **Theorem 1.** *Let X be a $\mathbb{Z}^2$-subshift. Assume that for every finite set of periods P, X contains a configuration that avoids all periods of P.*

Then X contains an aperiodic configuration that has an avoidance of every period p at distance at most $g'(\|p\|) = g(B(0, \|p\|)')$ from 0, where g is the function defined in Lemma 7.

▶ Remark. g' is not polynomial because $B(0,n)'$ contains exponentially large vectors. Since our bounds are likely very far from sharp, we leave the exact computation to the reader.

Proof. Take $B(0,n)$ the set of periods of norm n or less and $x_1, \dots, x_n, \dots$ a sequence of configurations such that x_n avoids $B(0,n)'$.

By applying Lemma 5 on $B(0,n)'$, we obtain for each x_n a ball of radius $\sum_{B(0,n)'} \|p\|$ that avoids $B(0,n)'$. Applying Lemma 7 on $B(0,n)$, we get that x_n avoids all periods in $B(0,i)'$ in some ball $B(z_n, g(B(0,i))'))$ for all $i \leq n$. Since X is compact, any limit point of the sequence $(\sigma_{z_n}(x_n))_{n \in \mathbb{N}}$ is in X, and it avoids all periods in $B(0,i)$ in the ball $B(0, g(B(0,i)'))$ for all $i \in \mathbb{N}$. It is in particular aperiodic. ◀

▶ **Corollary 2.** *For any subshift X with no aperiodic point, there is a finite set of periods P such that any configuration of X is periodic for some period $p \in P$.*

Proof. This is a simple reciprocal to Theorem 1. ◀

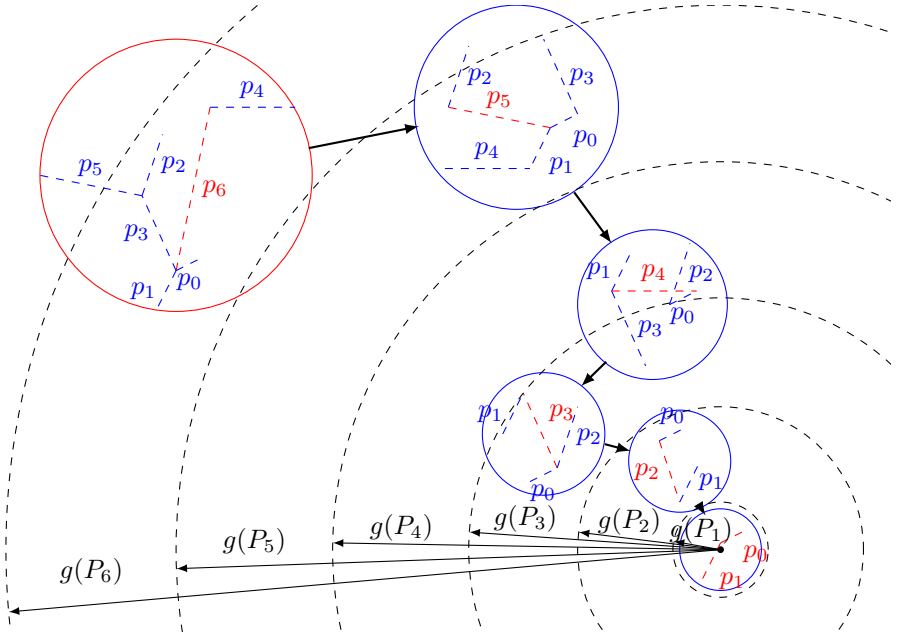


Figure 2 The red ball contains avoidance for $P_n{}'$. Each dashed line represents an avoidance. We then consider the set of blue lines, $P_{n-1}{}'$. We gather these avoidances in a close smaller ball with Lemma 6. We repeat this process until there are only two avoidances in the ball. The red avoidances are disposed in the way we wanted, around the center of the smallest blue ball.

3 Consequences

3.1 Existence of an aperiodic configuration is Π_1^0

▶ **Corollary 8.** *The following problem is Π_1^0-computable:*
Input *A finite set of forbidden patterns $\mathcal{F}$.*
Output *Does the $\mathbb{Z}^2$-SFT $X_\mathcal{F}$ contain an aperiodic configuration?*

Proof. Let $(p_i)_{i\in\mathbb{N}}$ be an enumeration of all possible periods and $P_n = \{p_0, \dots, p_n\}$. Theorem 1 gives us a bound on the size of the patterns in which to look for avoidances of each period. For each $n \in \mathbb{N}$ in order, the algorithm enumerates all patterns on a ball of radius $g(P_n)$ that do not contain a forbidden pattern, and check if one of them contains avoidances for every period of P_k ($k \leq n$) in the ball of radius $g(P_k)$ in its center. If such a pattern does not exist for some n, it means that either the SFT is empty or that all its points are periodic for some period p with $\|p\| \leq n$.

Assume the algorithm runs infinitely. For every k, there exists some $n \geq k$ such that if a pattern on the ball of radius $g(P_n)$ does not contain a forbidden pattern, then the subpattern on the ball of radius $g(P_k)$ is in the language of X. Therefore, for each P_k we find a pattern in $Ł(X)$ that avoids all periods of P_k, and we conclude by Theorem 1. ◀

3.2 Structure of subshifts without aperiodic points

In this subsection, we consider notions from dynamical system theory. A dynamical system is given by a pair (C, Φ) where C is a compact set and $\Phi : C \to C$ is a continuous function.

▶ **Definition 9** (Topological conjugacy). Let (C_1, Φ_1) and (C_2, Φ_2) be dynamical systems.

(C_1, Φ_1) and (C_2, Φ_2) are *topologically conjugate* if there exists a continuous bijection $\pi : C_1 \to C_2$ such that $\pi \circ \Phi_1 = \Phi_2 \circ \pi$.

(C_1, Φ_1) and (C_2, Φ_2) are *almost topologically conjugate* if there exists (C_3, Φ_3) and continuous surjections $\pi_i : C_3 \to C_i$ that are bijective almost everywhere[3] such that $\pi_i \circ \Phi_3 = \Phi_i \circ \pi_i$ for $i = 1, 2$.

See [13] or [11] for more information on topological conjugacy and almost conjugacy in the context of symbolic dynamics. We need slightly more general definitions since we consider subshifts of different dimensions.

Notice that we can have $(C_2, \Phi_2) = (C_3, \Phi_3)$ (and $\pi_2 = id$) in the last definition; this is the case in the next proof.

▶ **Theorem 10.** *Let X be a two-dimensional subshift with no aperiodic point. There exists a vector v and a one-dimensional subshift Y such that (X, σ_v) is almost topologically conjugate to (Y, σ).*

If X is of finite type, then Y can be chosen of finite type as well.

Proof. Let X be a two-dimensional subshift of finite type with no aperiodic point. By Corollary 2, there is a finite set of periods P such that any configuration of X is periodic of some period $p \in P$. We assume that P does not contain two colinear periods, by taking their least common integer multiple if necessary.

For the clarity of the argument, we assume in the following that P does not contain any period colinear to $(0, 1)$. Since P is finite, the proof can be adapted for a different vector.

Take $p = (p^0, p^1) \in P$, assuming $p^0 > 0$, and denote $X_p = \{x \in X : x \text{ admits } p \text{ as a period.}\}$. X_p is a closed set and it is a classical argument (see for instance [9, §2.1.2] or [2, Lemma 5.2]) that $(X_p, \sigma_{(0,1)})$ is topologically conjugate to a one-dimensional SFT, which we repeat here for completeness. Define:

$$\pi_p = \left\{ \begin{array}{ccc} \Sigma^{\mathbb{Z}^2} & \to & (\Sigma^{p^0-1})^{\mathbb{Z}} \\ x & \mapsto & ((x_{i,j})_{0 \le i < p^0})_{j \in \mathbb{Z}} \end{array} \right.$$

Denote $Y_p = \pi_p(X_p)$. It is not hard to see that π_p is a one-to-one continuous function between X_p and Y_p and that Y_p is a subshift of finite type if X_p is, since it can be defined by a finite recoding of the forbidden patterns of X_p. Furthermore, $\pi_p \circ \sigma_{(0,1)} = \sigma \circ \pi_p$, so it is a topological conjugacy between $(X_p, \sigma_{(0,1)})$ and (Y_p, σ).

For any $p_1 \neq p_2 \in P$, $X_{p_1} \cap X_{p_2}$ is a set of 2-periodic configurations that admit non-colinear periods p_1 and p_2; there are a finite number of such configurations, so $|X_{p_1} \cap X_{p_2}| < +\infty$. In other words, $X = \bigcup_{p \in P} X_p$ and the union is disjoint except for a finite set of configurations.

Denote $Y = \sqcup_{p \in P} Y_p$ (disjoint union). Y is a subshift on the alphabet $\sqcup_{p \in P} \Sigma_p$, where Σ_p is the alphabet of Y_p. Furthermore, Y is of finite type if every Y_p is of finite type.

Define $\varphi : Y \to X$ by $\varphi|_{Y_p} = \pi_p^{-1}$. We can check that φ is surjective and almost everywhere bijective, and that $\varphi \circ \sigma_1 = \sigma_{(0,1)} \circ \varphi$. We have proved that (Y, σ_1) is almost topologically conjugate to $(X, \sigma_{(0,1)})$. ◀

3.3 Various properties of subshifts with no aperiodic points

Theorem 10 implies that the property of having no aperiodic point gives a very strong structure to a subshift. This is particularly the case for subshifts of finite type, where many

[3] Except for a finite set of points.

problems that are indecidable in dimension 2 are completely solved in dimension 1, and these solutions carry through almost topological conjugacy.

In this section, we make use of notations that were defined in the proof of Theorem 10: X_p, Y_p, π_p and φ.

3.3.1 Decision problems

Decision problems have been a staple of the theory of multidimensional subshifts of finite type: the seminal paper of Wang proved that the emptiness problem (given a list of forbidden patterns $\mathcal{F}$, is $X_\mathcal{F} = \emptyset$?) is decidable for two-dimensional non-aperiodic subshifts of finite type, but Berger later proved that the problem was undecidable without this assumption [3]. We consider other classical decision problems: the extension problem, which is undecidable for multidimensional subshifts of finite type (as a consequence of the above), and the injectivity and surjectivity problems, which are undecidable even on the two-dimensional full shift [10].

A $\mathbb{Z}^d$-*cellular automaton* is a continuous function $F : \Sigma^{\mathbb{Z}^d} \to \Sigma^{\mathbb{Z}^d}$ that commutes with every shift function. It can be defined equivalently by a local rule $f : \Sigma^\Gamma \to \Sigma$ for a finite shape Γ by $F(x)_v = f(\sigma_v(x)_\Gamma)$ for all $v \in \mathbb{Z}^d$.

▶ **Corollary 11.** *The following problems are decidable for two-dimensional subshifts of finite type with no aperiodic point:*

Extension problem *given a list of forbidden patterns $\mathcal{F}$ and a pattern w, do we have $w \in \mathcal{L}(X_\mathcal{F})$?*

Injectivity / surjectivity problem *given a list of forbidden patterns $\mathcal{F}$ and a cellular automaton $\Phi : \Sigma^{\mathbb{Z}^2} \to \Sigma^{\mathbb{Z}^2}$, is $\Phi|_{X_\mathcal{F}}$ injective? surjective on $X_\mathcal{F}$?*

Links between periodic points and the above problems have already been considered in [12, 7].

Proof sketches.

Extension problem. Assume w has shape $[-n, n]^2$. By Theorem 10 we have $X = \varphi(Y)$ where φ is continuous on a compact space, hence uniformly continuous. In other words, for every n, there exists r such that the value of $\varphi(y)_{[-n,n]^2}$ only depends on $y_{[-r,r]}$. Since the extension problem is decidable on one-dimensional subshifts of finite type, enumerate all words $v \in \mathcal{L}(Y)$ and check whether $w = \varphi(v)$ for some v.

Injectivity problem. By Corollary 2, there is a finite set of periods P such that $X_\mathcal{F} = \bigcup_{p \in P} X_p$.

A cellular automaton is injective if and only if it is reversible. Since the image of a configuration of period p by a cellular automaton also has period p, we have $\Phi|_{X_\mathcal{F}}(X_p) \subset X_p$ and (in the injective case) $\Phi|^{-1}_{X_\mathcal{F}}(X_p) \subset X_p$. It follows that $\Phi|_{X_\mathcal{F}}$ is injective if, and only if, $\Phi|_{X_p}$ is injective for every period $p \in P$.

Let $\pi_p : X_p \to Y_p$ be the continuous bijection defined in the proof of Theorem 10. $\pi_p \circ \Phi|_{X_p} \circ \pi_p^{-1}$ is a CA on Y_p and it shares injectivity with $\Phi|_{X_\mathcal{F}}$. Injectivity of CA is decidable for one-dimensional subshifts of finite type [7].

Surjectivity problem. Surjectivity is more delicate as a point in $X_p \cap X_q$ can be the image of a point from X_p or X_q. However, $\Phi|_{X_\mathcal{F}}$ is surjective if and only if:

1. $\forall p \in P,\ \forall x \in X_p \setminus \bigcup_{p' \neq p} X_{p'},\ x \in \Phi|_{X_\mathcal{F}}(X_p)$;
2. $\forall p \neq p' \in P,\ \forall x \in X_p \cap X_{p'},\ \exists p'' \in P,\ x \in \Phi|_{X_\mathcal{F}}(X_{p''})$,

Denote $X^\cap$ the finite set of configurations in case 2 (notice that we do not necessarily have $p'' = p$ or $p'' = p'$ if x admits other periods as well).
As in the previous case, we translate these properties on Y_p and $\Phi_p = \pi_p \circ \Phi|_{X_p} \circ \pi_p^{-1}$ to work on $\mathbb{Z}$-SFT. Following [7], we can describe $\Phi_p(Y_p)$ by a finite automaton.
For case 1, add to the finite automaton describing $\Phi_p(Y_p)$ an independent cycle for each element of $X^\cap$ and determine whether the resulting automaton describe the same SFT as the automaton describing Y_p. This algorithm is explained in [13], Section 3.4.
For case 2, since $x \in X^\cap$ is 2-periodic, $\pi_p(x)$ is periodic, and it is easy to check by hand whether some $\Phi_p(Y_p)$ accepts x. Do this for all $x \in X^\cap$. ◀

▶ Remark. If we did not know that X admits a finite set of periods, the first proof would still show that the extension problem is in Σ_1^0 (RE). Since it is easy to show that it is in Π_1^0 (co-RE), our main result is technically unnecessary here.

3.3.2 Topological entropy

Topological entropy is a widely-used parameter in information theory (channel capacity) and dynamical systems theory (conjugacy invariant). Entropy dimension is a more refined notion for systems of entropy zero, introduced in [5] and mainly used for multidimensional subshifts [14].

▶ **Corollary 12.** *Any two-dimensional subshift X with no aperiodic point has zero topological entropy. Its entropy dimension is at most one.*

Proof sketch. By Corollary 2, there is a finite set of periods P such that $\mathcal{L}(X) = \bigcup_{p \in P} \mathcal{L}(X_p)$. Consider a pattern w of shape $[0, n-1]^2$ in $\mathcal{L}(X_p)$, assuming for clarity that $p = (p^0, p^1)$ with $p^0 \geq 0$ and $p^1 \geq 0$. Since w cannot contain an avoidance for p, it is entirely determined by its p^0 bottommost rows and p^1 leftmost columns. Therefore there are at most $(p^0 + p^1)n$ such patterns. A similar argument applies when $p^0 < 0$ or $p^1 < 0$.

It follows that there are at most $\sum_p (|p^0| + |p^1|)n$ patterns of shape $[0, n-1]^2$ in $\mathcal{L}(X)$, proving the statement. ◀

3.3.3 Density of periodic points

Density of periodic points is a typical question in dynamical systems, for example when studying chaos in the sense of Devaney. See [7] for more details, including a proof that two-dimensional subshifts of finite type do not have dense 2-periodic points in general, even under an additional irreducibility hypothesis.

X is *irreducible* (or *transitive*) if for any two patterns $\gamma_1,\ \gamma_2 \in \mathcal{L}(X)$ of shapes Γ_1 and Γ_2 respectively, there exists $x \in X$ and two coordinates v_1, v_2 such that $\sigma_{v_1}(x)_{\Gamma_1} = \gamma_1$ and $\sigma_{v_2}(x)_{\Gamma_2} = \gamma_2$.

▶ **Corollary 13.** *Any irreducible two-dimensional subshift of finite type X with no aperiodic point has dense 2-periodic points.*

Proof sketch. By Corollary 2, consider P a finite and minimal set of periods such that any configuration of X is periodic for some period $p \in P$. If P is not a singleton, take $p_1 \neq p_2 \in P$. There exists two finite patterns γ_1 and γ_2 that contain an avoidance of p_1 and p_2, respectively (otherwise, P would not be minimal). By irreducibility, there exists $x \in X$ where γ_1 and γ_2 both appear, and therefore x avoids every $p \in P$, a contradiction. Therefore P is a singleton $\{p\}$ and $X = X_p$ is conjugated to Y_p. One-dimensional irreducible subshifts of finite type, such as Y_p, have dense periodic points ([7], Proposition 9.1). The image by π_p^{-1} of a periodic point in Y_p is a 2-periodic point in X, from which the statement follows. ◀

3.4 The full caracterization of slopes of $\mathbb{Z}^3$-SFTs

Intuitively, slopes of a subshift are the directions that some configuration admits as a unique direction of periodicity. More formally:

▶ **Definition 14.** Let X be a $\mathbb{Z}^d$-subshift. $\theta \in (\mathbb{Q} \cup \infty)^{d-1}$ is a *slope of periodicity* of X if there exists a configuration $x \in X$ and a vector $v \in \mathbb{Z}^d$ such that:
- $v\mathbb{Z} = \{v' \mid \sigma_{v'}(x) = x\}$
- and $\theta_i = v_1/v_{i+1}$, for all $i \in \{0, \dots, d-1\}$.

The set of slopes of periodicity of a subshift is a conjugacy invariant. A consequence of Corollary 8 is that the sets of slopes of periodicity of $\mathbb{Z}^3$-SFTs is a Σ^0_2-computable set, and together with [16] this implies the following caracterization:

▶ **Theorem 15.** *Σ^0_2-computable subsets of $S \subseteq (\mathbb{Q} \cup \{\infty\})^2$ are exactly the sets realizable as sets of slopes of $\mathbb{Z}^3$-subshifts of finite type.*

Proof. We know from [16] that one can realize any such Σ^0_2 set S as a set of slopes of a $\mathbb{Z}^3$-subshift. Let us now show the remaining direction.

Given a slope θ and a set of forbidden patterns $\mathcal{F}$ as an input, we want to check whether there exists a configuration in $X_\mathcal{F}$ whose vectors of periodicity all have direction θ.

Using the notations of the proof of Theorem 10, for any $p \in \mathbb{Z}^2$, the set X_p of configurations of period p (for some $k > 0$) can be seen as a $\mathbb{Z}^2$-SFT Y computable from $\mathcal{F}$ and p.

There is a smallest vector p_θ such that all vectors of direction θ are integer multiples of it. Remark that θ is a slope of periodicity of $X_\mathcal{F}$ is and only if X_{kp_θ} contains an aperiodic configuration for some $k > 0$. By Corollary 8, checking whether X_{kp_θ} contains an aperiodic configuration for a given k is Π^0_1-computable. Therefore checking whether there is a $k > 0$ for which this holds is Σ^0_2-computable. ◀

4 Counterexample for dimensions $d > 2$

We build a counterexample to Theorem 1 in higher dimension. Take $\Sigma = \{0, 1\}$ and define X as follows:
- All symbols 1 must form lines of direction vector $(1, 0, 0)$ (horizontal) or $(0, 0, 1)$ (vertical);
- There is at most one vertical line;
- All horizontal lines are repeated periodically with period $(0, 0, n)$, where n is the distance of the vertical line to any horizontal line.

In particular, if there is no vertical line, then there is at most one horizontal line. To sum up, a subshift configuration can be :
1. all zeroes,
2. one horizontal line,
3. one vertical line, or
4. the situation depicted in Figure 3.

The configuration described in Figure 3 admits $(0, 0, n)$ as period, and no shorter period. In particular, for every finite set of periods P, X contains a configuration that avoids P (by taking n large enough). However, Σ admits no aperiodic point[4]. This example can easily be

[4] Notice that there cannot be a configuration with a single horizontal line and a single vertical line, which would be aperiodic.

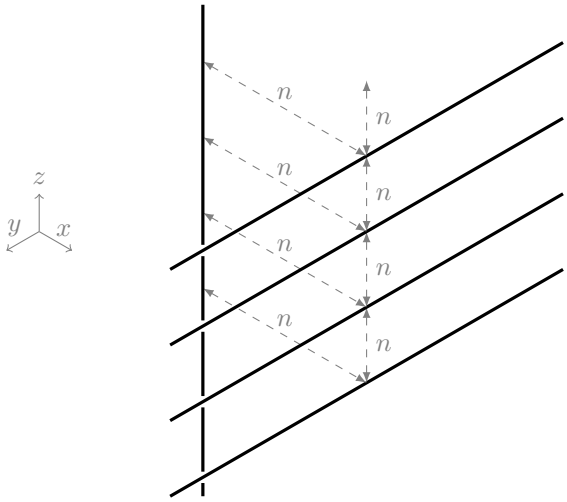


Figure 3 A typical configuration of X: a line of ones along z at distance n of an (xy) plane of lines along x. The only other types of configurations of X are the configurations containing either a single vertical line, a single horizontal line, or no line at all.

generalised to any $d > 3$ by considering a $\mathbb{Z}^d$-subshift X' that contains a copy of X in at most one coordinate, and 0 everywhere else: that is,

$$x \in X' \Leftrightarrow \forall j \in \mathbb{Z}^{d-3}, (x_{i,j})_{i\in\mathbb{Z}^3} \in X \text{ and } (\forall j_1 \neq j_2, (x_{i,j_1})_{i\in\mathbb{Z}^3} = 0 \text{ or } (x_{i,j_2})_{i\in\mathbb{Z}^3} = 0).$$

This proves that Theorem 1 does not hold in any dimension $d > 2$.

5 Open problems

We have made clear that our main result does not hold for subshifts of dimension $d \geq 3$. We do not know, however, whether Theorem 10 or Corollary 12 holds in higher dimension, since the counterexample introduced in Section 4 does not contradict these results.

This counterexample is a subshift containing points with arbitrarily large periods but no aperiodic point. We do not know whether such a counterexample with infinitely many directions of periodicity exist. Moreover, the structure of d-dimensional subshifts of finite type for $d \geq 3$ remains open; the existence of this counterexample suggests that a making use of the finite type hypothesis is necessary in higher dimension.

References

1 Alexis Ballier, Bruno Durand, and Emmanuel Jeandel. Structural aspects of tilings. In *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*, pages 61–72, 2008. doi:10.4230/LIPIcs.STACS.2008.1334.

2 Alexis Ballier and Emmanuel Jeandel. Structuring multi-dimensional subshifts. *CoRR*, abs/1309.6289, 2013. arXiv:1309.6289.

3 Robert Berger. *The Undecidability of the Domino Problem*. PhD thesis, Harvard University, 1964.

4 M.-G. D. Birkhoff. Quelques théorèmes sur le mouvement des systèmes dynamiques. *Bulletin de la SMF*, 40:305–323, 1912.

5 Maria De Carvalho. Entropy dimension of dynamical systems. *Portugaliae Mathematica*, 54:19–40, 1997.

6 Bruno Durand. Tilings and Quasiperiodicity. *Theoretical Computer Science*, 221(1-2):61–75, jun 1999. doi:10.1016/S0304-3975(99)00027-4.

7 Francesca Fiorenzi. Periodic configurations of subshifts on groups. *International Journal of Algebra and Computation*, 19(03):315–335, 2009.

8 Yuri Gurevich and I Koryakov. Remarks on Berger's paper on the domino problem. *Siberian Math. Journal*, pages 319–320, 1972.

9 Emmanuel Jeandel and Pascal Vanier. Characterizations of periods of multidimensional shifts. *Ergodic Theory and Dynamical Systems*, 35(2):431–460, 2015. doi:10.1017/etds.2013.60.

10 Jarkko Kari. Reversibility and surjectivity problems of cellular automata. *Journal of Computer and System Sciences*, 48(1):149–182, 1994. doi:10.1016/S0022-0000(05)80025-X.

11 B.P. Kitchens. *Symbolic Dynamics: One-sided, Two-sided and Countable State Markov Shifts*. Universitext. Springer Berlin Heidelberg, 1997.

12 Bruce Kitchens and Klaus Schmidt. Periodic points, decidability and Markov subgroups. In Berlin-Heidelberg-New York Springer Verlag, editor, *Dynamical Systems, Proceeding of the Special Year*, volume 1342 of *Lecture Notes in Mathematics*, pages 440–454, 1988.

13 Douglas A. Lind and Brian Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, New York, NY, USA, 1995.

14 Tom Meyerovitch. Growth-type invariants for $\mathbb{Z}^d$ subshifts of finite type and arithmetical classes of real numbers. *Inventiones mathematicae*, 184(3):567–589, 2011.

15 Tom Meyerovitch and Ville Salo. On pointwise periodicity in tilings, cellular automata and subshifts, 2017. arXiv:1703.10013.

16 Etienne Moutot and Pascal Vanier. Slopes of 3-dimensional subshifts of finite type. In *Computer Science - Theory and Applications - 13th International Computer Science Symposium in Russia, CSR 2018, Moscow, Russia, June 6-10, 2018, Proceedings*, page to appear, 2018.

17 Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. MIT Press, Cambridge, MA, USA, 1987.

18 Hao Wang. Proving Theorems by Pattern Recognition I. *Communications of the ACM*, 3(4):220–234, apr 1960. doi:10.1145/367177.367224.

19 Hao Wang. Notes on a class of tiling problems. *Fundamenta Mathematicae*, 82:295–305, 1975.

Semicomputable Geometry

Mathieu Hoyrup
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
mathieu.hoyrup@inria.fr

Diego Nava Saucedo[1]
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
diego.nava-saucedo@wanadoo.fr

Don M. Stull[2]
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
donald.stull@inria.fr

Abstract

Computability and semicomputability of compact subsets of the Euclidean spaces are important notions, that have been investigated for many classes of sets including fractals (Julia sets, Mandelbrot set) and objects with geometrical or topological constraints (embedding of a sphere). In this paper we investigate one of the simplest classes, namely the filled triangles in the plane. We study the properties of the parameters of semicomputable triangles, such as the coordinates of their vertices. This problem is surprisingly rich. We introduce and develop a notion of semicomputability of points of the plane which is a generalization in dimension 2 of the left-c.e. and right-c.e. numbers. We relate this notion to Solovay reducibility. We show that semicomputable triangles admit no finite parametrization, for some notion of parametrization.

2012 ACM Subject Classification Theory of computation → Computability

Keywords and phrases Computable set, Semicomputable set, Solovay reducibility, Left-ce real, Genericity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.129

Related Version A full version of the article including all the proofs is available at `https://hal.inria.fr/hal-01770562`.

Acknowledgements The authors would like to thank Daniel Graça, Emmanuel Jeandel and Cristóbal Rojas for interesting discussions on this topic, and three anonymous referees for helful comments.

1 Introduction

The notions of computable and computably enumerable sets of discrete objects such as $\mathbb{N}$ have been extended to sets of continuous objects such as real numbers. Arguably the most successful notions are defined for closed subsets of $\mathbb{R}^n$, especially $\mathbb{R}^2$ where they have a graphical interpretation. A computable subset of $\mathbb{R}^2$ corresponds to the intuitive notion of a set that can be drawn on a screen with arbitrary resolution by a single program. The computability of famous sets have been investigated in many articles. Whether the Mandelbrot set is computable is an open problem [6], related to a conjecture in complex

[1] Supported by LORIA.
[2] Supported by Inria Nancy Grand-Est.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 129; pp. 129:1–129:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

dynamics. It has been shown that filled Julia sets are computable, while their boundaries are not always computable [2]. The computability of the Lorenz attractor has been addressed in [5] and is still an open problem.

While the computability of such sets is usually a difficult question, the mathematical definitions of these sets immediately enable one to *semicompute* them, in the same way as one can only semicompute the halting problem: if a pixel does not intersect the set then this can be recognized in finite time, but if it does not then one may never know. For instance, the set of fixed-points of a computable function is semicomputable: if $x \neq f(x)$ then it can be eventually discovered by computing $f(x)$ with sufficient precision, but if $x = f(x)$ then we will never know.

Several studies have shown that topological or geometrical constraints on a semicomputable set make it computable [9, 8, 3].

In this paper, we study one of the simplest family of geometrical objects, namely filled triangles in $\mathbb{R}^2$. Part of the study extends to other classes of compact convex subsets of $\mathbb{R}^2$. While a filled triangle is computable if and only if the parameters defining it (coordinates, lengths, angles, etc.) are computable, the case of semicomputable triangles is less clear and leads us to several investigations.

We give a first characterization of semicomputable triangles. We introduce the notion of a semicomputable point, which is essentially a point that can be computably approximated from a limited set of directions. We show that determining whether a triangle is semicomputable reduces to identifying the semicomputability ranges of its vertices. We then study the properties of the semicomputability range and develop tools to help determining it, notably the quantitative version of Solovay reducibility which was independently introduced and studied in [1, 10].

We study the (non-)computability of several parameters associated to triangles by investigating the properties of generic semicomputable triangles, which are in a sense the most typical ones and are far from being computable.

We end this paper with a slightly different viewpoint, by showing that the problem is inherently complex in that the semicomputability of a triangle cannot be reduced to the semicomputability of its parameters, for any *finite* parametrization. This result is proved for a particular notion of parametrization, but other notions are possible and should be studied in the future.

Several proofs are not included due to space limitations. A full version of the article including all the proofs is available at `https://hal.inria.fr/hal-01770562`.

1.1 Background

A real number x is ***computable*** if there is a computable sequence of rationals q_i such that $|x - q_i| < 2^{-i}$. A real number x is ***left-c.e.*** if there is a computable increasing sequence of rationals converging to x, and ***right-c.e.*** if there is a computable decreasing sequence converging to x. A real number is ***difference-c.e.*** or ***d-c.e.*** if it is a difference of two left-c.e. numbers.

A rational box $B \subseteq \mathbb{R}^n$ is a product of n open intervals with rational endpoints. Let $(B_i)_{i\in\mathbb{N}}$ be a canonical enumeration of the rational boxes. A set $U \subseteq \mathbb{R}^d$ is an ***effective open set*** if it is a the union of a computable sequence of rational boxes. A ***semicomputable set*** is the complement of an effective open set. An ***effectively compact set*** is a compact set $K \subseteq \mathbb{R}^d$ such that the set $\{\langle i_1, \dots, i_k\rangle : K \subseteq B_{i_1} \cup \dots \cup B_{i_k}\}$ is c.e. ($\langle . \rangle : \mathbb{N}^* \to \mathbb{N}$ is a computable bijection). Equivalently, K is effectively compact if and only if K is bounded and semicomputable.

A function $f : A \subseteq \mathbb{R}^d \to \mathbb{R}^e$ is ***computable*** if the sets $f^{-1}(R_i)$ are uniformly effective open sets on A, i.e. if there exist uniformly effective open sets $U_i \subseteq \mathbb{R}^d$ such that $f^{-1}(R_i) = U_i \cap A$. A function $f : A \subseteq \mathbb{R}^d \to \mathbb{R}$ is ***left-c.e.*** if the sets $f^{-1}(q_i, +\infty)$ are uniformly effective open sets on A. Every bounded left-c.e. function $f : A \subseteq \mathbb{R}^d \to \mathbb{R}$ has a left-c.e. extension $\hat{f} : \mathbb{R}^d \to \mathbb{R}$.

Let $f : \mathbb{R}^d \times \mathbb{R}^e \to \mathbb{R}$ be left-c.e.

- If $K \subseteq \mathbb{R}^e$ is a non-empty effectively compact set then the function $f_{\min} : \mathbb{R}^d \to \mathbb{R}$ defined by $f_{\min}(x) = \min_{y \in K} f(x, y)$ is left-c.e.
- If $U \subseteq \mathbb{R}^e$ is an effective open set then $f_{\sup} : \mathbb{R}^d \to \mathbb{R}$ defined by $f_{\sup}(x) = \sup_{y \in U} f(x, y)$ is left-c.e.

2 Semicomputability of convex sets

In dimension 1, a compact convex set is simply a closed interval. Such a set $[a, b]$ is semicomputable exactly when a is left-c.e. and b is right-c.e., i.e. when the extremal points of the set have computable approximations oriented inwards the set. It can be generalized to certain compact convex sets of the plane. While in $\mathbb{R}$ there are only two possible directions, in $\mathbb{R}^2$ there are infinitely many ones, represented by angles.

Let $A = (x, y)$ be a point of the plane. For $\theta \in \mathbb{R}$, the θ-coordinate of A is $A_\theta = x \cos\theta + y \sin\theta = (OA, u_\theta)$, i.e., the inner product of the vector $OA = (x, y)$ with $u_\theta = (\cos\theta, \sin\theta)$ ($O = (0, 0)$ is the origin). Observe that the computability properties of A_θ do not depend on the choice of the origin, as long as it is computable.

▶ **Definition 2.1.** If θ is computable then we say that A is ***θ-c.e.*** if A_θ is left-c.e. For a closed interval $I = [a, b]$, we say that A is ***I-c.e.*** if the function mapping $\theta \in I$ to A_θ is left-c.e.

For a non-empty compact convex set S and $\theta \in \mathbb{R}$, define $S_\theta = \min_{X \in S} X_\theta$, and for an extremal point V of S let $J_V^S = \{\theta \in \mathbb{R} : S_\theta = V_\theta\}$. J_V^S is a closed interval modulo 2π.

▶ **Proposition 2.2.** *A non-empty compact convex set S is semicomputable iff the function mapping θ to S_θ is left-c.e. iff for $\theta \in \mathbb{Q}$, S_θ is uniformly left-c.e.*

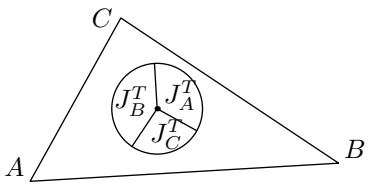
Proof. Assume that S is semicomputable, or equivalently effectively compact. The function $(A, \theta) \mapsto A_\theta$ is computable so the function $\theta \mapsto \min_{A \in S} A_\theta$ is left-c.e.

Conversely, assume that the function $\theta \mapsto S_\theta$ is left-c.e. For each θ let H_θ be the closed half-plane defined by $H_\theta = \{P \in \mathbb{R}^2 : P_\theta \geq S_\theta\}$. H_θ is semicomputable relative to and uniformly in θ, so $S = \bigcap_{\theta \in [0, 2\pi]} H_\theta$ is semicomputable as $[0, 2\pi]$ is effectively compact.

The function $\theta \mapsto S_\theta$ is L-Lipschitz for some L, so if for all $q \in \mathbb{Q}$, the number S_q is uniformly left-c.e. then the function is left-c.e. as $S_\theta = \sup\{S_q - L|q - \theta| : q \in \mathbb{Q}\}$. ◀

For a triangle, and more generally a convex polygon, the number of extremal points is finite and Proposition 2.2 can be improved as follows.

▶ **Theorem 2.3.** *A filled triangle $T = ABC$ is semicomputable iff each vertex $V \in \{A, B, C\}$ is J_V^T-c.e.*



In order to prove the theorem, we need the following Lemma.

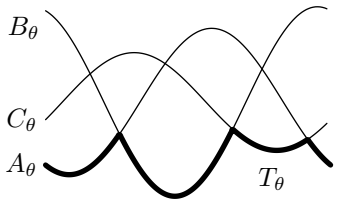
▶ **Lemma 2.4.** *Let $f : [a, b] \to \mathbb{R}$ be left-c.e. and such that there exists $\epsilon > 0$ such that f is non-increasing on $[a, a+\epsilon)$ and non-decreasing on $(b-\epsilon, b]$. There exists a left-c.e. extension $\hat{f} : \mathbb{R} \to \mathbb{R}$ of f that is non-increasing on $(a-\epsilon, a+\epsilon)$ and non-decreasing on $(b-\epsilon, b+\epsilon)$ and $\hat{f} = +\infty$ outside $(a-\epsilon, b+\epsilon)$.*

Proof. Let f_0 be a left-c.e. extension of f. Let $q, q', r, r' \in \mathbb{Q}$ satisfy $q < a < q' < r' < b < r$, $q' - q < \epsilon$ and $r - r' < \epsilon$. Define $\hat{f}(x) = f(x)$ if $x \in [q', r']$, $\hat{f}(x) = \sup_{[x,q']} f_0$ if $x \in [q, q']$, $\hat{f}(x) = \sup_{[r',x]} f_0$ if $x \in [r', r]$, $\hat{f}(x) = +\infty$ if $x < q$ or $x > r$. ◀

Proof of Theorem 2.3. If T is semicomputable then the function $\theta \mapsto T_\theta$ is left-c.e. It coincides with the function $\theta \mapsto V_\theta$ on J_V^T, so V is J_V^T-c.e.

Conversely assume that each vertex $V \in \{A, B, C\}$ is J_V^T-c.e. We show that the function mapping θ to T_θ is left-c.e. We know that it is left-c.e. on each J_V^T, and $\bigcup_{V\in\{A,B,C\}} J_V^T = \mathbb{R}$, but we must show how to merge the three algorithms. Let us assume that the origin of the Euclidean plane lies inside the triangle. If it is not the case, then one can translate the triangle by a rational vector, which preserves all the computability properties of T and its vertices.

If the origin is inside the triangle then for each vertex $V \in \{A, B, C\}$, if $J_V^T = [a, b]$ then there exists $\epsilon > 0$ such that the function V_θ is non-increasing on $(a-\epsilon, a+\epsilon)$ and non-decreasing on $(b-\epsilon, b+\epsilon)$, so by Lemma 2.4 there is a left-c.e. function $\hat{V}_\theta$ that coincides with V_θ on J_V^T, is non-increasing on $(a-\epsilon, a+\epsilon)$, non-decreasing on $(b-\epsilon, b+\epsilon)$ and $\hat{V}_\theta = +\infty$ for θ outside $(a-\epsilon, b+\epsilon)$. As a result, $T_\theta = \min\{A_\theta, B_\theta, C_\theta\} = \min\{\hat{A}_\theta, \hat{B}_\theta, \hat{C}_\theta\}$. ◀



So the semicomputability of the triangle can be decomposed in terms of the properties of the vertices treated separately, which leads us to investigate the properties of a single point.

3 Semicomputable point

The following is a generalization of left-c.e. and right-c.e. reals to points of the plane.

▶ **Definition 3.1.** A point A is ***semicomputable*** if there exist $\theta, \theta' \in \mathbb{Q}$ such that $\theta \neq \theta' \bmod \pi$ and A_θ and $A_{\theta'}$ are left-c.e.

Note that for a point, being semicomputable does not mean that the set $\{A\}$ is semicomputable. The latter is equivalent to saying that A is computable.

The vertices of a (non-degenerate) semicomputable triangle are necessarily semicomputable. We need tools to understand the directions in which the point is left-c.e.

▶ **Proposition 3.2.** *Let θ_1, θ_2 be computable such that $\theta_1 < \theta_2 < \theta_1 + \pi$. A point A is $[\theta_1, \theta_2]$-c.e. iff A_{θ_1} and A_{θ_2} are left-c.e.*

Proof. For $\theta \in [\theta_1, \theta_2]$, $A_\theta = \frac{\sin(\theta_2-\theta)}{\sin(\theta_2-\theta_1)} A_{\theta_1} + \frac{\sin(\theta-\theta_1)}{\sin(\theta_2-\theta_1)} A_{\theta_2} = \alpha(\theta)A_{\theta_1} + \beta(\theta)A_{\theta_2}$ where $\alpha(\theta)$ and $\beta(\theta)$ are nonnegative computable functions. ◀

▶ **Proposition 3.3.** *Let $I = [\alpha, \beta]$ with $\alpha < \beta$. A is I-c.e. iff A_θ is left-c.e. uniformly in $\theta \in (\alpha, \beta) \cap \mathbb{Q}$.*

Proof. The forward direction is straightforward. Let us prove the other direction. Assume that A_θ is left-c.e. uniformly in $\theta \in (\alpha, \beta) \cap \mathbb{Q}$. The function $\theta \mapsto A_\theta$ is L-Lipschitz for some L, so it is computable on the closure of $(\alpha, \beta) \cap \mathbb{Q}$, which is $[\alpha, \beta]$. Indeed, given $\theta \in [\alpha, \beta]$, take a sequence of rationals $\theta_i \in (\alpha, \beta)$ such that $|\theta - \theta_i| < 2^{-i}$, then $A_\theta = \sup_i A_{\theta_i} - L2^{-i}$. The sequence θ_i can be computed as follows: fix some rational $q \in (\alpha, \beta)$ and some k such that $\beta - q > 2^{-k}$ and $q - \alpha > 2^{-k}$, start from some rational sequence θ'_i such that $|\theta - \theta'_i| < 2^{-i}$ and define, for $i \geq k$, $\theta_i = \theta'_{i+1} + 2^{-i-1}$ if $\theta'_{i+1} \leq q$, $\theta_i = \theta'_{i+1} - 2^{-i-1}$ if $\theta'_{i+1} > q$, and $\theta_i = \theta_k$ for $i < k$. ◀

▶ **Definition 3.4.** Let A be a semicomputable point. Its ***semicomputability range*** I_A is defined as the union of the sets $[\alpha, \beta] \mod 2\pi$, for all $\alpha < \beta$ such that A is $[\alpha, \beta]$-c.e.

The range I_A is a connected subset of $\mathbb{R}/2\pi\mathbb{Z}$, i.e. is the set of equivalence classes of all the reals in an interval of $\mathbb{R}$. By abuse of notation we will often act as if I_A was a subset of $\mathbb{R}$. For instance if $\theta \in \mathbb{R}$ then when we write $\theta \in I_A$ we mean that the equivalence class of θ belongs to I_A. By $I_A = [\alpha, \beta]$ we mean that $I_a = [\alpha, \beta] \mod 2\pi$. By $\inf I_A$ we mean the equivalence class of $\inf I$ where $I \subseteq \mathbb{R}$ is any interval such that $I_A = I \mod 2\pi$.

The length of I_A is at most π, unless A is computable.

▶ **Proposition 3.5.** *A is computable $\iff$ $I_A = [0, 2\pi]$ $\iff$ $|I_A| > \pi$.*

Proof. We prove that if $|I_A| > \pi$ then A is computable, the other implications are obvious. Take $\theta, \theta' \in \mathbb{Q}$ such that $\theta, \theta + \pi, \theta', \theta' + \pi$ are pairwise distinct modulo 2π and all belong to I_A. One has $A_\theta = -A_{\theta+\pi}$ and $A_{\theta'} = -A_{\theta'+\pi}$ so all these numbers are computable, and the coordinates of A are linear combinations with computable coefficients of these numbers, so they are computable. ◀

For a computable angle θ, A_θ is left-c.e. $\iff$ $\theta \in I_A$. The uniformity in θ depends on whether the interval I_A is closed or open at each endpoint.

- If I_A is closed at an endpoint, A_θ is *uniformly* left-c.e. for θ around that endpoint,
- If I_A is open at an endpoint, A_θ is *non-uniformly* left-c.e. for θ around that endpoint,
- In particular, I_A is closed iff A is I_A-c.e. iff for $\theta \in I_A \cap \mathbb{Q}$, A_θ is left-c.e. *uniformly* in θ.

Using the last property and Definition 3.4, Theorem 2.3 can be reformulated as follows:

▶ **Corollary 3.6.** *A filled triangle $T = ABC$ is semicomputable iff each vertex $V \in \{A, B, C\}$ is semicomputable and $J_V^T \subseteq I_V$.*

In particular, if the filled triangle ABC is semicomputable then $|I_A| + |I_B| + |I_C| \geq 2\pi$ and $I_A \cup I_B \cup I_C = [0, 2\pi]$. This condition is not sufficient, as the intervals I_V must have the right orientations.

3.1 Semicomputable points and converging sequences

The intervals I for which a point A is I-c.e. are related to the regions containing computable sequences of points converging to A. However this is not an exact correspondence.

A two-dimensional cone with endpoint at A and delimited by the semi-lines starting at A with angles α, β, $\alpha \leq \beta < \alpha + \pi$, is denoted by $C(A, \alpha, \beta)$ and can be formally defined as $\{P \in \mathbb{R}^2 : (PA)_{\alpha - \pi/2} \geq 0 \text{ and } (PA)_{\beta + \pi/2} \geq 0\}$, where $(PA)_\theta = A_\theta - P_\theta$. Observe that this definition depends on the equivalence classes of α and β modulo 2π, so strictly speaking we do not need $\alpha \leq \beta < \alpha + \pi$ but $\alpha \leq \beta + 2k\pi < \alpha + \pi$ for some $k \in \mathbb{Z}$.

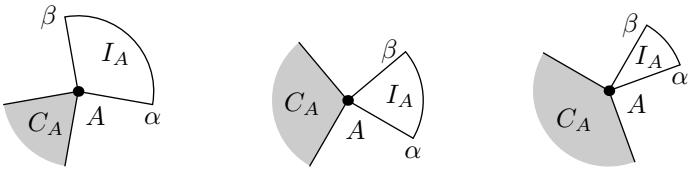


Figure 1 The semicomputability range (in white) and the Solovay cone (in gray) of the point $A = (x, y)$. (i) x, y are left-c.e., (ii) only x is left-c.e., (iii) x, y are not left-c.e.

▶ **Definition 3.7.** If A is semicomputable then we define its ***Solovay cone*** as $C_A = C(A, \beta + \pi/2, \alpha - \pi/2)$ where $\alpha = \inf I_A$ and $\beta = \sup I_A$.

The name will be explained in Section 4.

▶ **Proposition 3.8.** *C_A is the intersection of all the cones containing computable sequences converging to A.*

However there is not necessarily a computable sequence converging to A contained in C_A (an example will be given in Theorem 5.4).

▶ **Proposition 3.9.** *Let $I = [\alpha, \beta]$ with $\alpha < \beta \leq \alpha + \pi$ and A be I-c.e.*

- *If α left-c.e. and β right-c.e. then there exists a computable sequence A_i converging to A in the cone $C(A, \beta + \pi/2, \alpha - \pi/2)$.*
- *If α is $\emptyset'$-right-c.e. and β is $\emptyset'$-left-c.e. then there exists a computable sequence A_i converging to A and converging* to *the cone $C(A, \beta + \pi/2, \alpha - \pi/2)$, i.e., eventually contained in $C(A, \beta + \pi/2 - \epsilon, \alpha - \pi/2 + \epsilon)$ for every $\epsilon > 0$.*

We now identify the numbers α, β which can be endpoints of I_A for semicomputable A, when I_A is closed at these endpoints.

▶ **Theorem 3.10.** *For a real number α, the following are equivalent:*

- *α is $\emptyset'$-left-c.e.,*
- *$\alpha = \min I_A$ for some semicomputable point A.*

Symmetrically, β is $\emptyset'$-right-c.e. iff $\beta = \max I_A$ for some semicomputable point A.

4 Solovay derivatives

We have seen that the semicomputability of a triangle can be reduced to the semicomputability of its vertices and more precisely to their semicomputability ranges. Therefore we need tools to determine the range of a semicomputable point. This can be done using Solovay reducibility and its quantitative versions.

The coordinates of a semicomputable $A = (x, y)$ are d-c.e. and might not be either left-c.e. nor right-c.e. However, there is always a rotation with a rational angle mapping A to a semicomputable point $A' = (x', y')$ whose range $I_{A'}$ contains 0 i.e. such that x' is left-c.e. If $|I_A| > \pi/2$ then one can even take $I_{A'}$ containing 0 and $\pi/2$, i.e. one can take both x' and y' left-c.e. Hence in the study of semicomputable points one can restrict for simplicity to points (x, y) where x is left-c.e.

We first recall Solovay's notion of reduction between left-c.e. real numbers. We then define its quantitative version and study it. It has been independently introduced and studied in [1, 10], but the overlap is small.

4.1 Solovay derivatives

More on Solovay reducibility can be found in [11, 4]. It was originally defined for left-c.e. reals and has been extended to arbitrary reals in [15, 12].

Let $b_i \nearrow b$ denote that the sequence b_i is increasingly converging to b.

▶ **Definition 4.1.** Let b be left-c.e. We say that a is ***Solovay reducible*** to b if there exists a constant q and computable sequences $a_i \to a, b_i \nearrow b$ such that $|a - a_i| \leq q(b - b_i)$ for all i.

It is denoted by $a \leq_S b$. Equivalently, $a \leq_S b$ if there exists $q \in \mathbb{Q}$ such that $qb - a$ is left-c.e. and $-qb - a$ is right-c.e., which implies that a is d-c.e. We are interested in the optimal constants q and r such that $qb - a$ is left-c.e. and $rb - a$ is right-c.e.

Let b be left-c.e. If q is rational and $qb - a$ is left-c.e. then for every rational $q' > q$, $q'b - a$ is left-c.e. as well. In other words, the set $\{q \in \mathbb{Q} : qb - a \text{ is left-c.e.}\}$ is closed upwards. Similarly, the set $\{q \in \mathbb{Q} : qb - a \text{ is right-c.e.}\}$ is closed downwards. The following quantities have also been defined in [1].

▶ **Definition 4.2.** Let b be left-c.e. We define the ***upper and lower Solovay derivatives*** of a w.r.t. b as, respectively,

$$\overline{S}(a, b) = \inf\{q \in \mathbb{Q} : qb - a \text{ is left-c.e.}\},$$
$$\underline{S}(a, b) = \sup\{q \in \mathbb{Q} : qb - a \text{ is right-c.e.}\}.$$

The use of the word *derivative* will be justified in the sequel. By definition, $a \leq_S b \iff \overline{S}(a, b) < +\infty$ and $\underline{S}(b, a) > -\infty$. When $\underline{S}(a, b) = \overline{S}(a, b)$, we denote this value by $S(a, b)$. For instance it was proved in [1] and generalized in [10] that when b is Solovay complete $\underline{S}(a, b) = \overline{S}(a, b)$.

4.2 Basic properties

Here we investigate the possible values of $\underline{S}(a, b)$ and $\overline{S}(a, b)$ and their relationship. When a and b are both computable, $\underline{S}(a, b) = +\infty$ and $\overline{S}(a, b) = -\infty$.

▶ **Proposition 4.3.** *Let b be left-c.e. The following conditions are equivalent:*
1. $\overline{S}(a, b) < \underline{S}(a, b)$,
2. $\overline{S}(a, b) = -\infty$ *and* $\underline{S}(a, b) = +\infty$,
3. a, b *are computable.*

Proof. $3 \Rightarrow 2 \Rightarrow 1$ is direct. We prove $1 \Rightarrow 3$. If $\overline{S}(a, b) < \underline{S}(a, b)$ then for rationals $q < r$ in between, $qb - a$ is left-c.e. and $rb - a$ is right-c.e. which implies, by performing linear combinations, that a and b are computable. ◀

We consider this case as degenerate. In the other cases, i.e. when a, b are not both computable, one has $\underline{S}(a, b) \leq \overline{S}(a, b)$. The possible values of $(\underline{S}(a, b), \overline{S}(a, b))$ are:

	b computable	b left-c.e. not computable
a computable	$(+\infty, -\infty)$	$\underline{S}(a, b) = \overline{S}(a, b) = 0$
a left-c.e. not computable	$(+\infty, +\infty)$	$0 \leq \underline{S}(a, b) \leq \overline{S}(a, b)$
a right-c.e. not computable	$(-\infty, -\infty)$	$\underline{S}(a, b) \leq \overline{S}(a, b) \leq 0$
a d-c.e. not left/right-c.e.	$(-\infty, +\infty)$	$\underline{S}(a, b) \leq 0 \leq \overline{S}(a, b)$

The name "Solovay derivative" is partly justified by the next property which relates the quantities $\underline{S}(a, b)$ and $\overline{S}(a, b)$ to the difference quotient when approximating a and b computably. We will see later a strong connexion with the usual notion of derivative.

▶ **Proposition 4.4.** *Let a, b be d-c.e. and left-c.e. respectively, not both computable. If $a_i \to a$ and $b_i \nearrow b$ are computable sequences then*

$$\liminf \frac{a-a_i}{b-b_i} \leq \underline{S}(a,b) \leq \overline{S}(a,b) \leq \limsup \frac{a-a_i}{b-b_i}.$$

Proof. If $\limsup \frac{a-a_i}{b-b_i} < q$ then $a - a_i < q(b-b_i)$ for sufficiently large i, so $\overline{S}(a,b) \leq q$. Similarly, if $\liminf \frac{a-a_i}{b-b_i} > q$ then $a - a_i > q(b-b_i)$ for sufficiently large i, so $\underline{S}(a,b) \geq q$. ◀

In particular, if there are computable sequences $a_i \to a$ and $b_i \nearrow b$ such that $\frac{a-a_i}{b-b_i}$ has a limit s, then $\underline{S}(a,b) = \overline{S}(a,b) = s$.

▶ **Question 1.** *Are there always computable sequences $a_i \to a$ and $b_i \nearrow b$ such that*

$$\liminf \frac{a-a_i}{b-b_i} = \underline{S}(a,b) \leq \overline{S}(a,b) = \limsup \frac{a-a_i}{b-b_i}?$$

4.3 Calculation of the Solovay derivatives

We give formulas to derive the values of $\underline{S}(a,b)$ and $\overline{S}(a,b)$ in several situations.

▶ **Proposition 4.5** (Properties)**.**

1. *(Reflexivity) $\underline{S}(b,b) = \overline{S}(b,b) = 1$ if b is left-c.e. not computable.*
2. *When both a and b are left-c.e., one has $\underline{S}(a,b) = 1/\overline{S}(b,a)$.*
3. *(Transitivity) For all d-c.e. real a and left-c.e. reals b, c such that $a \leq_S b \leq_S c$,*
   - *If $\overline{S}(a,b) \geq 0$ then $\overline{S}(a,c) \leq \overline{S}(a,b)\overline{S}(b,c)$, otherwise $\overline{S}(a,c) \leq \overline{S}(a,b)\underline{S}(b,c)$.*
   - *If $\underline{S}(a,b) \geq 0$ then $\underline{S}(a,c) \geq \underline{S}(a,b)\underline{S}(b,c)$, otherwise $\underline{S}(a,c) \geq \underline{S}(a,b)\overline{S}(b,c)$.*
4. *In some cases we can also derive equalities. For all d-c.e. real a and left-c.e. reals b, c such that $a \leq_S b \leq_S c$ and $\underline{S}(a,b) = \overline{S}(a,b) =: S(a,b)$,*
   - *If $S(a,b) \geq 0$ then $\overline{S}(a,c) = S(a,b)\overline{S}(b,c)$ and $\underline{S}(a,c) = S(a,b)\underline{S}(b,c)$.*
   - *If $S(a,b) \leq 0$ then $\overline{S}(a,c) = S(a,b)\underline{S}(b,c)$ and $\underline{S}(a,c) = S(a,b)\overline{S}(b,c)$.*

4.3.1 Differentiation

The name Solovay derivative is justified by the following result, also obtained in [10] when b is Solovay complete.

▶ **Proposition 4.6.** *Let b be a non-computable left-c.e. real. If f is computable and differentiable at b then*

$$\underline{S}(f(b),b) = \overline{S}(f(b),b) = f'(b).$$

Proof. It is a direct application of Proposition 4.4. Let $b_i \nearrow b$ be a computable sequence. The sequence $f(b_i)$ is computable and $\lim(f(b) - f(b_i))/(b - b_i) = f'(b)$. ◀

It also implies that if f, g are computable, differentiable and $f'(b)$ and $g'(b)$ are positive then

$$S(f(a), g(b)) = \frac{f'(a)}{g'(b)} S(a,b).$$

This is proved by applying two times Proposition 4.5, item 4.

▶ **Example 4.7.** For instance, if b is not computable then $S(2b, b) = 2$ and $S(b^2, b) = 2b$ and $\overline{S}(\log(a), \log(b)) = b\overline{S}(a,b)/a$ and $\underline{S}(\log(a), \log(b)) = a\underline{S}(a,b)/a$.

In particular, for $a, b > 0$, $\overline{S}(a,b) = \frac{a}{b} \inf\{q \in \mathbb{Q} : \frac{b^q}{a} \text{ is left-c.e.}\}$.

Proposition 4.6 can be extended to bivariate differentiable functions.

▶ **Theorem 4.8.** *Let $f : \mathbb{R}^2 \to \mathbb{R}$ be totally differentiable and computable. Let y be left-c.e. and assume that x, y are not both computable.*

- *If* $\frac{\partial f}{\partial x}(x,y) > 0$, *then* $\begin{cases} \underline{S}(f(x,y),y) = \underline{S}(x,y)\frac{\partial f}{\partial x}(x,y) + \frac{\partial f}{\partial y}(x,y), \\ \overline{S}(f(x,y),y) = \overline{S}(x,y)\frac{\partial f}{\partial x}(x,y) + \frac{\partial f}{\partial y}(x,y). \end{cases}$
- *If* $\frac{\partial f}{\partial x}(x,y) < 0$, *then* $\begin{cases} \underline{S}(f(x,y),y) = \overline{S}(x,y)\frac{\partial f}{\partial x}(x,y) + \frac{\partial f}{\partial y}(x,y), \\ \overline{S}(f(x,y),y) = \underline{S}(x,y)\frac{\partial f}{\partial x}(x,y) + \frac{\partial f}{\partial y}(x,y). \end{cases}$
- *If* $\frac{\partial f}{\partial x}(x,y) = 0$ *and* $x \leq_S y$ *then* $\underline{S}(f(x,y),y) = \overline{S}(f(x,y),y) = \frac{\partial f}{\partial y}(x,y)$.

In particular, if $\frac{\partial f}{\partial x}(x,y) \neq 0$ then $f(x,y) \leq_{\mathrm{S}} y$ implies $x \leq_{\mathrm{S}} y$.

▶ Remark. In the remaining case where $\frac{\partial f}{\partial x}(x,y) = 0$ and $x \not\leq_{\mathrm{S}} y$, the values of $\underline{S}(f(x,y),y)$ and $\overline{S}(f(x,y),y)$ cannot be expressed in terms of $\underline{S}(x,y)$, $\overline{S}(x,y)$ and $\frac{\partial f}{\partial y}(x,y)$ only.

▶ **Example 4.9.**

- One has $\overline{S}(a+b,a) = 1 + \overline{S}(b,a)$, $\underline{S}(a+b,a) = 1 + \underline{S}(b,a)$.
- One has $\overline{S}(ab,a) = b + a\overline{S}(ab,a)$ and $\underline{S}(ab,a) = b + a\underline{S}(ab,a)$.

4.4 Back to semicomputable points

We now relate the semicomputability range of a point $A = (x,y)$ to the quantities $\overline{S}(y,x)$ and $\underline{S}(y,x)$, when x is left-c.e.

▶ **Proposition 4.10.** *Let $A = (x,y)$ be semicomputable but not computable with x left-c.e. and let $\alpha = \inf I_A$ and $\beta = \sup I_A$. One has $-\pi \leq \alpha \leq 0 \leq \beta \leq \pi$ and*

$$\alpha = \arctan(\overline{S}(y,x)) - \pi/2 \qquad \overline{S}(y,x) = \tan(\alpha + \pi/2)$$
$$\beta = \arctan(\underline{S}(y,x)) + \pi/2 \qquad \underline{S}(y,x) = \tan(\beta - \pi/2).$$

The functions tan and arctan are understood as functions between $[-\pi/2, \pi/2]$ and $[-\infty, +\infty]$.

Therefore the slopes of the Solovay cone C_A are $\underline{S}(y,x)$ and $\overline{S}(y,x)$, which explains the name of the cone.

We now give examples of semicomputable points and calculate their ranges. Let $A = (x,y)$ with x left-c.e.

- If x, y are Solovay incomparable left-c.e. reals then $\underline{S}(y,x) = 0$ and $\overline{S}(y,x) = +\infty$. The point $A = (x,y)$ is semicomputable with $I_A = [0, \pi/2]$.
- Let $x = \Omega$ be some Solovay complete left-c.e. real.
  - If y is left-c.e. incomplete then $S(y,\Omega) = 0$ and $I_A = (-\pi/2, \pi/2]$,
  - If y is right-c.e. incomplete then $S(y,\Omega) = 0$ and $I_A = [-\pi/2, \pi/2)$,
  - If y is d-c.e., neither left-c.e. nor right-c.e. then $S(y,\Omega) = 0$ and $I_A = (-\pi/2, \pi/2)$,
  - If $y = \Omega$ then $S(y,\Omega) = 1$ and $I_A = [-\pi/4, \pi/4]$.
- Let $y = f(x)$ where $f : \mathbb{R} \to \mathbb{R}$ is such that f' is computable and monotonic. One has $S(y,x) = f'(x)$ and $I_A = [\arctan(f'(x)) - \pi/2, \arctan(f'(x)) + \pi/2]$.

It is proved in [10] that every $\emptyset'$-computable (or Δ^0_2) number can be obtained as $S(b,\Omega)$ for some d-c.e. b and Solovay complete Ω. The proof of Theorem 3.10 shows that every $\emptyset'$-left-c.e. can be obtained this way, and symmetrically every $\emptyset'$-right-c.e. hence every $\emptyset'$-d-c.e. It gives a partial answer to Question 2.7 in [10].

▶ **Question 2.** *Can every Δ^0_3 real be obtained as $S(b,\Omega)$ for some left-c.e./d-c.e. real b?*

5 Generic triangles

All the classical parameters (like the angles or the coordinates of the centroid) of a semicomputable triangle are d-c.e. numbers, because the function mapping a triangle to a parameter is computable and Lipschitz. Some of them, like the sides lengths, the area or the perimeter, are always right-c.e.

In this section we show that these upper bounds on the effectiveness of the parameters are optimal. To do this we prove the existence of semicomputable triangles with prescribed properties. However instead of building them explicitly we use the existence of semicomputable triangles that are *generic* in some sense, and then investigate the properties of such triangles. We first give the minimal material needed, taken from [7].

▶ **Definition 5.1.** Let X be an effective Polish space and $A \subseteq X$. A point $x \in A$ is ***generic inside*** A if for every effective open set $U \subseteq X$, either $x \in U$ or there exists a neighborhood B of x such that $B \cap U \cap A = \emptyset$.

▶ **Example 5.2.**

- Taking $A = X$, being generic inside X amounts to being 1-generic,
- Every x is obviously generic inside $\{x\}$,
- In the space of real numbers with the Euclidean topology, a real number $x \in (0, 1)$ is ***right-generic*** if x is generic inside $[x, 1]$,
- The space of filled triangles is a subspace of the space of non-empty compact subsets of $\mathbb{R}^2$ with the Hausdorff metric and is an effective Polish space. A triangle T is ***inner-generic*** if it is generic inside $S(T) := \{T' \in \mathcal{T}, T' \subseteq T\}$. In other words, for every effective open set $\mathcal{U} \subseteq \mathcal{T}$, if T contains arbitrarily close (in the Hausdorff metric) triangles $T' \in \mathcal{U}$, then $T \in \mathcal{U}$.

The latter two examples are particular instances of the following general situation.

If τ' is a weaker topology on X then we define $S(x)$ as the closure of x in the topology τ', which is the intersection of the τ'-open sets containing x. Equivalently, $S(x) = \{y \in X : x \leq_{\tau'} y\}$ where $\leq_{\tau'}$ is the specialization pre-order defined by $x \leq y$ iff every τ'-neighborhood of x contains y.

▶ **Theorem 5.3** (Theorem 4.1.1 in [7]). *Let (X, τ) be an effective Polish space and τ' an effectively weaker topology, such that emptiness of finite intersections of basic open sets in τ, τ' is decidable. There exists a point x that is computable in (X, τ') and generic inside $S(x)$.*

For instance, $\mathbb{R}$ with the Euclidean topology is effective Polish, the topology τ' generated by the semi-lines $(q, +\infty)$ is effectively weaker, and its specialization pre-order is the natural ordering $\leq$ on $\mathbb{R}$. Theorem 5.3 implies the existence of right-generic left-c.e. reals.

In the effective Polish space $\mathcal{T}$ of filled triangles, we take the topology τ' generated by the following open sets: given a finite union U of open metric balls in $\mathbb{R}^2$, the set of triangles contained in U is a basic open set of the topology τ'. The specialization ordering is the reversed inclusion. Theorem 5.3 implies the existence of inner-generic semicomputable triangles.

Now we have the tools to prove the main result of this section.

▶ **Theorem 5.4.** *Let $T = ABC$ be an inner-generic semicomputable triangle.*

- *Each vertex A, B, C is generic inside T,*
- *For each vertex $V \in \{A, B, C\}$, $I_V = J_V^T$,*
- *For each vertex V, there is no computable sequence converging to V in the cone C_V,*

- *The slopes of the sides of T are 1-generic d-c.e. reals,*
- *The angles of T are 1-generic d-c.e. reals,*
- *A is not computable relative to the pair (B, C) (idem for B and C),*
- *The area of T is a left-generic right-c.e. real,*
- *The centroid of T is a 1-generic point with d-c.e. coordinates.*

This list could of course be extended *ad nauseam*.

6 Parametrizations

In the one-dimensional case, there is a simple parametrization of the semicomputable compact convex subsets of $\mathbb{R}$: they are exactly the closed intervals $[a, b]$ where a is left-c.e. and b is right-c.e. Apart from the fact that $a \leq b$, the two parameters a and b are independent. In this section we investigate the possibility of having a similar parametrization for classes of semicomputable compact convex subsets of $\mathbb{R}^2$, for instance the filled triangles. We show that for some definition of parametrization, no finite parametrization is possible.

A ***numbered set*** is a pair $\mathcal{S} = (S, \nu)$ where S is a countable set and $\nu_S : \mathrm{dom}(\nu) \subseteq \mathbb{N} \to S$ is surjective. If $\mathcal{S} = (S, \nu)$ is a numbered set then each $T \subseteq S$ has a canonical numbering, given by the restriction of ν to $\nu^{-1}(T)$. A ***morphism*** from $\mathcal{S} = (S, \nu)$ to $\mathcal{S}' = (S', \nu')$ is a function $\phi : S \to S'$ such that there exists a computable function $\varphi : \mathrm{dom}(\nu) \to \mathrm{dom}(\nu')$ such that $\nu' \circ \varphi = \phi \circ \nu$.

▶ **Definition 6.1.** Let $\mathcal{S} = (S, \nu_S)$ and $\mathcal{P} = (P, \nu_P)$ be numbered sets. A ***$\mathcal{P}$-parametrization*** of $\mathcal{S}$ is an isomorphism between $\mathcal{S}$ and a subset of $\mathcal{P}$.

We are interested in the case where $\mathcal{S}$ is the class of semicomputable triangles and $\mathcal{P} = \mathbb{R}^d_{\mathrm{lce}}$ is the class of vectors of d left-c.e. numbers, both with their canonical numberings. Proposition 2.2 implies the existence of a $\mathbb{R}^{\mathbb{N}}_{\mathrm{lce}}$-parametrization of the semicomputable filled triangles, i.e. that each such triangle T can be represented by a sequence of uniformly left-c.e. real numbers T_{θ_i}, where $(\theta_i)_{i\in\mathbb{N}}$ is a canonical enumeration of the rational numbers. We prove that no finite parametrization exists.

▶ **Theorem 6.2.** *For each $d \in \mathbb{N}$, there is no $\mathbb{R}^d_{\mathrm{lce}}$-parametrization of the semicomputable filled triangles.*

Proof. We first observe that an isomorphism between $\mathcal{T}$ and a subset of $\mathbb{R}^d_{\mathrm{lce}}$ would be order-preserving in both directions, where $\mathcal{T}$ is endowed with the reverse inclusion $\supseteq$ and $\mathbb{R}^d_{\mathrm{lce}}$ with the component-wise natural ordering $\leq$. This is a consequence of the generalization of the Myhill-Shepherdson theorem to effective continuous directed complete partial orders (dcpo's) [14]. It would imply that $(\mathcal{T}, \supseteq)$ embeds in $(\mathbb{R}^d, \leq)$, which we show is not possible. For this we use the order-theoretic notion of dimension and show that $(\mathcal{T}, \supseteq)$ is infinite-dimensional, while $(\mathbb{R}^d, \leq)$ is d-dimensional.

All the details about the dimension of partially ordered sets can be found in [13], we only give the key notions. A partially ordered set (poset) $(P, \leq)$ has dimension k if there exist k linear extensions of $\leq$ whose intersection is $\leq$, and k is minimal with this property. The standard n-dimensional ordering is $S_n = \{a_1, \ldots, a_n, A_1, \ldots, A_n\}$ with $a_i < A_j$ if $i \neq j$. If a poset $(P, \leq_P)$ embeds into a poset $(Q, \leq_Q)$ then the dimension of $(P, \leq_P)$ is no more than the dimension of $(Q, \leq_Q)$. The poset $(\mathbb{R}^d, \leq)$ has dimension d and we show that $(\mathcal{T}, \supseteq)$ is not finite-dimensional by embedding the standard ordering S_d into $(\mathcal{T}, \supseteq)$, for each $d \in \mathbb{N}$.

For each i, $a_i \in S_d$ is mapped to a large triangle t_i and $A_i \in S_d$ is mapped to a small triangle T_i such that $t_i \supseteq T_j \iff i \neq j$. This is achieved by starting from a regular polygon

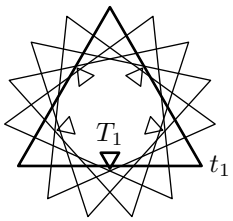


Figure 2 Embedding the standard 5-dimensional ordering in the poset of triangles. Note that T_1 is *not* contained in t_1.

with d vertices $v_1, \dots, v_d$, taking for each i a large triangle t_i containing all the vertices except v_i, and a small triangle T_i containing v_i. We simply show a picture for $d = 5$, but it can be generalized to any $d \in \mathbb{N}$. ◀

One could relax the notion of parametrization in different ways:

- If one requires a morphism from a subset of $\mathbb{R}^d_{\text{lce}}$ onto $\mathcal{T}$ then there is a $\mathbb{R}^2_{\text{lce}}$-parametrization, essentially because all the elements of the anti-diagonal of $\mathbb{R}^d_{\text{lce}}$ are pairwise incomparable.
- If one requires a one-to-one morphism from $\mathcal{T}$ to $\mathbb{R}^d_{\text{lce}}$ then there is a $\mathbb{R}_{\text{lce}}$-parametrization because $\mathcal{T}$ embeds in $\mathbb{R}^{\mathbb{N}}_{\text{lce}}$ and there is a one-to-one morphism from $\mathbb{R}^{\mathbb{N}}_{\text{lce}}$ to $\mathbb{R}_{\text{lce}}$.

In both cases, the parametrizations are not satisfactory because they are not geometrically meaningful. Other variations on the definition of parametrizations should be investigated.

The argument in the proof of Theorem 6.2 is actually very general and can be extended to many classes of sets.

▶ **Theorem 6.3.** *Let $\mathcal{F}$ be a class of compact semicomputable subsets of $\mathbb{R}^2$ that contains a set with non-empty interior and is closed under translations, scaling and rotations with rational parameters. There is no $\mathbb{R}^d$-parametrization of $\mathcal{F}$ for any $d \in \mathbb{N}$.*

Proof. We embed the standard d-dimensional ordering in $(\mathcal{F}, \supseteq)$.

Let $S \in \mathcal{F}$ be a set with non-empty interior. There exists a closed ball $\overline{B}(c, r)$ contained in S and intersecting the boundary ∂S of S in exactly one point. Indeed, take c_0 in the interior of S and $r_0 = d(c_0, \partial S)$. $\overline{B}(c_0, r_0)$ is contained in S and intersects ∂S in at least one point p. Let $c = (c_0 + p)/2$ and $r = r_0/2$. One easily checks that $\overline{B}(c, r)$ intersects ∂S in exactly one point.

Given $d \in \mathbb{N}$, let $(S_i)_{1 \leq i \leq d}$ be d distinct copies of S, rotated around c. The disk $\overline{B}(c, r)$ is contained in each S_i and intersects its boundary in exactly one point p_i. Therefore, for $i \neq j$, p_i belongs to the interior of S_j. For each i, let s_i be a small scaled copy of S containing p_i in its interior. As $p_i \in \partial S_i$, s_i is not contained in S_i. One can take s_i sufficiently small so that it is contained in each S_j, $j \neq i$. The family of sets S_i and s_i is an embedding of the standard d-dimensional ordering in $\mathcal{F}$. ◀

References

1. George Barmpalias and Andrew Lewis-Pye. Differences of halting probabilities. *J. Comput. Syst. Sci.*, 89:349–360, 2017.
2. Mark Braverman and Michael Yampolsky. *Computability of Julia Sets.* Springer, 2008.
3. Konrad Burnik and Zvonko Iljazovic. Computability of 1-manifolds. *Logical Methods in Computer Science*, 10(2), 2014.
4. Rod Downey and Denis Hirschfeldt. *Algorithmic Randomness and Complexity.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2010.

5 Daniel Graça, Cristóbal Rojas, and Ning Zhong. Computing geometric Lorenz attractors with arbitrary precision. To appear in Transactions of the American Mathematical Society, 2017.

6 Peter Hertling. Is the Mandelbrot set computable? *Math. Log. Q.*, 51(1):5–18, 2005.

7 Mathieu Hoyrup. Genericity of weakly computable objects. *Theory Comput. Syst.*, 60(3):396–420, 2017.

8 Zvonko Iljazovic. Co-c.e. spheres and cells in computable metric spaces. *Logical Methods in Computer Science*, 7(3), 2011.

9 Joseph S. Miller. Effectiveness for embedded spheres and balls. *Electronic Notes in Theoretical Computer Science*, 66(1):127–138, 2002. CCA 2002, Computability and Complexity in Analysis (ICALP 2002 Satellite Workshop).

10 Joseph S. Miller. On work of Barmpalias and Lewis-Pye: A derivation on the D.C.E. reals. In Adam R. Day, Michael R. Fellows, Noam Greenberg, Bakhadyr Khoussainov, Alexander G. Melnikov, and Frances A. Rosamond, editors, *Computability and Complexity - Essays Dedicated to Rodney G. Downey on the Occasion of His 60th Birthday*, volume 10010 of *Lecture Notes in Computer Science*, pages 644–659. Springer, 2017.

11 André Nies. *Computability and randomness*. Oxford logic guides. Oxford University Press, 2009.

12 Robert Rettinger and Xizhong Zheng. Solovay reducibility on d-c.e real numbers. In Lusheng Wang, editor, *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, volume 3595 of *Lecture Notes in Computer Science*, pages 359–368. Springer, 2005.

13 B. Schroeder. *Ordered Sets: An Introduction*. Birkhäuser Boston, 2003.

14 Klaus Weihrauch and T. Deil. Berechenbarkeit auf cpo-s. *Informatik-Berichte*, 63, 1980.

15 Xizhong Zheng and Robert Rettinger. On the extensions of Solovay-reducibility. In Kyung-Yong Chwa and J. Ian Munro, editors, *Computing and Combinatorics, 10th Annual International Conference, COCOON 2004, Jeju Island, Korea, August 17-20, 2004, Proceedings*, volume 3106 of *Lecture Notes in Computer Science*, pages 360–369. Springer, 2004.

On Computing the Total Variation Distance of Hidden Markov Models

Stefan Kiefer
University of Oxford, United Kingdom

Abstract

We prove results on the decidability and complexity of computing the total variation distance (equivalently, the L_1-distance) of hidden Markov models (equivalently, labelled Markov chains). This distance measures the difference between the distributions on words that two hidden Markov models induce. The main results are: (1) it is undecidable whether the distance is greater than a given threshold; (2) approximation is #P-hard and in PSPACE.

2012 ACM Subject Classification Theory of computation → Probabilistic computation, Theory of computation → Random walks and Markov chains

Keywords and phrases Labelled Markov Chains, Hidden Markov Models, Distance, Decidability, Complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.130

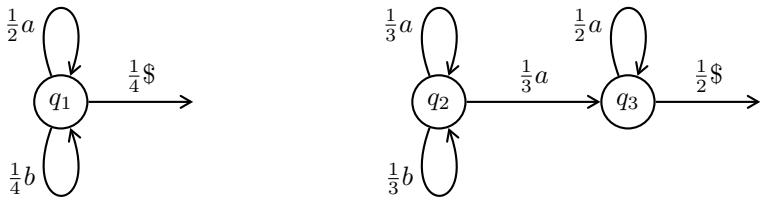
Related Version [15], https://arxiv.org/abs/1804.06170

Funding The author is supported by a Royal Society University Research Fellowship.

Acknowledgements The author thanks anonymous referees for their helpful comments.

1 Introduction

A (discrete-time, finite-state, finite-word) *labelled Markov chain (LMC)* (often called *hidden Markov model*) has a finite set Q of states and for each state a probability distribution over its outgoing transitions. Each outgoing transition is labelled with a letter from an alphabet Σ and leads to a target state, or is labelled with an end-of-word symbol \$. Here are two LMCs:



The LMC starts in a given initial state (or in a random state according to a given initial distribution), picks a random transition according to the state's distribution over the outgoing transitions, outputs the transition label, moves to the target state, and repeats until the end-of-word label \$ is emitted. This induces a probability distribution over finite words (excluding the end-of-word label \$). In the example above, if q_1 and q_2 are the initial states then the LMCs induce distributions π_1, π_2 with $\pi_1(aa) = \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{4}$ and $\pi_2(aa) = \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{1}{2} + \frac{1}{3} \cdot \frac{1}{2} \cdot \frac{1}{2}$. LMCs are widely employed in fields such as speech recognition (see [23] for a tutorial), gesture recognition [4], signal processing [8], and climate modeling [1]. LMCs are heavily used in computational biology [12], more specifically in DNA modeling [6] and biological

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 130; pp. 130:1–130:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

sequence analysis [11], including protein structure prediction [17] and gene finding [2]. In computer-aided verification, LMCs are the most fundamental model for probabilistic systems; model-checking tools such as Prism [18] or Storm [9] are based on analyzing LMCs efficiently.

A fundamental yet non-trivial question about LMCs is whether two LMCs generate the same distribution on words. This problem itself has applications in verification [16] and can be solved in polynomial time using algorithms that are based on linear algebra [24, 22, 7]. If two such distributions are not equal, one may ask how different they are. There exist various *distances* between discrete distributions, see, e.g., [7, Section 3]. One of them is the total variation distance (in the following just called *distance*), which can be defined by $d(\pi_1, \pi_2) = \max_{W \subseteq \Sigma^*} |\pi_1(W) - \pi_2(W)|$ in the case of LMCs. That is, $d(\pi_1, \pi_2)$ is the largest possible difference between probabilities that π_1 and π_2 assign to the same set of words. This distance is, up to a factor 2, equal to the L_1-norm of the difference between π_1 and π_2, i.e., $2d(\pi_1, \pi_2) = \sum_{w \in \Sigma^*} |\pi_1(w) - \pi_2(w)|$. Clearly, π_1 and π_2 are equal if and only if their distance is 0.

It is immediate from the definition of the distance that if L is a family of LMCs whose pairwise distances are bounded by $b \geq 0$ then for any event $W \subseteq \Sigma^*$ and any two LMCs $\mathcal{M}_1, \mathcal{M}_2 \in L$ we have $|\pi_1(W) - \pi_2(W)| \leq b$. From a verification point of view, this means that one needs to model check only one LMC in the family to obtain an approximation within b for the probabilities that the LMCs satisfy a given property W. Therefore, computing or approximating the distance can make model checking more efficient. It is shown in [3] that the *bisimilarity pseudometric* defined in [10] is an upper bound on the total variation distance and can be computed in polynomial time. The bisimilarity pseudometric has more direct bearings on *branching-time* system properties, which, in addition to emitted labels, take LMC states into account (not considered in this paper).

The problem of computing the distance was first studied in [20]: they show that computing the distance is NP-hard. In [7] it was shown that even approximating the distance within an $\varepsilon > 0$ given in binary is NP-hard. In this paper we improve these results. We show that it is undecidable whether the distance is greater than a given threshold. Further we show that approximating the distance is #P-hard and in PSPACE. The #P-hardness construction is relatively simple, perhaps simpler than the construction underlying the NP-hardness result in [7]. In contrast, our PSPACE algorithm requires a combination of special techniques: rounding-error analysis in floating-point arithmetic and Ladner's result [19] on counting in polynomial space.

2 Preliminaries

Let Q be a finite set. We view elements of $\mathbb{R}^Q$ as *vectors*, more specifically as row vectors. We write $\mathbf{1}$ for the all-1 vector, i.e., the element of $\{1\}^Q$. For a vector $\mu \in \mathbb{R}^Q$, we denote by $\mu^\top$ its transpose, a column vector. A vector $\mu \in [0,1]^Q$ is a *distribution over* Q if $\mu \mathbf{1}^\top = 1$. For $q \in Q$ we write δ_q for the (*Dirac*) distribution over Q with $\delta_q(q) = 1$ and $\delta_q(r) = 0$ for $r \in Q \setminus \{q\}$. We view elements of $\mathbb{R}^{Q \times Q}$ as *matrices*. A matrix $M \in [0,1]^{Q \times Q}$ is called *stochastic* if each row sums up to one, i.e., $M\mathbf{1}^\top = \mathbf{1}^\top$.

▶ **Definition 1.** A *labelled (discrete-time, finite-state, finite-word) Markov chain (LMC)* is a quadruple $\mathcal{M} = (Q, \Sigma, M, \eta)$ where Q is a finite set of states, Σ is a finite alphabet of labels, the mapping $M : \Sigma \to [0,1]^{Q \times Q}$ specifies the transitions, and $\eta \in [0,1]^Q$, with $\eta^\top + \sum_{a \in \Sigma} M(a)\mathbf{1}^\top = \mathbf{1}^\top$, specifies the end-of-word probability of each state.

Intuitively, if the LMC is in state q, then with probability $M(a)(q, q')$ it emits a and moves to state q', and with probability $\eta(q)$ it stops emitting labels. For the complexity results

in this paper, we assume that all numbers in η and in the matrices $M(a)$ for $a \in \Sigma$ are rationals given as fractions of integers represented in binary. We extend M to the mapping $M : \Sigma^* \to [0,1]^{Q \times Q}$ with $M(a_1 \cdots a_k) = M(a_1) \cdots M(a_k)$ for $a_1, \ldots, a_k \in \Sigma$. Intuitively, if the LMC is in state q then with probability $M(w)(q,q')$ it emits the word $w \in \Sigma^*$ and moves (in $|w|$ steps) to state q'. We require that each state of an LMC have a positive-probability path to some state q with $\eta(q) > 0$.

Fix an LMC $\mathcal{M} = (Q, \Sigma, M, \eta)$ for the rest of this section. To an (initial) distribution π over Q we associate the discrete probability space $(\Sigma^*, 2^{\Sigma^*}, \Pr_\pi)$ with $\Pr_\pi(w) := \Pr_\pi(\{w\}) := \pi M(w) \eta^\top$. To avoid clutter and when confusion is unlikely, we may identify the distribution $\pi \in [0,1]^Q$ with its induced probability measure $\Pr_\pi$; i.e., for a word or set of words W we may write $\pi(W)$ instead of $\Pr_\pi(W)$.

Given two initial distributions π_1, π_2, the *(total variation) distance* between π_1 and π_2 is defined as follows:[1]

$$d(\pi_1, \pi_2) \;:=\; \sup_{W \subseteq \Sigma^*} |\pi_1(W) - \pi_2(W)| \,.$$

As $\pi_1(W) - \pi_2(W) = \pi_2(\Sigma^* \setminus W) - \pi_1(\Sigma^* \setminus W)$, we have $d(\pi_1, \pi_2) = \sup_{W \subseteq \Sigma^*} (\pi_1(W) - \pi_2(W))$. The following proposition follows from basic principles, see, e.g., [21, Lemma 11.1]. In particular, it says that the supremum is attained and the total variation distance is closely related to the L_1-distance:

▶ **Proposition 2.** *Let $\mathcal{M}$ be an LMC. For any two initial distributions π_1, π_2 we have:*

$$d(\pi_1, \pi_2) \;=\; \max_{W \subseteq \Sigma^*} (\pi_1(W) - \pi_2(W)) \;=\; \frac{1}{2} \sum_{w \in \Sigma^*} |\pi_1(w) - \pi_2(w)|$$

The maximum is attained by $W = \{w \in \Sigma^* : \pi_1(w) \geq \pi_2(w)\}$.

In view of this proposition, all complexity results on the (total variation) distance hold equally for the L_1-distance.

An LMC $\mathcal{M}$ is called *acyclic* if its transition graph is acyclic. Equivalently, $\mathcal{M}$ is acyclic if for all $q \in Q$ we have that $\Pr_{\delta_q}$ has finite support, i.e., $\{w \in \Sigma^* : \Pr_{\delta_q}(w) > 0\}$ is finite.

3 The Threshold-Distance Problem

In [20, Section 6] (see also [7, Theorem 7]), a reduction is given from the *clique* decision problem to show that computing the distance in LMCs is NP-hard. In that reduction the distance is rational and its bit size polynomial in the input. It was shown in [5, Proposition 12] that the distance d can be irrational. Define the *non-strict (resp. strict) threshold-distance* problem as follows: Given an LMC, two initial distributions π_1, π_2, and a threshold $\tau \in [0,1] \cap \mathbb{Q}$, decide whether $d(\pi_1, \pi_2) \geq \tau$ (resp. $d(\pi_1, \pi_2) > \tau$). In [5, Proposition 14] it was shown that the non-strict threshold-distance problem is NP-hard with respect to Turing reductions.

In the following two subsections we consider the threshold-distance problem for general and acyclic LMCs, respectively.

[1] One could analogously define the total variation distance between two LMCs $\mathcal{M}_1 = (Q_1, \Sigma, M_1, \eta_1)$ and $\mathcal{M}_2 = (Q_2, \Sigma, M_2, \eta_2)$ with initial distributions π_1 and π_2 over Q_1 and Q_2, respectively. Our definition is without loss of generality, as one can take the LMC $\mathcal{M} = (Q, \Sigma, M, \eta)$ where Q is the disjoint union of Q_1 and Q_2, and M, η are defined using M_1, M_2, η_1, η_2 in the straightforward manner.

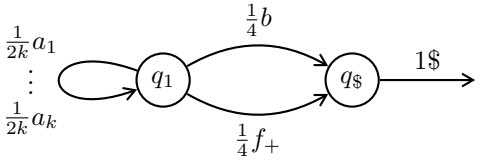
3.1 General LMCs

We show:

▶ **Theorem 3.** *The strict threshold-distance problem is undecidable.*

Proof. We reduce from the emptiness problem for probabilistic automata. A *probabilistic automaton* is a tuple $\mathcal{A} = (Q, \Sigma, M, \alpha, F)$ where Q is a finite set of states, Σ is a finite alphabet of labels, the mapping $M : \Sigma \to [0,1]^{Q\times Q}$, where $M(a)$ is a stochastic matrix for each $a \in \Sigma$, specifies the transitions, $\alpha \in [0,1]^Q$ is an initial distribution, and $F \subseteq Q$ is a set of accepting states. Extend M to $M : \Sigma^* \to [0,1]^{Q\times Q}$ as in the case of LMCs. In the case of a probabilistic automaton, $M(w)$ is a stochastic matrix for each $w \in \Sigma^*$. For each $w \in \Sigma^*$ define $\Pr_{\mathcal{A}}(w) := \alpha M(w)\eta^\top$ where $\eta \in \{0,1\}^Q$ denotes the characteristic vector of F. The probability $\Pr_{\mathcal{A}}(w)$ can be interpreted as the probability that $\mathcal{A}$ accepts w, i.e., the probability that after inputting w the automaton $\mathcal{A}$ is in an accepting state. The *emptiness problem* asks, given a probabilistic automaton $\mathcal{A}$, whether there is a word $w \in \Sigma^*$ such that $\Pr_{\mathcal{A}}(w) > \frac{1}{2}$. This problem is known to be undecidable [22, p. 190, Theorem 6.17].

In the following we assume $\Sigma = \{a_1, \ldots, a_k\}$. Given a probabilistic automaton $\mathcal{A}$ as above, construct an LMC $\mathcal{M} = (Q \cup \{q_1, q_\$\}, \Sigma \cup \{b, f_+, f_-\}, \overline{M}, \delta_{q_\$})$ such that $q_1, q_\$$ are fresh states, and b, f_+, f_- are fresh labels. The transitions originating in the fresh states $q_1, q_\$$ are as follows:

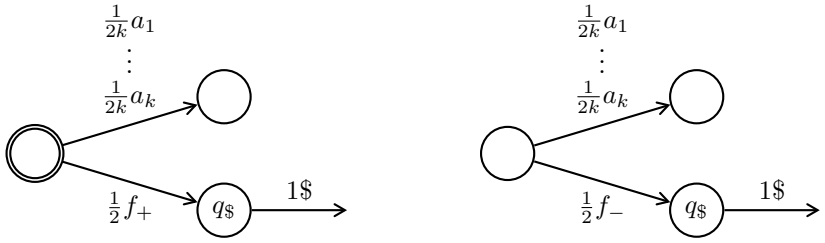


Here and in the subsequent pictures we use a convention that there be a state $q_\$$ with $\eta(q_\$) = 1$ and that $\eta(q) = 0$ hold for all other states.

Define $\pi_1 := \delta_{q_1}$. Then for all $w \in \Sigma^*$ we have:

$$\pi_1(wb) \;=\; \pi_1(wf_+) \;=\; \left(\frac{1}{2k}\right)^{|w|} \cdot \frac{1}{4} \tag{1}$$

The transitions originating in the states in Q are defined so that all $q \in Q$ emit each $a \in \Sigma$ with probability $\frac{1}{2k}$ (like q_1). For all $q \in F$ there is a transition to $q_\$$ labelled with $\frac{1}{2}$ and f_+; for all $q \in Q \setminus F$ there is a transition to $q_\$$ labelled with $\frac{1}{2}$ and f_-:



Formally, for $q, r \in Q$ and $a \in \Sigma$ set $\overline{M}(a)(q,r) := \frac{1}{2k} M(a)(q,r)$. For $q \in F$ set $\overline{M}(f_+)(q, q_\$) := \frac{1}{2}$, and for $q \in Q \setminus F$ set $\overline{M}(f_-)(q, q_\$) := \frac{1}{2}$. Define $\pi_2 := \alpha$ (in the natural way, i.e., with $\pi_2(q_1) = \pi_2(q_\$) = 0$). Then for all $w \in \Sigma^*$ we have:

$$\begin{aligned} \pi_2(wf_+) &= \left(\frac{1}{2k}\right)^{|w|} \cdot \Pr_{\mathcal{A}}(w) \cdot \frac{1}{2} \qquad \text{and} \\ \pi_2(wf_-) &= \left(\frac{1}{2k}\right)^{|w|} \cdot (1 - \Pr_{\mathcal{A}}(w)) \cdot \frac{1}{2} \end{aligned} \tag{2}$$

Consider $L := \Sigma^*\{b, f_+\}$. We have $\pi_1(L) = 1$. One can compute $\pi_2(L)$ in polynomial time by computing the probability of reaching a transition labelled by f_+ (the label b is not reachable). We claim that there is $w \in \Sigma^*$ with $\Pr_{\mathcal{A}}(w) > \frac{1}{2}$ if and only if $d(\pi_1, \pi_2) > \pi_1(L) - \pi_2(L)$. It remains to prove this claim.

Suppose there is no $w \in \Sigma^*$ with $\Pr_{\mathcal{A}}(w) > \frac{1}{2}$. Then, by (1) and (2), for all $w \in \Sigma^*$ we have $\pi_1(wf_+) \geq \pi_2(wf_+)$. Hence:

$$\{w \in (\Sigma \cup \{b, f_+, f_-\})^* : \pi_1(w) > 0,\ \pi_1(w) \geq \pi_2(w)\} \quad = \quad L$$

By Proposition 2 it follows $d(\pi_1, \pi_2) = \pi_1(L) - \pi_2(L)$.

Conversely, suppose there is $w \in \Sigma^*$ with $\Pr_{\mathcal{A}}(w) > \frac{1}{2}$. Consider $L' := L \setminus \{wf_+\}$. We have:

$$\begin{aligned} d(\pi_1, \pi_2) &\geq \pi_1(L') - \pi_2(L') && \text{Proposition 2} \\ &= \pi_1(L) - \pi_1(wf_+) - \pi_2(L) + \pi_2(wf_+) && \text{definition of } L' \\ &= \pi_1(L) - \pi_2(L) + \left(\frac{1}{2k}\right)^{|w|} \cdot \left(\frac{1}{2}\Pr_{\mathcal{A}}(w) - \frac{1}{4}\right) && \text{by (1) and (2)} \\ &> \pi_1(L) - \pi_2(L) && \Pr_{\mathcal{A}}(w) > \frac{1}{2} \end{aligned}$$

◀

Cortes, Mohri, and Rastogi [7] conjectured "that the problem of computing the [...] distance [...] is in fact undecidable", see the discussion after the proof of [7, Theorem 7]. Theorem 3 proves *one interpretation* of that conjecture. But the distance can be approximated with arbitrary precision, cf. Section 4, so the distance is "computable" in this sense.

In [5, Theorem 15] it was shown that there is a polynomial-time many-one reduction from the square-root-sum problem to the non-strict threshold-distance problem for LMCs. Decidability of the non-strict threshold-distance problem remains open.

3.2 Acyclic LMCs

It was shown in [20, Section 6] and [5, Proposition 14] that the non-strict threshold-distance problem is NP-hard with respect to Turing reductions, even for acyclic LMCs. We improve this result to PP-hardness:

▶ **Proposition 4.** *The non-strict and strict threshold-distance problems are PP-hard, even for acyclic LMCs and even with respect to many-one reductions.*

The proof uses the connection between PP and #P. Consider the problem #NFA, which is defined as follows: given a nondeterministic finite automaton (NFA) $\mathcal{A}$ over alphabet Σ, and a number $n \in \mathbb{N}$ in unary, compute $|L(\mathcal{A}) \cap \Sigma^n|$, i.e., the number of accepted words of length n. The problem #NFA is #P-complete [14]. The following lemma forms the core of the proof of Proposition 4:

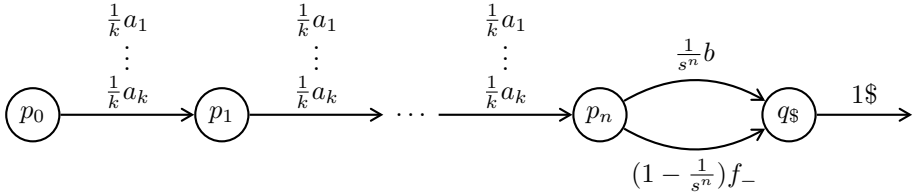
▶ **Lemma 5.** *Given an NFA $\mathcal{A} = (Q, \Sigma, \delta, q^{(1)}, F)$ and a number $n \in \mathbb{N}$ in unary, one can compute in polynomial time an acyclic LMC $\mathcal{M}$ and initial distributions π_1, π_2 and a rational number y such that*

$$d(\pi_1, \pi_2) \;=\; y + \frac{|\Sigma^n \setminus L(\mathcal{A})|}{|\Sigma|^n |Q|^n}\,.$$

Proof. In the following we assume $Q = \{q^{(1)}, \ldots, q^{(s)}\}$ and $\Sigma = \{a_1, \ldots, a_k\}$. Construct the acyclic LMC $\mathcal{M} = (Q', \Sigma \cup \{b, f_+, f_-\}, M)$ such that

$$Q' \;=\; \{p_0, p_1, \ldots, p_n, q_\$\} \;\cup\; \{q_i^{(j)} : 0 \le i \le n,\ 1 \le j \le s\} \;\cup\; \{r_i : 0 \le i \le n\}$$

and b, f_+, f_- are fresh labels. The transitions and end-of-word probabilities originating in the states $p_0, \ldots, p_n, q_\$$ are as follows:



Define $\pi_1 := \delta_{p_0}$. Then for all $w \in \Sigma^n$ we have:

$$\pi_1(wb) \;=\; \frac{1}{k^n} \cdot \frac{1}{s^n} \tag{3}$$

$$\pi_1(wf_-) \;=\; \frac{1}{k^n} \cdot \Big(1 - \frac{1}{s^n}\Big) \tag{4}$$

The transitions originating in the states $q_i^{(j)}, r_i$ are as follows. For each $a \in \Sigma$ and each $i \in \{0, \ldots, n-1\}$ set:

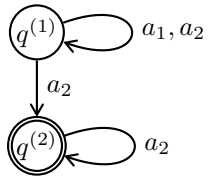
$$M(a)\big(q_i^{(j)}, q_{i+1}^{(j')}\big) \;:=\; \frac{1}{k} \cdot \frac{1}{s} \qquad \forall\, j \in \{1, \ldots, s\} \quad \forall\, q^{(j')} \in \delta(q^{(j)}, a)$$

$$M(a)\big(q_i^{(j)}, r_{i+1}\big) \;:=\; \frac{1}{k} \cdot \Big(1 - \frac{|\delta(q^{(j)}, a)|}{s}\Big) \qquad \forall\, j \in \{1, \ldots, s\}$$

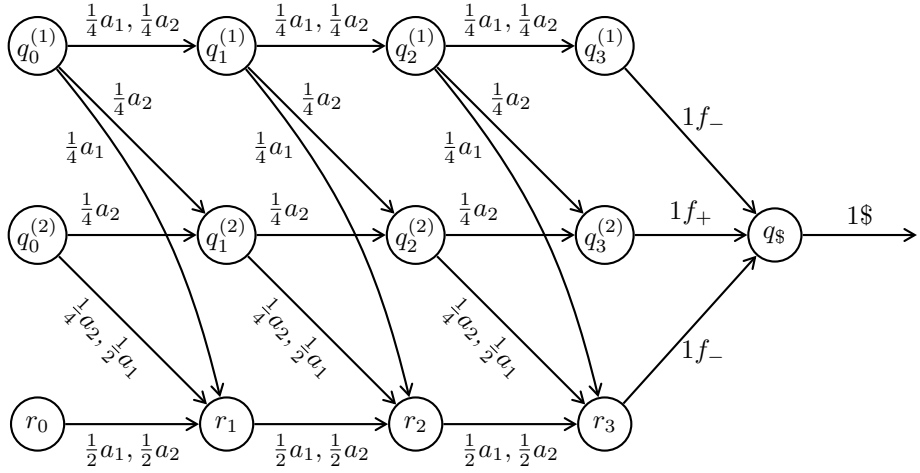
$$M(a)\big(r_i, r_{i+1}\big) \;:=\; \frac{1}{k}$$

Observe that if $i \in \{0, \ldots, n-1\}$ then r_i and all $q_i^{(j)}$ emit each $a \in \Sigma$ with probability $1/k$. For each $q^{(j)} \in F$ set $M(f_+)(q_n^{(j)}, q_\$) := 1$. For each $q^{(j)} \notin F$ set $M(f_-)(q_n^{(j)}, q_\$) := 1$. Finally, set $M(f_-)(r_n, q_\$) := 1$.

▶ **Example 6.** We illustrate this construction with the following NFA $\mathcal{A}$ over $\Sigma = \{a_1, a_2\}$:



For $n = 3$ we obtain the following transitions:



Define $\pi_2 := \delta_{q_0^{(1)}}$. For all $w \in \Sigma^*$ write $\#acc(w)$ for the number of accepting w-labelled runs of the automaton $\mathcal{A}$, i.e., the number of w-labelled paths from $q^{(1)}$ to a state in F. For all $w \in \Sigma^n$ we have:

$$\pi_2(wf_+) \;=\; \frac{1}{k^n} \cdot \frac{\#acc(w)}{s^n} \tag{5}$$

$$\pi_2(wf_-) \;=\; \frac{1}{k^n} \cdot \Big(1 - \frac{\#acc(w)}{s^n}\Big) \tag{6}$$

Define $B := \Sigma^n\{b, f_-\}$. By (3), (4) we have $\pi_1(B) = 1$. One can compute $\pi_2(B)$ in polynomial time by computing the probability of reaching a transition labelled by f_- (the label b is not reachable). Set $y := \pi_1(B) - \pi_2(B)$.

It follows from Proposition 2 that $d(\pi_1, \pi_2) = \pi_1(L) - \pi_2(L)$ holds for

$$L \;:=\; \{w \in (\Sigma \cup \{b, f_+, f_-\})^* : 0 < \pi_1(w) \geq \pi_2(w)\}\,.$$

Observe that $L(\mathcal{A}) = \{w \in \Sigma^n : \#acc(w) \geq 1\}$. Hence it follows with (3), (4), (6):

$$L \;=\; \Sigma^n\{b\} \;\cup\; (\Sigma^n \cap L(\mathcal{A}))\{f_-\}$$

Defining $\overline{L(\mathcal{A})} := \Sigma^n \setminus L(\mathcal{A})$ we can write:

$$L \;=\; B \setminus \big(\overline{L(\mathcal{A})}\{f_-\}\big)$$

Thus we have:

$$\begin{aligned} d(\pi_1, \pi_2) \;&=\; \pi_1\Big(B \setminus \big(\overline{L(\mathcal{A})}\{f_-\}\big)\Big) - \pi_2\Big(B \setminus \big(\overline{L(\mathcal{A})}\{f_-\}\big)\Big) && \text{as argued above} \\ &=\; y + \pi_2\big(\overline{L(\mathcal{A})}\{f_-\}\big) - \pi_1\big(\overline{L(\mathcal{A})}\{f_-\}\big) && \text{definition of } y \end{aligned}$$

Observe that $\overline{L(\mathcal{A})} = \{w \in \Sigma^n : \#acc(w) = 0\}$. Hence we can continue:

$$\begin{aligned} &=\; y + \frac{\big|\overline{L(\mathcal{A})}\big|}{k^n} - \frac{\big|\overline{L(\mathcal{A})}\big|}{k^n} \cdot \Big(1 - \frac{1}{s^n}\Big) && \text{by (6), (4)} \\ &=\; y + \frac{\big|\overline{L(\mathcal{A})}\big|}{k^n s^n} \;=\; y + \frac{|\Sigma^n \setminus L(\mathcal{A})|}{|\Sigma|^n |Q|^n} && \text{definitions} \end{aligned}$$

◀

The PP lower bound from Proposition 4 is tight for acyclic LMCs:

▶ **Theorem 7.** *The non-strict and strict threshold-distance problems are PP-complete for acyclic LMCs.*

▶ **Remark 8.** The works [20, 7] also consider the L_k-distances for integers k:

$$d_k(\pi_1, \pi_2) \;:=\; \sum_{w \in \Sigma^*} |\pi_1(w) - \pi_2(w)|^k$$

For any fixed even k one can compute d_k in polynomial time, see, e.g., [7, Theorem 6]. In contrast, it is NP-hard to compute or even approximate d_k for any odd k [7, Theorems 7 and 10]. Our PP- and #P-hardness results (Proposition 4 and Theorem 9) hold for d_1 (due to Proposition 2) but the reductions do not apply in an obvious way to d_k for any $k \geq 2$. However, the argument in the proof of Theorem 7 for the PP upper bound does generalize to all d_k, see [15].

4 Approximation

As the strict threshold-distance problem is undecidable (Theorem 3), one may ask whether the distance can be approximated. It is not hard to see that the answer is yes. In fact, it was shown in [5, Corollary 8] that the distance can be approximated within an arbitrary additive error even for *infinite-word* LMCs, but no complexity bounds were given. In this section we provide bounds on the complexity of approximating the distance for (finite-word) LMCs.

4.1 Hardness

Lemma 5 implies hardness of approximating the distance:

▶ **Theorem 9.** *Given an LMC and initial distributions π_1, π_2 and an error bound $\varepsilon > 0$ in binary, it is #P-hard to compute a number x with $|d(\pi_1, \pi_2) - x| \leq \varepsilon$, even for acyclic LMCs.*

Proof. Recall that the problem #NFA is #P-complete [14]. Let $\mathcal{A}$ be the given NFA and $n \in \mathbb{N}$. Let $\mathcal{M}, \pi_1, \pi_2, y$ be as in Lemma 5. Approximate $d(\pi_1, \pi_2)$ within $1/(3|\Sigma|^n|Q|^n)$ and call the approximation $\tilde{d}$. It follows from Lemma 5 that $|L(\mathcal{A}) \cap \Sigma^n|$ is the unique integer u with

$$\left| y + \frac{|\Sigma|^n - u}{|\Sigma|^n |Q|^n} - \tilde{d} \right| \;\leq\; \frac{1}{3|\Sigma|^n|Q|^n} \,.$$

Such u can be computed in polynomial time. ◀

Theorem 9 improves the NP-hardness result of [5, Proposition 9]. In fact, PP and #P are substantially harder than NP: By Toda's theorem [25], the polynomial-time hierarchy (PH) is contained in $\mathrm{P}^{\mathrm{PP}} = \mathrm{P}^{\#\mathrm{P}}$. Therefore, any problem in PH can be decided in deterministic polynomial time with the help of an oracle for the threshold-distance problem or for approximating the distance.

4.2 Acyclic LMCs

Towards approximation algorithms, define $W_2 := \{w \in \Sigma^* : \pi_1(w) \geq \pi_2(w)\}$ and $W_1 := \{w \in \Sigma^* : \pi_1(w) < \pi_2(w)\}$. By Proposition 2 we have:

$$d(\pi_1, \pi_2) \;=\; \pi_1(W_2) - \pi_2(W_2) \;=\; 1 - \pi_1(W_1) - \pi_2(W_2) \tag{7}$$

Therefore, to approximate $d(\pi_1, \pi_2)$ it suffices to approximate $\pi_i(W_i)$. A simple sampling scheme leads to the following theorem:

▶ **Theorem 10.** *There is a randomized algorithm, R, that, given an acyclic LMC $\mathcal{M}$ and initial distributions π_1, π_2 and an error bound $\varepsilon > 0$ and an error probability $\delta \in (0,1)$, does the following:*

- *R computes, with probability at least $1-\delta$, a number x with $|d(\pi_1,\pi_2) - x| \leq \varepsilon$;*
- *R runs in time polynomial in $\frac{\log \delta}{\varepsilon}$ and in the encoding size of $\mathcal{M}$ and π_1, π_2.*

Note that $\frac{1}{\varepsilon}$ is not polynomial in the bit size of ε, so combining Theorems 9 and 10 does not imply breakthroughs in computational complexity.

Proof. Let $i \in \{1,2\}$. The length of a longest word w with $\pi_i(w) > 0$ is polynomial in the encoding of the (acyclic) LMC $\mathcal{M}$. Thus, one can sample, in time polynomial in the encoding of $\mathcal{M}, \pi_1, \pi_2$, a word w according to $\Pr_{\pi_i}$; i.e., any w is sampled with probability $\pi_i(w)$. Similarly, one can check in polynomial time whether $w \in W_i$. If m samples are taken, the proportion, say $\hat{p}_i$, of samples w such that $w \in W_i$ is an estimation of $\pi_i(W_i)$. By Hoeffding's inequality, we have $|\hat{p}_i - \pi_i(W_i)| \geq \varepsilon/2$ with probability at most $2e^{-m\varepsilon^2/2}$. Choose $m \geq -\frac{2}{\varepsilon^2} \ln \frac{\delta}{4}$. It follows that $|\hat{p}_i - \pi_i(W_i)| > \varepsilon/2$ with probability at most $\delta/2$. Therefore, by (7), the algorithm that returns $1 - \hat{p}_1 - \hat{p}_2$ has the required properties. ◀

4.3 General LMCs

Finally we aim at an algorithm that approximates the distance within ε, for ε given in binary. By Theorem 9 such an algorithm cannot run in polynomial time unless P = PP. For LMCs that are not necessarily acyclic, words of polynomial length may have only small probability, so sampling approaches need to sample words of exponential length. Thus, a naive extension of the algorithm from Theorem 10 leads to a randomized exponential-time algorithm. We will develop a non-randomized PSPACE algorithm, resulting in the following theorem:

▶ **Theorem 11.** *Given an LMC, and initial distributions π_1, π_2, and an error bound $\varepsilon > 0$ in binary, one can compute in PSPACE a number x with $|d(\pi_1,\pi_2) - x| \leq \varepsilon$.*

The approximation algorithm combines special techniques. The starting point is again the expression for the distance in (7). The following lemma allows the algorithm to neglect words that are longer than exponential:

▶ **Lemma 12.** *Given an LMC, and initial distributions π_1, π_2, and a rational number $\lambda > 0$ in binary, one can compute in polynomial time a number $n \in \mathbb{N}$ in binary such that*

$$\pi_i(\Sigma^{>n}) \;\leq\; \lambda \qquad \text{for both } i \in \{1,2\}\,.$$

For n as in Lemma 12 and both $i \in \{1,2\}$, define $W_i' := W_i \cap \Sigma^{\leq n}$. By Lemma 12 it would suffice to approximate $\pi_i(W_i')$ for both i, as we have by (7):

$$\pi_1(W_1') + \pi_2(W_2') \quad\leq\quad 1 - d(\pi_1,\pi_2) \quad\leq\quad \pi_1(W_1') + \pi_2(W_2') + 2\lambda \tag{8}$$

However, it not obvious if $\pi_i(W_i')$ can be approximated efficiently, as for exponentially long words w it is hard to check if $w \in W_i'$ holds. Indeed, $\pi_i(w)$ may be very small and may have exponential bit size. The main trick of our algorithm will be to approximate $\pi_i(w)$ using floating-point arithmetic with small *relative* error, say $\widetilde{\pi}_i(w) \in [\pi_i(w)(1-\theta), \pi_i(w)(1+\theta)]$ for small $\theta > 0$. This allows us to approximate $\pi_1(W_1') + \pi_2(W_2')$ (crucially, not the two summands individually). Indeed, define approximations for W_1' and W_2' by

$$\widetilde{W_1} \;:=\; \{w \in \Sigma^{\leq n} : \widetilde{\pi}_1(w) < \widetilde{\pi}_2(w)\} \qquad \text{and} \qquad \widetilde{W_2} \;:=\; \{w \in \Sigma^{\leq n} : \widetilde{\pi}_1(w) \geq \widetilde{\pi}_2(w)\}\,.$$

Then we have:

$$\begin{aligned}
\pi_2(w) \;\le\; \pi_1(w) \;<\; \pi_2(w) + \theta\pi_1(w) + \theta\pi_2(w) \qquad & \text{for } w \in \widetilde{W}_1 \cap W_2' \\
\pi_1(w) \;<\; \pi_2(w) \;\le\; \pi_1(w) + \theta\pi_1(w) + \theta\pi_2(w) \qquad & \text{for } w \in \widetilde{W}_2 \cap W_1'
\end{aligned}$$

It follows:

$$\begin{aligned}
\pi_2(\widetilde{W}_1 \cap W_2') \;\le\; \pi_1(\widetilde{W}_1 \cap W_2') \;\le\; \pi_2(\widetilde{W}_1 \cap W_2') + 2\theta \\
\pi_1(\widetilde{W}_2 \cap W_1') \;\le\; \pi_2(\widetilde{W}_2 \cap W_1') \;\le\; \pi_1(\widetilde{W}_2 \cap W_1') + 2\theta
\end{aligned} \tag{9}$$

Hence we have:

$$\begin{aligned}
\pi_1(W_1') + \pi_2(W_2') \;&=\; \pi_1(\widetilde{W}_1 \cap W_1') + \pi_2(\widetilde{W}_1 \cap W_2') + \pi_2(\widetilde{W}_2 \cap W_2') + \pi_1(\widetilde{W}_2 \cap W_1') \\
&\overset{(9)}{\le}\; \pi_1(\widetilde{W}_1) + \pi_2(\widetilde{W}_2) \\
&\overset{(9)}{\le}\; \pi_1(W_1') + \pi_2(W_2') + 4\theta
\end{aligned}$$

By combining this with (8) we obtain:

$$\pi_1(\widetilde{W}_1) + \pi_2(\widetilde{W}_2) - 4\theta \quad\le\quad 1 - d(\pi_1, \pi_2) \quad\le\quad \pi_1(\widetilde{W}_1) + \pi_2(\widetilde{W}_2) + 2\lambda \tag{10}$$

It remains to tie two loose ends:

1. develop a PSPACE method to approximate $\pi_i(w)$ within *relative* error θ for any $\theta > 0$ in binary, where w is an at most exponentially long word (given on a special input tape);
2. based on this method, approximate $\pi_i(\widetilde{W}_i)$ in PSPACE.

For item 1 we use floating-point arithmetic, for item 2 we use Ladner's result [19] on counting in polynomial space.

For $k \in \mathbb{N}$, define $\mathbb{F}_k := \{m \cdot 2^z : z \in \mathbb{Z},\ 0 \le m \le 2^k - 1\}$, the set of *$k$-bit floating-point numbers*. For our purposes, nonnegative floating-point numbers suffice, and there is no need to bound the exponent z, as all occurring exponents will have polynomial bit size. We define rounding as usual: for $x \ge 0$ write $\langle x \rangle_k$ for the number in $\mathbb{F}_k$ that is nearest to x (break ties in an arbitrary but deterministic way). Then there is δ with $\langle x \rangle_k = x \cdot (1 + \delta)$ and $|\delta| < 2^{-k}$, see [13, Theorem 2.2]. A standard analysis of rounding errors in finite-precision arithmetic [13, Chapter 3] yields the following lemma:

▶ **Lemma 13.** *Let π be an initial distribution and $0 < \theta < 1$. Let $k \in \mathbb{N}$ be such that $2^k \ge 2(n+1)|Q|/\theta$. Let $w = a_1 a_2 \cdots a_m \in \Sigma^*$ with $m \le n$. Compute $\widetilde{\pi}(w)$ as*

$$((\cdots((\pi \cdot M(a_1)) \cdot M(a_2)) \cdots) \cdot M(a_m)) \cdot \eta^\top ,$$

where rounding $\langle \cdot \rangle_k$ is applied after each individual (scalar) multiplication and addition. Then $\widetilde{\pi}(w) \in [\pi(w)(1-\theta), \pi(w)(1+\theta)]$.

Proof. For all $i \in \mathbb{N}$ write $\gamma_i := i \cdot 2^{-k}/(1 - i \cdot 2^{-k})$. By [13, Equation (3.11)] there are matrices $\Delta_1, \ldots, \Delta_m$ and a vector $\widetilde{\eta}$ such that

$$\widetilde{\pi}(w) \;=\; \pi \cdot (M(a_1) + \Delta_1) \cdot (M(a_2) + \Delta_2) \cdots (M(a_m) + \Delta_m) \cdot (\eta + \widetilde{\eta})^\top$$

and $|\Delta_i| \le \gamma_{|Q|} M(a_i)$ and $|\widetilde{\eta}| \le \gamma_{|Q|}\eta$, where by $|\Delta_i|$ and $|\widetilde{\eta}|$ we mean the matrix and vector obtained by taking the absolute value componentwise. (In words, the result $\widetilde{\pi}(w)$ of the

floating-point computation is the result of applying an exact computation with slightly perturbed data—a "backward error" result.) It follows:

$$\begin{aligned}
|\widetilde{\pi}(w) - \pi(w)| &\leq \left(-1 + \prod_{j=1}^{m+1} \left(1 + \gamma_{|Q|}\right) \right) \pi(w) && \text{by [13, Lemma 3.8]} \\
&\leq \gamma_{(m+1)\cdot|Q|}\pi(w) && \text{by [13, Lemma 3.3]} \\
&\leq 2(n+1)|Q| \cdot 2^{-k}\pi(w) && \text{as } (n+1)|Q| \cdot 2^{-k} \leq 1/2 \\
&\leq \theta\pi(w)
\end{aligned}$$

◀

The development so far suggests the following approximation approach: Let $\varepsilon > 0$ be the error bound from the input. Let $n \in \mathbb{N}$ be the number from Lemma 12, where λ is such that $2\lambda = \varepsilon/2$. Let $k \in \mathbb{N}$ be the smallest number such that $2^k \geq 2(n+1)|Q|/\theta$, where θ is such that $4\theta = \varepsilon/2$. Observe that k (the bit size of 2^k) is polynomial in the input. Define, for each word w and both i, the approximation $\widetilde{\pi}_i(w)$ as in Lemma 13. This defines also $\widetilde{W}_1, \widetilde{W}_2$. By (10) we have:

$$\pi_1(\widetilde{W}_1) + \pi_2(\widetilde{W}_2) - \frac{\varepsilon}{2} \quad \leq \quad 1 - d(\pi_1, \pi_2) \quad \leq \quad \pi_1(\widetilde{W}_1) + \pi_2(\widetilde{W}_2) + \frac{\varepsilon}{2}$$

Thus we can complete the proof of Theorem 11 by proving the following lemma:

▶ **Lemma 14.** *For both i, one can approximate $\pi_i(\widetilde{W}_i)$ within $\varepsilon/4$ in PSPACE.*

Proof. We discuss only the approximation of $\pi_1(\widetilde{W}_1)$; the case of $\pi_2(\widetilde{W}_2)$ is similar.

Construct a "probabilistic PSPACE Turing machine" $\mathcal{T}$ that samples a random word w according to $\Pr_{\pi_1}$. For that, $\mathcal{T}$ uses probabilistic branching according to the transition probabilities in M. While producing w in this way, but without storing w as a whole, $\mathcal{T}$ computes also the values $\widetilde{\pi}_1(w), \widetilde{\pi}_2(w)$ according to Lemma 13. If and when w gets longer than n then $\mathcal{T}$ rejects. If $\widetilde{\pi}_1(w) < \widetilde{\pi}_2(w)$ then $\mathcal{T}$ accepts; otherwise $\mathcal{T}$ rejects. The probability that $\mathcal{T}$ accepts equals $\pi_1(\widetilde{W}_1)$. This probability can be computed in PSPACE by Ladner's result [19] on counting in polynomial space. To be precise, note that this probability is a fraction p/q of two natural numbers p, q of at most exponential bit size. By Ladner's result one can compute arbitrary bits of p, q in PSPACE. Hence an approximation within $\varepsilon/4$ can also be computed in PSPACE. Technical details about how we apply Ladner's result are provided in [15]. ◀

5 Open Problems

In this paper we have considered the total variation distance between the distributions on finite words that are generated by two LMCs. In a more general version of LMCs, the end-of-word probabilities are zero, so that the LMC generates infinite words. The production of finite words $w \in \Sigma^*$ can be simulated by producing $w\$\$\$\cdots$ where \$ is an end-of-word symbol. It follows that the undecidability and hardness results of this paper apply equally to infinite-word LMCs. In fact, all these results strengthen those from [5], where the total variation distance between infinite-word LMCs is studied. The PSPACE approximation algorithm in this paper (Theorem 11) applies only to finite words, and the author does not know if it can be generalized to infinite-word LMCs. Whether the non-strict threshold-distance problem is decidable is open, both for finite- and for infinite-word LMCs.

Another direction concerns LMCs that are *not hidden*, i.e., where each emitted label identifies the next state; or, slightly more general, *deterministic* LMCs, i.e., where each state

and each emitted label identify the next state. The reduction that shows square-root-sum hardness in [5, Theorem 15] also applies to the threshold-distance problem for deterministic finite-word LMCs, but the author does not know a hardness result for approximating the distance between deterministic LMCs.

References

1 P. Ailliot, C. Thompson, and P. Thomson. Space-time modelling of precipitation by using a hidden Markov model and censored Gaussian distributions. *Journal of the Royal Statistical Society*, 58(3):405–426, 2009.

2 M. Alexandersson, S. Cawley, and L. Pachter. SLAM: Cross-species gene finding and alignment with a generalized pair hidden Markov model. *Genome Research*, 13:469–502, 2003.

3 D. Chen, F. van Breugel, and J. Worrell. On the complexity of computing probabilistic bisimilarity. In *Proceedings of FoSSaCS*, volume 7213 of *LNCS*, pages 437–451. Springer, 2012.

4 F.-S. Chen, C.-M. Fu, and C.-L. Huang. Hand gesture recognition using a real-time tracking method and hidden Markov models. *Image and Vision Computing*, 21(8):745–758, 2003.

5 T. Chen and S. Kiefer. On the total variation distance of labelled Markov chains. In *Proceedings of CSL-LICS*, pages 33:1–33:10, 2014.

6 G.A. Churchill. Stochastic models for heterogeneous DNA sequences. *Bulletin of Mathematical Biology*, 51(1):79–94, 1989.

7 C. Cortes, M. Mohri, and A. Rastogi. L_p distance and equivalence of probabilistic automata. *International Journal of Foundations of Computer Science*, 18(04):761–779, 2007.

8 M.S. Crouse, R.D. Nowak, and R.G. Baraniuk. Wavelet-based statistical signal processing using hidden Markov models. *IEEE Transactions on Signal Processing*, 46(4):886–902, April 1998.

9 C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk. A Storm is coming: A modern probabilistic model checker. In *Proceedings of Computer Aided Verification (CAV)*, pages 592–600. Springer, 2017.

10 J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled Markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004.

11 R. Durbin. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

12 S.R. Eddy. What is a hidden Markov model? *Nature Biotechnology*, 22(10):1315–1316, October 2004.

13 N. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, second edition, 2002.

14 S. Kannan, Z. Sweedyk, and S. Mahaney. Counting and random generation of strings in regular languages. In *Proceedings of SODA*, pages 551–557, 1995.

15 S. Kiefer. On computing the total variation distance of hidden Markov models. Technical report, arxiv.org, 2018. Available at `https://arxiv.org/abs/1804.06170`.

16 S. Kiefer, A.S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. Language equivalence for probabilistic automata. In *Proceedings of Computer Aided Verification (CAV)*, volume 6806 of *LNCS*, pages 526–540. Springer, 2011.

17 A. Krogh, B. Larsson, G. von Heijne, and E.L.L. Sonnhammer. Predicting transmembrane protein topology with a hidden Markov model: Application to complete genomes. *Journal of Molecular Biology*, 305(3):567–580, 2001.

18 M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of Computer Aided Verification (CAV)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

19 R. E. Ladner. Polynomial space counting problems. *SIAM Journal on Computing*, 18(6):1087–1097, 1989.

20 R.B. Lyngsø and C.N.S. Pedersen. The consensus string problem and the complexity of comparing hidden Markov models. *J. Comput. Syst. Sci.*, 65(3):545–569, 2002.

21 M. Mitzenmacher and E. Upfal. *Probability and Computing*. Cambridge University Press, 2005.

22 A. Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.

23 L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

24 M.-P. Schützenberger. On the definition of a family of automata. *Inf. and Control*, 4:245–270, 1961.

25 S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal of Computing*, 20(5):865–877, 1991.

To Infinity and Beyond

Ines Klimann
Univ Paris Diderot, Sorbonne Paris Cité, IRIF, UMR 8243 CNRS, F-75013 Paris, France
klimann@irif.fr

Abstract

We prove that if a group generated by a bireversible Mealy automaton contains an element of infinite order, then it must have exponential growth. As a direct consequence, no infinite virtually nilpotent group can be generated by a bireversible Mealy automaton.

2012 ACM Subject Classification Theory of computation → Models of computation, Theory of computation → Formal languages and automata theory → Automata over infinite objects

Keywords and phrases automaton groups, growth of a group, exponential growth

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.131

Funding This work was partially supported by the French *Agence Nationale pour la Recherche*, through the Project **MealyM** ANR-JS02-012-01.

Acknowledgements The author is very grateful to Dmytro Savchuk for suggesting Corollary 16.

The study on how (semi)groups grow has been highlighted since Milnor's question on the existence of groups of intermediate growth (faster than any polynomial and slower than any exponential) in 1968 [13], and since the very first example of such a group was given by Grigorchuk [5]. Uncountably many examples have followed this first one, see for instance [6]. Bartholdi and Erschler have even obtained results on precise computations of growth, in particular they proved that if a function satisfies some frame property, then there exists a finitely generated group with growth equivalent to it [1]. Besides, until Nekrashevych's very recent work [14], all the known examples of intermediate growth groups were automaton groups or based on automaton groups.

There exists no sufficient or necessary criterion to test whether a Mealy automaton generates a group of intermediate growth and it is not even known if this property is decidable. However, there is no known example in the literature of a bireversible Mealy automaton generating an intermediate growth group and it is legitimate to wonder whether it is possible. This is the scope of the present article, which can be seen as an extension of earlier results on the two-state case [8, 9]. We prove that if a group generated by a bireversible Mealy automaton has an element of infinite order, then it must have exponential growth. It has been conjectured, and proved in some cases [4], that an infinite group generated by a bireversible Mealy automaton always has an element of infinite order, which suggests that, indeed, a group generated by a bireversible Mealy automaton either is finite, or has exponential growth. Note that a similar result has been proved for a very different family of Mealy automata, namely reset automata [15]: since a Mealy automaton which is simultaneously reset and bireversible has a unique state, these families are somehow orthogonal; moreover the proof we present here is completely different from the proof in [15], in particular it is not based on the potential existence of a free subsemigroup of rank 2.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 131; pp. 131:1–131:12


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Finally, let us mention the work by Brough and Cain to obtain some necessary criteria to decide whether a semigroup is an automaton semigroup [2]. Our work can be seen as partially answering a similar question: can a given group be generated by a bireversible Mealy automaton? A consequence of our result is that no infinite virtually nilpotent group (and in particular no abelian group) can be.

This article is organized as follows. In Section 1, we define automaton groups and the growth of a group, and give some properties of the connected components of the powers of a Mealy automaton. In Section 2, we study the behavior of certain equivalence classes of words on the state set of a Mealy automaton. Finally, the main result and some of its consequences appear in Section 3.

1 Basic notions

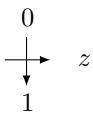
Throughout the paper, the cardinality of a finite set E is denoted by $|E|$. A *finite word* of *length* n on E is a finite sequence of n elements of E and is denoted classically as the concatenation of its elements. The set of finite words over E is denoted by E^*, the set of non-empty finite words by E^+, and the set of words of length n by E^n. In general the elements of E are written in plain letters, *e.g.* q, while the words on E are written in bold letters, *e.g.* $\mathbf{u}$. The length of $\mathbf{u}$ is denoted by $|\mathbf{u}|$, its letters are numbered from 0 to $|\mathbf{u}|-1$ and, if i is an integer, its $(i \mod |\mathbf{u}|)$-th letter is denoted by $\mathbf{u}\{i\}$; for example its first letter is $\mathbf{u}\{0\}$, while its last letter is $\mathbf{u}\{-1\}$. If L is a set of words on E, $L\{i\}$ denotes the set $\{\mathbf{u}\{i\} \mid \mathbf{u} \in L\}$.

1.1 Semigroups and groups generated by Mealy automata

We first recall the formal definition of an automaton. A *(finite, deterministic, and complete) automaton* is a triple $\big(Q, \Sigma, \delta = (\delta_i \colon Q \to Q)_{i\in\Sigma}\big)$, where the *state set* Q and the *alphabet* Σ are non-empty finite sets, and the δ_i are functions.

A *Mealy automaton* is a quadruple $\mathcal{A} = (Q, \Sigma, \delta, \rho)$, such that (Q, Σ, δ) and (Σ, Q, ρ) are both automata. In other terms, a Mealy automaton is a complete, deterministic, letter-to-letter transducer with the same input and output alphabet. Its *size* is the cardinality of its state set and is denoted by $\#\mathcal{A}$.

The graphical representation of a Mealy automaton is standard, see Figure 1. But, for practical reasons, we use sometimes other graphical representations for the transitions. For example the transition from x to z with input letter 0 and output letter 1 in the automaton of Figure 1 can be represented

either by $x \xrightarrow{0|1} z$, or by  .

Let $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ be a Mealy automaton. Each state $q \in Q$ defines a mapping from Σ^* into itself, recursively by:

$$\forall i \in \Sigma,\ \forall \mathbf{s} \in \Sigma^*, \qquad \rho_q(i\mathbf{s}) = \rho_q(i)\rho_{\delta_i(q)}(\mathbf{s})\ .$$

The image of the empty word is itself. For each $q \in Q$, the mapping ρ_q is length-preserving and prefix-preserving. We say that $\rho_q : \Sigma^* \to \Sigma^*$ is the *action induced* by q. For $\mathbf{u} = q_1 \cdots q_n \in Q^n$ with $n > 0$, set $\rho_{\mathbf{u}} \colon \Sigma^* \to \Sigma^*, \rho_{\mathbf{u}} = \rho_{q_n} \circ \cdots \circ \rho_{q_1}$.

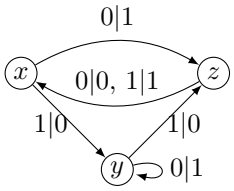


Figure 1 The Aleshin automaton.

The semigroup of mappings from Σ^* to Σ^* generated by $\{\rho_q, q \in Q\}$ is called the *semigroup generated by* $\mathcal{A}$ and is denoted by $\langle \mathcal{A} \rangle_+$.

We extend to δ the former notation on ρ, in a natural way. Hence $\delta_i \colon Q^* \to Q^*, i \in \Sigma$, are the functions extended to Q^*, and for $\mathbf{s} = i_1 \cdots i_n \in \Sigma^n$ with $n > 0$, we set $\delta_{\mathbf{s}} \colon Q^* \to Q^*$, $\delta_{\mathbf{s}} = \delta_{i_n} \circ \cdots \circ \delta_{i_1}$.

A Mealy automaton $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ is *invertible* if the functions ρ_q are permutations of the alphabet Σ. In this case, the actions induced by the states are permutations on words of the same length and thus we may consider the group of mappings from Σ^* to Σ^* generated by $\{\rho_q, q \in Q\}$: it is called the *group generated by* $\mathcal{A}$ and is denoted by $\langle \mathcal{A} \rangle$.

When $\mathcal{A}$ is invertible, define its *inverse* $\mathcal{A}^{-1}$ as the Mealy automaton with state set Q^{-1}, a disjoint copy of Q, and alphabet Σ, where the transition $p^{-1} \xrightarrow{j|i} q^{-1}$ belongs to $\mathcal{A}^{-1}$ if and only if the transition $p \xrightarrow{i|j} q$ belongs to $\mathcal{A}$. Clearly the action induced by the state p^{-1} of $\mathcal{A}^{-1}$ is the reciprocal of the action induced by the corresponding state p in $\mathcal{A}$.

A Mealy automaton $(Q, \Sigma, \delta, \rho)$ is *reversible* if the functions δ_i induced on Q by the input letters of the transitions are permutations. The connected components of a reversible automaton are strongly connected. In a reversible automaton of state set Q and alphabet Σ, for any word $\mathbf{s} \in \Sigma^*$ and any state q, there exists exactly one path in the automaton with label $\mathbf{s}$ and final state q, hence we can consider the *backtrack action* induced by q: it associates to $\mathbf{s}$ the output label $\mathbf{t} \in \Sigma^{|\mathbf{s}|}$ of this single path.

A Mealy automaton is *coreversible* if the functions induced on Q by the letters as output letters of the transitions are permutations.

A Mealy automaton is *bireversible* if it is both reversible and coreversible. It is quite simple to see that the actions and the backtrack actions induced by the states of a bireversible automaton are both permutations.

Two Mealy automata are said to be *isomorphic* if they are identical up to the labels of their states.

1.2 Growth of a semigroup or a group

Let H be a semigroup generated by a finite set S. The *length* of an element g of the semigroup, denoted by $|g|$, is the length of its shortest decomposition as a product of generators:

$$|g| = \min\{n \mid \exists s_1, \ldots, s_n \in S,\ g = s_1 \cdots s_n\}\,.$$

The *growth function* γ_H^S of the semigroup H with respect to the generating set S enumerates the elements of H with respect to their length:

$$\gamma_H^S(n) = |\{g \in H\,;\, |g| \leq n\}|\,.$$

The *growth functions* of a group are defined similarly by taking symmetrical generating sets.

The growth functions corresponding to two generating sets are equivalent [12], so we may define the *growth* of a group or a semigroup as the equivalence class of its growth functions. Hence, for example, a finite (semi)group has bounded growth, while an infinite abelian (semi)group has polynomial growth, and a non-abelian free (semi)group has exponential growth.

It is quite easy to obtain groups of polynomial or exponential growth. Answering a question of Milnor [13], Grigorchuk gave an example of an automaton group of intermediate growth [5]: faster than any polynomial, slower than any exponential, opening thus a new classification criterion for groups, that has been deeply studied since this seminal article (see [7] and references therein). Besides, let us mention that until Nekrashevych's very recent work [14], all the known examples of intermediate growth groups were automaton groups or based on automaton groups.

Note an important point for our purpose: let G be a finitely generated group and let $(I_n)_{n>0}$ a sequence of subsets of G, compatible with the length function, *i.e.* the sets I_n are pairwise distinct and the elements of I_n have all length less than or equal to n. The growth function of $(I_n)_{n>0}$ is given by $(\sum_{k\leq n} |I_k|)_{n>0}$; if it grows exponentially, then so does G. In the same spirit, a group which admits a subgroup of exponential growth grows exponentially.

1.3 The powers of a Mealy automaton and their connected components

The powers of a Mealy automaton have been shown to play an important role in the finiteness and the order problems for an automaton (semi)group, as highlighted in [9, 11, 4]. The *n-th power* of the automaton $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ is the Mealy automaton

$$\mathcal{A}^n = \left(Q^n, \Sigma, (\delta_i \colon Q^n \to Q^n)_{i\in\Sigma}, (\rho_{\mathbf{u}} \colon \Sigma \to \Sigma)_{\mathbf{u}\in Q^n} \right) .$$

Note that the powers of a reversible (*resp.* bireversible) Mealy automaton are reversible (*resp.* bireversible). It is straightforward to see that the action induced by a word $\mathbf{u}$ of states of $\mathcal{A}$ is in fact the action induced by $\mathbf{u}$ seen as a state of $\mathcal{A}^{|\mathbf{u}|}$. The backtrack action induced by $\mathbf{u}$ can be defined in the same spirit.

The (semi)group generated by a connected component of some power of $\mathcal{A}$ is a sub(semi)group of the (semi)group generated by $\mathcal{A}$.

Let $\mathbf{u}$ and $\mathbf{v}$ be elements of Q^+ and $\mathcal{C}$ be a connected component of some power of $\mathcal{A}$: $\mathbf{v}$ *can follow* $\mathbf{u}$ *in* $\mathcal{C}$ if $\mathbf{uv}$ is the prefix of some state of $\mathcal{C}$. We denote by $\{\mathbf{u}? \hookrightarrow \mathcal{C}\}$ the set of states that can follow $\mathbf{u}$ in $\mathcal{C}$:

$$\{\mathbf{u}? \hookrightarrow \mathcal{C}\} = \{q \in Q \mid \mathbf{u}q \text{ is the prefix of some state of } \mathcal{C}\} .$$

We define similarly the fact that $\mathbf{v}$ *can precede* $\mathbf{u}$ in $\mathcal{C}$ if $\mathbf{vu}$ is the suffix of some state of $\mathcal{C}$, and we introduce the set

$$\{?\mathbf{u} \hookrightarrow \mathcal{C}\} = \{q \in Q \mid q\mathbf{u} \text{ is the suffix of some state of } \mathcal{C}\} .$$

The aim of this section is to give some intuition on the links between the connected components of consecutive powers of $\mathcal{A}$. Since a word can be extended with a prefix or a suffix, most of the results exposed here are expressed in both cases, but only the first result is proved in both cases, to show how bireversibility allows to consider similarly the actions and the backtrack actions.

▶ **Lemma 1.** *Let $\mathcal{A}$ be a bireversible Mealy automaton with state set Q and $\mathcal{C}$ a connected component of one of its powers. If $\mathbf{u} \in Q^+$ is a proper prefix of some state of $\mathcal{C}$, then the cardinality of the set $\{\mathbf{u}? \looparrowright \mathcal{C}\}$ depends only on the length of $\mathbf{u}$.*

Proof. Suppose that $\mathbf{u}'$ is such that $\mathbf{u}\mathbf{u}'$ is a state of $\mathcal{C}$, and let $\mathbf{v}$ be a prefix of some state $\mathbf{v}\mathbf{v}'$ of $\mathcal{C}$ with the same length as $\mathbf{u}$. Since $\mathcal{C}$ is a connected component in a reversible Mealy automaton, it is strongly connected, so there exists a word $\mathbf{s} \in \Sigma^*$ such that $\delta_{\mathbf{s}}(\mathbf{u}\mathbf{u}') = \mathbf{v}\mathbf{v}'$. Now, consider the action induced by $\mathbf{s}$ on $\mathbf{u}p$, for $p \in \{\mathbf{u}? \looparrowright \mathcal{C}\}$:

$$\begin{array}{ccc} & \mathbf{s} & \\ \mathbf{u} & \to & \mathbf{v} \\ & \mathbf{s}' & \\ p & \to & p' \end{array}$$

Since the automaton $\mathcal{A}$ is reversible, the action induced by $\mathbf{s}'$ is a permutation of Q, and we have

$$|\{\mathbf{u}? \looparrowright \mathcal{C}\}| \leq |\{\mathbf{v}? \looparrowright \mathcal{C}\}| .$$

The reciprocal inequality is obtained symmetrically. ◀

▶ **Lemma 2.** *Let $\mathcal{A}$ be a bireversible Mealy automaton with state set Q and $\mathcal{C}$ a connected component of one of its powers. If $\mathbf{u} \in Q^+$ is a proper suffix of some state of $\mathcal{C}$, then the cardinality of the set $\{?\mathbf{u} \looparrowright \mathcal{C}\}$ depends only on the length of $\mathbf{u}$.*

Proof. Suppose that $\mathbf{u}'$ is such that $\mathbf{u}'\mathbf{u}$ is a state of $\mathcal{C}$, and let $\mathbf{v}$ be a suffix of some state $\mathbf{v}'\mathbf{v}$ of $\mathcal{C}$ with the same length as $\mathbf{u}$. Since $\mathcal{C}$ is a connected component in a reversible Mealy automaton, it is strongly connected, so there exist words $\mathbf{s}, \mathbf{t} \in \Sigma^*$ such that $\delta_{\mathbf{s}}(\mathbf{u}'\mathbf{u}) = \mathbf{v}'\mathbf{v}$ and $\rho_{\mathbf{u}'\mathbf{u}}(\mathbf{s}) = \mathbf{t}$:

$$\begin{array}{ccc} & \mathbf{s} & \\ \mathbf{u}' & \to & \mathbf{v}' \\ & \mathbf{s}' & \\ \mathbf{u} & \to & \mathbf{v} \\ & \mathbf{t} & \end{array}$$

Now, consider the backtrack action induced by $\mathbf{t}$ on $p\mathbf{u}$, for $p \in \{?\mathbf{u} \looparrowright \mathcal{C}\}$:

$$\begin{array}{ccc} p & \to & p' \\ & \mathbf{s}' & \\ \mathbf{u} & \to & \mathbf{v} \\ & \mathbf{t} & \end{array}$$

Since the automaton $\mathcal{A}$ is bireversible, the backtrack action induced by $\mathbf{s}'$ is a permutation of Q, and we have

$$|\{?\mathbf{u} \looparrowright \mathcal{C}\}| \leq |\{?\mathbf{v} \looparrowright \mathcal{C}\}| .$$

The reciprocal inequality is obtained symmetrically. ◀

Consider a state q of $\mathcal{A}$. For any integer $n > 0$, we denote by $\mathrm{cc}(q^n)$ the connected component of q^n in $\mathcal{A}^n$. The sequence of such components has some properties which we give here. These properties can be seen as properties of the branch represented by q^ω in the Schreier trie of $\mathcal{A}$ (also known as the orbit tree of the dual of $\mathcal{A}$) which has been introduced in [11, 4] (to keep this article self-contained, we give here only the properties of this branch, but for a more global intuition on the constructions, the reader can consult these references).

The first point is given by Lemma 1: for any $n > 0$, the component $\mathrm{cc}(q^{n+1})$ can be seen as several full copies of the component $\mathrm{cc}(q^n)$; indeed, if $\mathbf{u}$ and $\mathbf{v}$ are states of $\mathrm{cc}(q^n)$, then in $\mathrm{cc}(q^{n+1})$ there are as many states with prefix $\mathbf{u}$ as states with prefix $\mathbf{v}$, *i.e.*

$$\forall \mathbf{u} \in \mathrm{cc}(q^n), \forall \mathbf{v} \in \mathrm{cc}(q^n),\ |\{\mathbf{u}? \looparrowright \mathrm{cc}(q^{n+1})\}| = |\{\mathbf{v}? \looparrowright \mathrm{cc}(q^{n+1})\}| \,.$$

Hence the ratio between the size of $\mathrm{cc}(q^{n+1})$ and the size of $\mathrm{cc}(q^n)$ is necessarily an integer: it is the cardinality of the set $\{\mathbf{u}? \looparrowright \mathrm{cc}(q^{n+1})\}$ for any state $\mathbf{u}$ of $\mathrm{cc}(q^n)$, and in particular for $\mathbf{u} = q^n$.

We define by

$$\left(\frac{\#\,\mathrm{cc}(q^{n+1})}{\#\,\mathrm{cc}(q^n)}\right)_{n>0} = \left(|\{q^n? \looparrowright \mathrm{cc}(q^{n+1})\}|\right)_{n>0}$$

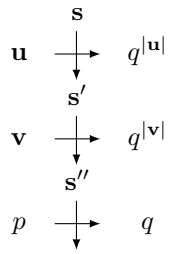
the *sequence of ratios* associated to the state q.

▶ **Lemma 3.** *Let $\mathcal{A}$ be a bireversible Mealy automaton with state set Q, $q \in Q$ a state of $\mathcal{A}$, and $\mathbf{u}$ and $\mathbf{v}$ be two elements of Q^+. If $\mathbf{uv}$ is a state of $\mathrm{cc}(q^{|\mathbf{uv}|})$, then $\mathbf{v}$ is a state of $\mathrm{cc}(q^{|\mathbf{v}|})$ and*

$$\{\mathbf{uv}? \looparrowright \mathrm{cc}(q^{|\mathbf{uv}|+1})\} \subseteq \{\mathbf{v}? \looparrowright \mathrm{cc}(q^{|\mathbf{v}|+1})\} \,.$$

Proof. Take $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ and let $p \in \{\mathbf{uv}? \looparrowright \mathrm{cc}(q^{|\mathbf{uv}|+1})\}$.

Since $\mathcal{A}$ is reversible, there exists a word $\mathbf{s} \in \Sigma^*$ such that $\delta_{\mathbf{s}}(\mathbf{uv}p) = q^{|\mathbf{uv}|+1}$:



Hence $\delta_{\mathbf{s}'}(\mathbf{v}) = q^{|\mathbf{v}|}$ and so $\mathbf{v} \in \mathrm{cc}(q^{|\mathbf{v}|})$, and $\delta_{\mathbf{s}'}(\mathbf{v}p) = q^{|\mathbf{v}|+1}$, which means that $p \in \{\mathbf{v}? \looparrowright \mathrm{cc}(q^{|\mathbf{v}|+1})\}$. ◀

▶ **Lemma 4.** *Let $\mathcal{A}$ be a bireversible Mealy automaton with state set Q, $q \in Q$ a state of $\mathcal{A}$, and $\mathbf{u}$ and $\mathbf{v}$ be two elements of Q^+. If $\mathbf{uv}$ is a state of $\mathrm{cc}(q^{|\mathbf{uv}|})$, then $\mathbf{u}$ is a state of $\mathrm{cc}(q^{|\mathbf{u}|})$ and*

$$\{?\mathbf{uv} \looparrowright \mathrm{cc}(q^{|\mathbf{uv}|+1})\} \subseteq \{?\mathbf{u} \looparrowright \mathrm{cc}(q^{|\mathbf{u}|+1})\} \,.$$

As $\{q^{n+1}? \looparrowright \mathrm{cc}(q^{n+2})\} \subseteq \{q^n? \looparrowright \mathrm{cc}(q^{n+1})\}$ by Lemma 3, it is straightforward to see that the sequence of ratios associated to q decreases:

$$\forall n > 0,\ \frac{\#\,\mathrm{cc}(q^{n+2})}{\#\,\mathrm{cc}(q^{n+1})} \leq \frac{\#\,\mathrm{cc}(q^{n+1})}{\#\,\mathrm{cc}(q^n)} \,,$$

and hence is ultimately constant. We say that q *has a constant ratio* if this sequence is in fact constant, and then the unique value of the sequence of ratios associated to q is called the *ratio of* q.

It has been proven in [11, Prop. 6] that q induces an action of infinite order if and only if the sizes of the components $(\mathrm{cc}(q^n))_{n>0}$ are unbounded, *i.e.* the limit of the sequence of ratios associated to q is greater than 1.

We study now some properties of followers and predecessors in the components $\mathrm{cc}(q^n)$, when q has a constant ratio.

The next lemma is an improvement of Lemma 3.

▶ **Lemma 5.** *Let $\mathcal{A}$ be a bireversible Mealy automaton, and q be a state of $\mathcal{A}$ of constant ratio. Let $\mathbf{u}$ and $\mathbf{v}$ be two elements of Q^+ such that $\mathbf{uv}$ is a state of $\mathrm{cc}(q^{|\mathbf{uv}|})$. We have:*

$$\{\mathbf{uv}? \rightsquigarrow \mathrm{cc}(q^{|\mathbf{uv}|+1})\} = \{\mathbf{v}? \rightsquigarrow \mathrm{cc}(q^{|\mathbf{v}|+1})\}\,.$$

Proof. The left part is a subset of the right one by Lemma 3 and both sets have the same cardinality, which is the ratio of q, by hypothesis. ◀

In particular, by taking the word $\mathbf{v}$ of length 1 in the previous lemma, we can see that the set of followers of a word $\mathbf{w}$ in $\mathrm{cc}(q^n)$ only depends on its last letter $\mathbf{w}\{-1\}$.

We obtain the symmetrical result by considering the mirror Mealy automaton obtained by inverting the direction of the transitions.

▶ **Lemma 6.** *Let $\mathcal{A}$ be a bireversible Mealy automaton of state set Q, q be a state of $\mathcal{A}$ of constant ratio, and $n > 1$ be an integer. If $\mathbf{u} \in Q^+$ is a suffix of some state of $\mathrm{cc}(q^n)$, then the set $\{?\mathbf{u} \rightsquigarrow \mathrm{cc}(q^n)\}$ only depends on $\mathbf{u}\{0\}$, the first letter of $\mathbf{u}$.*

The next lemma links up the sets of followers and of predecessors in $\mathrm{cc}(q^n)$ when q has a constant ratio.

▶ **Lemma 7.** *Let $\mathcal{A}$ be a bireversible Mealy automaton of state set Q, q be a state of $\mathcal{A}$ of constant ratio, and $n > 1$ be an integer. The sets of followers and of predecessors in $\mathrm{cc}(q^n)$ have the same cardinality which is the ratio of q.*

Proof. By Lemmas 5 and 6, we only have to prove that

$$|\{q? \rightsquigarrow \mathrm{cc}(q^n)\}| = |\{?q \rightsquigarrow \mathrm{cc}(q^n)\}|\,.$$

Even simpler: notice that if $\mathbf{u}$ is a prefix of some state of $\mathrm{cc}(q^n)$, then it is also a prefix of some state in $\mathrm{cc}(q^{n+k})$ for any $k > 0$, and it has the same set of followers in both. Of course, an equivalent property holds for the sets of predecessors.

So it is sufficient to prove that

$$|\{q? \rightsquigarrow \mathrm{cc}(q^2)\}| = |\{?q \rightsquigarrow \mathrm{cc}(q^2)\}|\,,$$

which is true because

$$|\,\mathrm{cc}(q^2)| = |\,\mathrm{cc}(q)| \times |\{q? \rightsquigarrow \mathrm{cc}(q^2)\}| = |\{?q \rightsquigarrow \mathrm{cc}(q^2)\}| \times |\,\mathrm{cc}(q)|\,.$$ ◀

2 Several equivalences on words

2.1 Minimization and Nerode classes

Let $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ be a Mealy automaton.

The *Nerode equivalence* $\equiv$ *on* Q is the limit of the sequence of increasingly finer equivalences $(\equiv_k)$ recursively defined by:

$$\begin{aligned} \forall p, q \in Q, \qquad & p \equiv_0 q \iff \rho_{p|\Sigma} = \rho_{q|\Sigma}\,, \\ \forall k \geqslant 0,\ & p \equiv_{k+1} q \iff \big(p \equiv_k q \quad \wedge \quad \forall i \in \Sigma,\ \delta_i(p) \equiv_k \delta_i(q)\big)\,. \end{aligned}$$

Since the set Q is finite, this sequence is ultimately constant. For every element q in Q, we denote by $[q]$ the class of q w.r.t. the Nerode equivalence, called the *Nerode class* of q. Extending to the n-th power of $\mathcal{A}$, we denote by $[\mathbf{u}]$ the Nerode class in Q^n of $\mathbf{u} \in Q^n$.

Two states of a Mealy automaton belong to the same Nerode class if and only if they represent the same element in the generated (semi)group, *i.e.* if and only if they induce the same action on Σ^*. Two words on Q of the same length n are *equivalent* if they belong to the same Nerode class in Q^n. By extension, any two words on Q are *equivalent* if they induce the same action.

The *minimization* of $\mathcal{A}$ is the Mealy automaton $\mathfrak{m}(\mathcal{A}) = (Q/\equiv, \Sigma, \tilde{\delta}, \tilde{\rho})$, where for every (q, i) in $Q \times \Sigma$, $\tilde{\delta}_i([q]) = [\delta_i(q)]$ and $\tilde{\rho}_{[q]} = \rho_q$. This definition is consistent with the standard minimization of "deterministic finite automata" where instead of considering the mappings $(\rho_q \colon \Sigma \to \Sigma)_q$, the computation is initiated by the separation between terminal and non-terminal states.

A Mealy automaton is *minimal* if it has the same size as its minimization.

Two states of two different connected reversible minimal Mealy automata with the same alphabet induce the same action if and only if the automata are isomorphic and both states are in correspondence by this isomorphism (which can be formally defined by considering the union of the involved Mealy automata). As a direct consequence, if two connected reversible minimal Mealy automata have different sizes, then any two states of each of them cannot be equivalent.

As we have seen in Section 1.3, a state q of an invertible-reversible Mealy automaton induces an action of infinite order if and only if the sizes of the $(\mathrm{cc}(q^n))_{n>0}$ are unbounded. The proof of [11, Prop. 6] can be easily adapted to see that q induces an action of infinite order if and only if the sizes of the $(\mathfrak{m}(\mathrm{cc}(q^n)))_{n>0}$ are unbounded, but you can see it by a direct argument: if the sizes are bounded, there is an infinite set $I \subseteq \mathbb{N}$ such that all the element $(\mathfrak{m}(\mathrm{cc}(q^n)))_{n\in I}$ are isomorphic, and in this sequence there exist at least two different integer $i \neq j$ such that q^i and q^j are represented by the same state in the minimal automata, and so they induce the same action; if the sizes are unbounded, the sequence $(\#\,\mathfrak{m}(\mathrm{cc}(q^n)))_{n>0}$ has infinitely many values, and there are infinitely many powers of the action induced by q that are pairwise different.

Note that the Nerode classes of a connected reversible Mealy automaton have the same cardinality. The size of the minimization of an automaton in this case is the ratio between the size of the former automaton and the cardinality of the Nerode classes.

▶ **Lemma 8.** *Let $\mathcal{A}$ be a connected bireversible Mealy automaton, N a Nerode class of a connected component of one of its powers, and p and q two elements of $N\{-1\}$. There are as many elements of N with last letter p as with last letter q.*

Proof. The proof of this lemma is quite similar to the proof of Lemma 5 considering not words of length 1 as predecessors, but words of length $n-1$, where n is the length of the states in N. ◀

2.2 Restricted Nerode classes

When the considered automaton is not connected, it can be interesting to consider the restriction of the Nerode class of an element to its connected component: we denote it by $[\![q]\!]$ and call it the *restricted Nerode class* of q. The restricted Nerode class of a state coincide with its Nerode class in the case of a connected Mealy automaton.

▶ **Lemma 9.** *The restricted Nerode classes of two elements in the same connected component of a reversible Mealy automaton have the same cardinality.*

Let $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ be a bireversible automaton and q a state of $\mathcal{A}$. As it will be discussed in Section 3, the result of this article is somehow a generalization of a much simpler but similar result proved in [10]: in the case where all the powers of $\mathcal{A}$ are connected, the group generated by $\mathcal{A}$ has exponential growth. The strategy used in [10] is based on the fact that $[\![q^n]\!]q \subseteq [\![q^{n+1}]\!]$ when all the powers of $\mathcal{A}$ are connected. However in the more general case we study here, this fact is false: a priori there is no inclusion link between $[\![q^n]\!]$ and $[\![q^{n+1}]\!]$, because if $\mathbf{u}$ is a state of $[\![q^n]\!]$, then nothing ensures that $\mathbf{u}q$ belongs to $\mathrm{cc}(q^{n+1})$; hence we have to find a different strategy. For this purpose, we introduce the *q-restricted Nerode class* of q^n, *i.e.* the set of states of $[\![q^n]\!]$ which admit q as a suffix: $[\![q^n]\!] = [\![q^n]\!] \cap Q^*q$.

The aim of this section is to study the sequence $([\![q^n]\!])_{n>0}$, in particular in the case where q has a constant ratio.

▶ **Lemma 10.** *The sequence of q-restricted Nerode classes satisfies the following inclusion property:*

$$\forall n > 0,\ [\![q^n]\!] q \subseteq [\![q^{n+1}]\!] .$$

Proof. Let $\mathbf{u}$ be an element of $[\![q^n]\!]$: $\mathbf{u}$ is a state of $\mathrm{cc}(q^n)$, and $\{\mathbf{u}? \rightsquigarrow \mathrm{cc}(q^{n+1})\}$ depends only on $\mathbf{u}\{-1\} = q$ by Lemma 5. In particular q can follow $\mathbf{u}$ since $q \in \{q? \rightsquigarrow \mathrm{cc}(q^{n+1})\}$. Since $\mathbf{u}$ and q^n induce the same action, so do $\mathbf{u}q$ and q^{n+1}. ◀

The next results give more information on the growth of the q-restricted Nerode classes of q^n with respect to n.

▶ **Proposition 11.** *Let $\mathcal{A}$ be a bireversible Mealy automaton, and q be a state of $\mathcal{A}$ of constant ratio k. The ratio between the sizes of $[\![q^{n+1}]\!]$ and $[\![q^n]\!]$ is an integer and:*

$$\forall n > 0,\ \frac{|\,[\![q^{n+1}]\!]\,|}{|\,[\![q^n]\!]\,|} = |[\![q^{n+1}]\!]\{-2\}| .$$

In particular this ratio cannot be greater than k.

Proof. Let $p \in [\![q^{n+1}]\!]\{-2\}$. By this choice, p can precede the final q in $[\![q^{n+1}]\!]$. Now, take $\mathbf{u} \in Q^{n-1}p$: we have $\mathbf{u}q \in [\![q^{n+1}]\!]$ if and only if $\mathbf{u} \in [\![q^n]\!]$ (the direct sense is obtained by Lemma 5 and the reciprocal sense by Lemma 10). Hence by Lemma 8, $|[\![q^n]\!] \cap Q^*p| = |\,[\![q^n]\!]\,|$ (which means somehow that we can pinpoint the Nerode class by p instead of q, without changing the cardinality).

Consequently, the sets $[\![q^{n+1}]\!] \cap Q^*pq$ and $[\![q^{n+1}]\!] \cap Q^*qq$ have the same cardinality.

Since $p \in [\![q^{n+1}]\!]\{-2\}$ can precede q in $\mathrm{cc}(q^{n+1})$, we obtain the bound $k = \{?q \rightsquigarrow \mathrm{cc}(q^2)\}$ by Lemmas 6 and 7. ◀

▶ **Proposition 12.** *Let $\mathcal{A}$ be a bireversible Mealy automaton, and q be a state of $\mathcal{A}$ of constant ratio k. The sequence*

$$\left(\frac{|\,[\![\, q^{n+1} \,]\!]\,|}{|\,[\![\, q^{n} \,]\!]\,|}\right)_{n>0}$$

is ultimately increasing to a limit less than or equal to k.

Proof. Consider the sequence $([\![\, q^n \,]\!])_{n>0}$ and particularly the sequence $([\![\, q^n \,]\!]\{0\})_{n>0}$: by Lemma 10, this last sequence is increasing (for inclusion). Denote by Q_1 its limit: $Q_1 = [\![\, q^n \,]\!]\{0\}$ for n large enough, say $n \geq N$; in particular this set contains q.

Now, suppose $n > N$ and take $p \in [\![\, q^{n+1} \,]\!]\{-2\}$: clearly $([\![\, q^{n+1} \,]\!] \cap Q^*pq)\{0\}$ is a subset of $Q_1 = [\![\, q^{n+1} \,]\!]\{0\} = [\![\, q^n \,]\!]\{0\}$; since it has the same cardinality as the set $([\![\, q^{n+1} \,]\!] \cap Q^*qq)\{0\} = [\![\, q^n \,]\!]\{0\}$ by Lemma 8, it is in fact equal to Q_1. But q belongs to Q_1, so it means that the set $[\![\, q^{n+1} \,]\!] \cap qQ^*pq$ is not empty, take $\mathbf{u}$ one of its elements. By Lemma 6, q can precede $\mathbf{u}$ in $\mathrm{cc}(q^{n+2})$ and $q\mathbf{u} \in [\![\, q^{n+2} \,]\!]$. Hence any penultimate letter in $[\![\, q^{n+1} \,]\!]$ is also a penultimate letter in $[\![\, q^{n+2} \,]\!]$:

$$[\![\, q^{n+1} \,]\!]\{-2\} \subseteq [\![\, q^{n+2} \,]\!]\{-2\}\,.$$

The result is now a direct consequence of Proposition 11 ◀

3 Main result

Because all the elements of a semigroup generated by an invertible-reversible Mealy automaton, all whose powers are connected, have infinite order [11, Prop. 6], the main result of this article, Theorem 14, is somehow a generalization of this previous result obtained in [10]:

▶ **Theorem 13.** *A semigroup generated by an invertible-reversible Mealy automaton, all of whose powers are connected, has exponential growth.*

To prove this generalization, we need to reinforce the hypothesis on the structure of the automaton which is supposed here to be bireversible and not only invertible-reversible, but we do not use anymore the really strong hypothesis on the connected powers. Since it is (easily) decidable if a Mealy automaton is bireversible, while the condition on the powers is not known to be decidable, except in very restricted cases, the result here is much more interesting and powerful, but it is also trickier to establish. The question of the existence of elements of infinite order in a semigroup generated by a bireversible automaton has been under study for several years now and has been solved in quite a few cases [9, 11, 4].

Here is the generalized version of the former theorem we prove in this section:

▶ **Theorem 14.** *A group generated by a bireversible Mealy automaton which contains an element of infinite order has exponential growth.*

Note that the result still holds for the generated semigroup and the proof is easily adaptable.

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ be a bireversible Mealy automaton and $\mathbf{u} \in (Q \sqcup Q^{-1})^*$ which induces an action of infinite order. The Mealy automaton $(\mathcal{A} \cup \mathcal{A}^{-1})^{|\mathbf{u}|}$ is bireversible, $\mathbf{u}$ is one of its states, and this automaton generates a subgroup of $\langle \mathcal{A} \rangle$. So without loss of generality, we can suppose that $\mathbf{u}$ is in fact a state of $\mathcal{A}$. To be consistent with the rest of the article, let us call it q.

For any integer $i > 0$, denote $r_i = \frac{\#\,\mathrm{cc}(q^{i+1})}{\#\,\mathrm{cc}(q^i)}$. The sequence of ratios associated to q is of the form $(r_1, r_2, \ldots, r_j, r_{j+1} = r_j, \ldots)$, where $r_i \geq r_{i+1}$ for any $i \geq 1$ and $r_i = r_j$ for any $i \geq j$. Now, consider the component $\mathrm{cc}(q^j)$ as a Mealy automaton and $\mathbf{q} = q^j$ as its state: the state $\mathbf{q}$ induces an action of infinite order and the sequence of ratios associated to $\mathbf{q}$ is of the form $(r_j^j, r_j^j, \ldots)$: $\mathbf{q}$ has constant ratio. Moreover, $\mathrm{cc}(\mathbf{q})$ generates a subgroup of the group $\langle\mathcal{A}\rangle$, so if we prove that $\langle\mathrm{cc}(\mathbf{q})\rangle$ has exponential growth, so has $\langle\mathcal{A}\rangle$. So, without loss of generality we can suppose that q has a constant ratio, say k.

Note that the following inequalities hold:

$$\forall n > 0,\ \frac{|\,\mathrm{cc}(q^n)|}{|Q| \times |\,[\![\, q^n \,]\!]\,|} \leq \frac{|\,\mathrm{cc}(q^n)|}{|[\![q^n]\!]|} \leq \frac{|\,\mathrm{cc}(q^n)|}{|\,[\![\, q^n \,]\!]\,|}\,. \tag{1}$$

Indeed, the second inequality is a consequence of the fact that $[\![\, q^n \,]\!] \subseteq [\![q^n]\!]$, and the first inequality follows from Lemma 8 and the fact that $[\![q^n]\!] = \cup_{p\in Q}([\![q^n]\!] \cap Q^*p)$.

The central part in (1) is in fact the size of the minimization of $\mathrm{cc}(q^n)$, and the cardinality of $\mathrm{cc}(q^n)$ is equal to $|Q| \times k^{n-1}$ since q has constant ratio k, so (1) can be re-written as:

$$\forall n > 0,\ \frac{k^{n-1}}{|\,[\![\, q^n \,]\!]\,|} \leq \#\,\mathfrak{m}(\mathrm{cc}(q^n)) \leq \frac{|Q| \times k^{n-1}}{|\,[\![\, q^n \,]\!]\,|}\,. \tag{2}$$

Let us prove that for n large enough, $|\,[\![\, q^{n+1} \,]\!]\,| < k \times |\,[\![\, q^n \,]\!]\,|$. From Proposition 11 we know that the corresponding non strict inequality is satisfied. Now, from Propositions 11 and 12 we know that for N large enough, if the equality holds at rank N, it also holds at every rank greater than N: for any $n \geq N$, $|\,[\![\, q^{n+1} \,]\!]\,| = k \times |\,[\![\, q^n \,]\!]\,|$. This means that for n large enough: $|\,[\![\, q^n \,]\!]\,| = bk^{n-1}$, where b does not depend on n. Hence by the right part of Equation (2), the minimizations of the connected powers of q have bounded size, which implies that q has finite order as seen in Section 2.1, in contradiction with the hypotheses. So the equality does not hold for any N.

Denote by ℓ the limit of the sequence

$$\left(\frac{|\,[\![\, q^{n+1} \,]\!]\,|}{|\,[\![\, q^n \,]\!]\,|}\right)_{n>0}$$

(it exists from Proposition 12): $\ell < k$ from the above paragraph, so for n large enough, there exists a constant c such that $|\,[\![\, q^n \,]\!]\,| = \frac{\ell^n}{kc}$.

So Equation (2) becomes:

$$\forall n > N,\ c\left(\frac{k}{\ell}\right)^n \leq \#\,\mathfrak{m}(\mathrm{cc}(q^n)) \leq c|Q|\left(\frac{k}{\ell}\right)^n\,. \tag{3}$$

Since $\ell < k$, there exists α such that $\left(\frac{k}{l}\right)^\alpha > |Q|$. Let us denote $\mathbf{u} = q^\alpha$ and $K = \left(\frac{k}{\ell}\right)^\alpha$, we have that for n large enough:

$$c \cdot K^n \leq \#\,\mathfrak{m}(\mathrm{cc}(\mathbf{u}^n)) \leq c \cdot |Q| \cdot K^n < c \cdot K^{n+1} \leq \#\,\mathfrak{m}(\mathrm{cc}(\mathbf{u}^{n+1})) \leq c \cdot |Q| \cdot K^{n+1}\,. \tag{4}$$

Consequently, the minimizations of the components $\mathrm{cc}(\mathbf{u}^n)$ are pairwise not isomorphic, for n large enough, because they do not have the same size. So their states induce different elements of the group $\langle\mathcal{A}\rangle$. Hence the sets

$$I_n = \{\rho_{\mathbf{v}} \mid \mathbf{v} \text{ is a state of } \mathfrak{m}(\mathrm{cc}(\mathbf{u}^n))\}$$

are pairwise disjoint. By Equation (4), the growth of the sequence $(I_n)_{n>0}$ is exponential, and so is the growth of $\mathcal{A}$. ◀

By combining Theorem 14 with the fact that connected bireversible Mealy automata of prime size cannot generate infinite Burnside groups [4], we have:

▶ **Corollary 15.** *Any infinite group generated by a bireversible connected Mealy automaton of prime size has exponential growth.*

Another consequence of Theorem 14 concerns virtually nilpotent groups. This class is important in the classification of groups and contains in particular all the abelian groups. It is known that any infinite virtually nilpotent group contains an element of infinite order [3, Proposition 10.48] and has polynomial growth [16]. This leads to

▶ **Corollary 16.** *No infinite virtually nilpotent group can be generated by a bireversible Mealy automaton.*

References

1 L. Bartholdi and A. Erschler. Groups of given intermediate word growth. *Ann. Inst. Fourier*, 64(5):2003–2036, 2014.
2 T. Brough and A.J. Cain. Automaton semigroups: New constructions results and examples of non-automaton semigroups. *Theoretical Computer Science*, 674:1–15, 2017.
3 C. Drutu and M. Kapovich. Lectures on geometric group theory. URL: `https://www.math.ucdavis.edu/~kapovich/EPR/ggt.pdf`.
4 Th. Godin and I. Klimann. On bireversible Mealy automata and the Burnside problem. *Theoretical Computer Science*, 707:24–35, 2018.
5 R. Grigorchuk. Degrees of growth of finitely generated groups and the theory of invariant means. *Izv. Akad. Nauk SSSR Ser. Mat.*, 48-5:939–985, 1984.
6 R. Grigorchuk. Degrees of growth of finitely generated groups, and the theory of invariant means. *Mathematics of the USSR-Izvestiya*, 25(2):259, 1985.
7 R. Grigorchuk and I. Pak. Groups of intermediate growth: an introduction. *L'Enseignement Mathématique*, 54(3-4):251–272, 2008.
8 R.I. Grigorchuk. On periodic groups generated by finite automata. In *18-th USSR Algebraic Conference, Kishinev*, 1985.
9 I. Klimann. Automaton semigroups: The two-state case. *Theor. Comput. Syst. (special issue STACS'13)*, pages 1–17, 2014.
10 I. Klimann. On level-transitivity and exponential growth. *Semigroup Forum*, 95(3):441–447, 2016.
11 I. Klimann, M. Picantin, and D. Savchuk. A connected 3-state reversible Mealy automaton cannot generate an infinite Burnside group. In *DLT' 15*, volume 9168 of *LNCS*, pages 313–325, 2015.
12 A. Mann. *How groups grow*, volume 395 of *Lecture Note Series*. The London Mathematical Society, 2012.
13 J. Milnor. Advanced problem 5603. *MAA Monthly*, 75:685–686, 1968.
14 V. Nekrashevich. Palindromic subshifts and simple periodic groups of intermediate growth. *Annals of Mathematics*, 187(3):667–719, 2018.
15 F. Olukoya. The growth rates of automaton groups generated by reset automata. arXiv:1708.07209, 2017.
16 J.A. Wolf. Growth of finitely generated solvable groups and curvature of riemanniann manifolds. *J. Differential Geom.*, 2(4):421–446, 1968.

On the Identity Problem for the Special Linear Group and the Heisenberg Group

Sang-Ki Ko
Korea Electronics Technology Institute, South Korea
sangkiko@keti.re.kr

Reino Niskanen
Department of Computer Science, University of Liverpool, UK
r.niskanen@liverpool.ac.uk

Igor Potapov
Department of Computer Science, University of Liverpool, UK
potapov@liverpool.ac.uk

Abstract

We study the identity problem for matrices, i.e., whether the identity matrix is in a semigroup generated by a given set of generators. In particular we consider the identity problem for the *special linear group* following recent NP-completeness result for $\mathrm{SL}(2,\mathbb{Z})$ and the undecidability for $\mathrm{SL}(4,\mathbb{Z})$ generated by 48 matrices. First we show that there is no embedding from pairs of words into 3×3 integer matrices with determinant one, i.e., into $\mathrm{SL}(3,\mathbb{Z})$ extending previously known result that there is no embedding into $\mathbb{C}^{2\times 2}$. Apart from theoretical importance of the result it can be seen as a strong evidence that the computational problems in $\mathrm{SL}(3,\mathbb{Z})$ are decidable. The result excludes the most natural possibility of encoding the Post correspondence problem into $\mathrm{SL}(3,\mathbb{Z})$, where the matrix products extended by the right multiplication correspond to the Turing machine simulation. Then we show that the *identity problem* is decidable in polynomial time for an important subgroup of $\mathrm{SL}(3,\mathbb{Z})$, the Heisenberg group $\mathrm{H}(3,\mathbb{Z})$. Furthermore, we extend the decidability result for $\mathrm{H}(n,\mathbb{Q})$ in any dimension n. Finally we are tightening the gap on decidability question for this long standing open problem by improving the undecidability result for the identity problem in $\mathrm{SL}(4,\mathbb{Z})$ substantially reducing the bound on the size of the generator set from 48 to 8 by developing a novel reduction technique.

2012 ACM Subject Classification Theory of computation → Models of computation,Computing methodologies → Symbolic and algebraic algorithms,Theory of computation → Program verification

Keywords and phrases matrix semigroup, identity problem, special linear group, Heisenberg group, decidability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.132

Related Version [27], `https://arxiv.org/abs/1706.04166`

Funding This work was supported by EPSRC grant "Reachability problems for words, matrices and maps" (EP/M00077X/1).

1 Introduction

The dynamics of many systems can be represented by matrices and matrix products. The analysis of such systems lead to solving reachability questions in matrix semigroups which is essential part in verification procedures, control theory questions, biological systems'

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 132; pp. 132:1–132:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

predictability, security etc. [9, 10, 16, 17, 20, 21, 28, 33, 35, 36, 37]. Many nontrivial algorithms for decision problems on matrix semigroups have been developed for matrices under different constraints on the dimension, the size of a generating set or for specific subclasses of matrices: e.g., commutative matrices [2], row-monomial matrices [30] or 2×2 matrix semigroups generated by non-singular integer matrices [41], upper-triangular integer matrices [25], matrices from the special linear group [4, 15], etc.

Despite visible interest in this research domain, we still see a significant lack of algorithms and complexity results for answering decision problems in matrix semigroups. Many computational problems for matrix (semi)groups are computationally hard starting from dimension two and very often become undecidable from dimensions three or four even in the case of integer matrices. The central decision problem in matrix semigroups is the membership problem, which was originally considered by A. Markov in 1947 [32]. Let $S = \langle G \rangle$ be a matrix semigroup finitely generated by a generating set of square matrices G. The *membership problem* is to decide whether or not a given matrix M belongs to the matrix semigroup S. By restricting M to be the identity matrix we call the problem the *identity problem*.

▶ **Problem 1** (Identity problem). *Let $S = \langle G \rangle$, where G is a finite set of n-dimensional matrices over $\mathbb{K} = \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}, \ldots$. Is the identity matrix in the semigroup, i.e., does $\boldsymbol{I} \in S$ hold?*

The identity problem is computationally equivalent to another fundamental problem – the *subgroup problem* (i.e., to decide whether a semigroup contains a subgroup) as any subset of matrices, which can form a product leading to the identity also generate a group [15][1].

The decidability status of the identity problem was unknown for a long time for matrix semigroups of any dimension, see Problem 10.3 in "Unsolved Problems in Mathematical Systems and Control Theory" [10], but it was shown in [6] to be undecidable for 48 matrices from $\mathbb{Z}^{4\times4}$ by proving that the identity correspondence problem (a variant of the Post correspondence problem over a group alphabet) is undecidable, and embedding pairs of words over free group alphabet into $\mathrm{SL}(4,\mathbb{Z})$ as two blocks on the main diagonal and by a morphism f as follows $f(a) = \left(\begin{smallmatrix} 1 & 2 \\ 0 & 1 \end{smallmatrix}\right)$, $f(a^{-1}) = \left(\begin{smallmatrix} 1 & -2 \\ 0 & 1 \end{smallmatrix}\right)$, $f(b) = \left(\begin{smallmatrix} 1 & 0 \\ 2 & 1 \end{smallmatrix}\right)$ and $f(b^{-1}) = \left(\begin{smallmatrix} 1 & 0 \\ -2 & 1 \end{smallmatrix}\right)$. In the seminal paper of Paterson in 1970, see [39], an injective morphism from pairs of words in alphabet $\Sigma = \{a, b\}$ into 3×3 integral matrices, $g(u, v) = \begin{pmatrix} n^{|u|} & 0 & 0 \\ 0 & n^{|v|} & 0 \\ \sigma(u) & \sigma(v) & 1 \end{pmatrix}$ (where σ represents each word as an n-adic number), was used to prove undecidability of the mortality problem (i.e., the membership problem of the zero matrix) and which later led to many undecidability results of matrix problems in dimension three, e.g., [12, 24]. Finding new injective morphisms is hard, but having them gives an opportunity to prove new undecidability results.

In 1999, Cassaigne, Harju and Karhumäki significantly boosted the research on finding algorithmic solutions for 2×2 matrix semigroups by showing that there is no injective semigroup morphism from pairs of words over any finite alphabet (with at least two elements) into complex 2×2 matrices [12]. This result led to substantial interest in finding algorithmic solutions for such problems as the identity problem, mortality, membership, vector reachability, freeness etc. for 2×2 matrices.

For example, in 2007 Gurevich and Schupp [23] showed that the membership problem is decidable in polynomial time for the finitely generated subgroups of the modular group and later in 2017 Bell, Hirvensalo and Potapov proved that the identity problem for a semigroup

[1] The product of matrices which is equal to the identity is still the identity element after a cyclic shift, so every element from this product has the inverse.

generated by matrices from $\mathrm{SL}(2,\mathbb{Z})$ is NP-complete by developing a new effective technique to operate with compressed word representations of matrices and closing the gap on complexity improving the original EXPSPACE solution proposed in 2005 [15]. The first algorithm for the membership problem which covers the cases beyond $\mathrm{SL}(2,\mathbb{Z})$ and $\mathrm{GL}(2,\mathbb{Z})$ has been proposed in [41] and provides the solution for a semigroup generated by non-singular 2×2 integer matrices. Later, these techniques have been applied to build another algorithm to solve the membership problem in $\mathrm{GL}(2,\mathbb{Z})$ extended by singular matrices [42]. The current limit of decidability is standing for 2×2 matrices which are defined over hypercomplex numbers (quaternions) for which most of the problems have been shown to be undecidable in [5] and correspond to reachability problems for 3-sphere rotation.

In our paper, we show that there is no embedding from pairs of words into 3×3 integer matrices with determinant one (i.e., into $\mathrm{SL}(3,\mathbb{Z})$), which is a strong evidence that computational problems in $\mathrm{SL}(3,\mathbb{Z})$ are decidable as all known undecidability techniques for low-dimensional matrices are based on encoding of Turing machine computations via the Post correspondence problem (PCP) which cannot be applied in $\mathrm{SL}(3,\mathbb{Z})$ following our result. In case of the PCP encoding the matrix products extended by the right multiplication correspond to the Turing machine simulation and the only known proof alternatives are recursively enumerable sets and Hilbert's tenth problem that provide undecidability for matrix equations, but of very high dimensions [3, 13, 26].

So in analogy to 1999 result from [12] on non-existence of embedding into 2×2 matrix semigroups over complex numbers, we expand a horizon of decidability area for matrix semigroups and show that there is no embedding from a set of pairs of words over a semigroup alphabet to any matrix semigroup in $\mathrm{SL}(3,\mathbb{Z})$. It follows almost immediately that there is no embedding from a set of pairs of group words into $\mathbb{Z}^{3\times 3}$.[2] The matrix semigroup in $\mathrm{SL}(3,\mathbb{Z})$ has attracted a lot of attention recently as it can be represented by a set of generators and relations [18, 19] similar to $\mathrm{SL}(2,\mathbb{Z})$ where it was possible to convert numerical problems into symbolic problems and solve them with novel computational techniques; see [4, 15, 41, 42]. Comparing to the relatively simple representation of $\mathrm{SL}(2,\mathbb{Z}) = \langle S,T \mid S^4 = \boldsymbol{I}_2, (ST)^6 = \boldsymbol{I}_2\rangle$, where $S = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$ and $T = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$ the case of $\mathrm{SL}(3,\mathbb{Z}) = \langle X,Y,Z \mid X^3 = Y^3 = Z^2 = (XZ)^3 = (YZ)^3 = (X^{-1}ZXY)^2 = (Y^{-1}ZYX)^2 = (XY)^6 = \boldsymbol{I}_3\rangle$ looks more challenging containing both non-commutative and partially commutative elements.

As the decidability status of the *identity problem* in dimension three is still a long standing open problem, we look for an important subgroup of $\mathrm{SL}(3,\mathbb{Z})$, the Heisenberg group $\mathrm{H}(3,\mathbb{Z})$, for which the *identity problem* could be decidable following our result on non-existence of embedding. The Heisenberg group is an important subgroup of $\mathrm{SL}(3,\mathbb{Z})$ which is useful in the description of one-dimensional quantum mechanical systems [11, 22, 29]. We show that the *identity problem* for a matrix semigroup generated by matrices from $\mathrm{H}(3,\mathbb{Z})$ and even $\mathrm{H}(3,\mathbb{Q})$ is decidable in polynomial time. Furthermore, we extend the decidability result for $\mathrm{H}(n,\mathbb{Q})$ in any dimension n. Moreover we tighten the gap between (un)decidability results for the *identity problem* substantially reducing the bound on the size of the generator set from 48 (see [6]) to 8 in $\mathrm{SL}(4,\mathbb{Z})$ by developing a novel reduction technique.

[2] The idea that such result may hold was motivated by analogy from combinatorial topology, where the identity problem is decidable for the braid group B_3 which is the universal central extension of the modular group $\mathrm{PSL}(2,\mathbb{Z})$ [40], an embedding for a set of pairs of words into the braid group B_5 exists, see [7], and non-existence of embeddings were proved for B_4 in [1]. So $\mathrm{SL}(3,\mathbb{Z})$ was somewhere in the goldilocks zone between B_3 and B_5.

2 Preliminaries

We say that a semigroup S is *generated* by a subset G of S if each element of S can be expressed as a composition of elements of G. In this case, we call G the *generating set* of S. Given an *alphabet* $\Sigma = \{a_1, \ldots, a_m\}$, a finite *word* u is an element of semigroup Σ^*. The *empty word* is denoted by ε. Let $\Gamma = \{a_1, \ldots, a_\ell, a_1^{-1}, \ldots, a_\ell^{-1}\}$ be a generating set of a free group $\mathrm{FG}(\Gamma)$. The elements of $\mathrm{FG}(\Gamma)$ are all *reduced* words over Γ, i.e., words not containing $a_i a_i^{-1}$ or $a_i^{-1} a_i$ as a subword. In this context, we call Γ a finite *group alphabet*, i.e., an alphabet with an involution. The multiplication of two elements (reduced words) $u, v \in \mathrm{FG}(\Gamma)$ corresponds to the unique reduced word of the concatenation uv. This multiplication is called *concatenation* throughout the paper. Later in the encoding of words over a group alphabet we denote a^{-1} by $\overline{a}$ and the alphabet of inverse letters is denoted as $\Sigma^{-1} = \{a^{-1} \mid a \in \Sigma\}$.

In the next lemma, we present an encoding from an arbitrary group alphabet to a binary group alphabet used in Section 5. The result is crucial as it allows us to present the results of the above section over the smallest domain.

▶ **Lemma 2** (Birget, Margolis [8]). *Let $\Gamma = \{z_1, \ldots, z_\ell, \overline{z_1}, \ldots, \overline{z_\ell}\}$ be a group alphabet and $\Gamma_2 = \{c, d, \bar{c}, \bar{d}\}$ be a binary group alphabet. Define the mapping $\alpha : \Gamma \to \mathrm{FG}(\Gamma_2)$ by $\alpha(z_i) = c^i d \bar{c}^i$, and $\alpha(\overline{z_i}) = c^i \bar{d} \bar{c}^i$, where $1 \leq i \leq \ell$. Then α is a monomorphism, that is, an injective morphism. Note that α can be extended to domain $\mathrm{FG}(\Gamma)$ in the usual way.*

The special linear group is $\mathrm{SL}(n, \mathbb{K}) = \{M \in \mathbb{K}^{n \times n} \mid \det(M) = 1\}$, where $\mathbb{K} = \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}, \ldots$. The *identity matrix* is denoted by $\boldsymbol{I}_n$ and the *zero matrix* is denoted by $\mathbf{0}_n$. The *Heisenberg group* $\mathrm{H}(3, \mathbb{K})$ is formed by the 3×3 matrices of the form $M = \left(\begin{smallmatrix} 1 & a & c \\ 0 & 1 & b \\ 0 & 0 & 1 \end{smallmatrix}\right)$, where $a, b, c \in \mathbb{K}$. It is easy to see that the Heisenberg group is a non-commutative subgroup of $\mathrm{SL}(3, \mathbb{K})$. We can consider the Heisenberg group as a set of all triples with the following group law: $(a_1, b_1, c_1) \otimes (a_2, b_2, c_2) = (a_1 + a_2, b_1 + b_2, c_1 + c_2 + a_1 b_2)$. By $\psi(M)$ we denote the triple $(a, b, c) \in \mathbb{K}^3$ which corresponds to the upper-triangular coordinates of M. Let M be a matrix in $\mathrm{H}(3, \mathbb{K})$ such that $\psi(M) = (a, b, c)$. We define the *superdiagonal vector* of M to be $\vec{v}(M) = (a, b)$. Given two vectors $\mathbf{u} = (u_1, u_2)$ and $\mathbf{v} = (v_1, v_2)$, the *cross product* of $\mathbf{u}$ and $\mathbf{v}$ is defined as $\mathbf{u} \times \mathbf{v} = u_1 v_2 - u_2 v_1$. Two vectors are *parallel* if the cross product is zero.

The Heisenberg group can be also defined in higher dimensions. The Heisenberg group of dimension n over $\mathbb{K}$ is denoted by $\mathrm{H}(n, \mathbb{K})$ and is the group of square matrices in $\mathbb{K}^{n \times n}$ of the form $\left(\begin{smallmatrix} 1 & \mathbf{a}^\mathsf{T} & c \\ 0 & \boldsymbol{I}_{n-2} & \mathbf{b} \\ 0 & 0 & 1 \end{smallmatrix}\right)$, where $\mathbf{a}, \mathbf{b} \in \mathbb{K}^{n-2}, c \in \mathbb{K}$. Similar to the Heisenberg group in dimension three, we can also consider the Heisenberg group in dimension n for any integer $n \geq 3$ as a set of all triples with the following group law: $(\mathbf{a}_1, \mathbf{b}_1, c_1) \otimes (\mathbf{a}_2, \mathbf{b}_2, c_2) = (\mathbf{a}_1 + \mathbf{a}_2, \mathbf{b}_1 + \mathbf{b}_2, c_1 + c_2 + \mathbf{a}_1 \cdot \mathbf{b}_2)$, where $\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_1, \mathbf{b}_2 \in \mathbb{K}^{n-2}$ and $\mathbf{a}_1 \cdot \mathbf{b}_2$ is the dot product of vectors $\mathbf{a}_1$ and $\mathbf{b}_2$.

We extend the function ψ to n-dimensional Heisenberg group: For a matrix M, $\psi(M)$ is the triple $(\mathbf{a}, \mathbf{b}, c) \in (\mathbb{K}^{n-2})^2 \times \mathbb{K}$ which corresponds to the upper-triangular coordinates of M. The product $M_1 M_2$ has $c_1 + c_2 + \mathbf{a}_1 \cdot \mathbf{b}_2$ in the upper-right corner whereas $M_2 M_1$ has $c_1 + c_2 + \mathbf{a}_2 \cdot \mathbf{b}_1$. The other coordinates are identical as we add numbers in the same coordinate. Clearly, the two products are equivalent if and only if $\mathbf{a}_1 \cdot \mathbf{b}_2 = \mathbf{a}_2 \cdot \mathbf{b}_1$ holds.

▶ **Lemma 3.** *Let M_1 and M_2 be two matrices from the Heisenberg group $\mathrm{H}(n, \mathbb{K})$ and $\psi(M_i) = (\mathbf{a}_i, \mathbf{b}_i, c_i)$ for $i = 1, 2$. Then $M_1 M_2 = M_2 M_1$ holds if and only if $\mathbf{a}_1 \cdot \mathbf{b}_2 = \mathbf{a}_2 \cdot \mathbf{b}_1$.*[3]

[3] Note that, in dimension three, the condition can be stated as superdiagonal vectors of M_1 and M_2 being parallel.

3 On embedding from pairs of words into $\mathrm{SL}(3,\mathbb{K})$

In this section, we show that there is no embedding from a set of pairs of words over a semigroup alphabet to the special linear group $\mathrm{SL}(3,\mathbb{Z})$, which can be seen as a strong evidence about decidability of computational problems for this class. If the injective morphism would exist, then we could encode Turing machine computations via the PCP which would provide undecidability proofs for various membership problems in $\mathrm{SL}(3,\mathbb{Z})$.

Let $\Sigma = \{0,1\}$. The monoid $\Sigma^* \times \Sigma^*$ has a generating set $S = \{(0,\varepsilon),(1,\varepsilon),(\varepsilon,0),(\varepsilon,1)\}$, where ε is the empty word. We simplify the notation by setting $a = (0,\varepsilon)$, $b = (1,\varepsilon)$, $c = (\varepsilon,0)$ and $d = (\varepsilon,1)$. It is easy to see that we have the following relations:

$$ac = ca, \qquad bc = cb, \qquad ad = da, \qquad bd = db. \tag{1}$$

In other words, a and b commute with c and d. Furthermore, these are the only relations. That is, a and b do not commute with each other, and neither do c and d. The monoid $\Sigma^* \times \Sigma^*$ is a partially commutative monoid or a trace monoid. A necessary and sufficient conditions for existence of an embedding of trace monoids into $\mathbb{N}^{2\times 2}$ was given in [14] but, to the authors' best knowledge, there are no similar results even for $\mathbb{N}^{3\times 3}$. Let $\varphi : \Sigma^* \times \Sigma^* \to \mathrm{SL}(3,\mathbb{K})$ be an injective morphism and denote $A = \varphi(a)$, $B = \varphi(b)$, $C = \varphi(c)$ and $D = \varphi(d)$. Our goal is to show that φ does not exist for $\mathbb{K} = \mathbb{Z}$. Additionally, we provide an embedding for $\mathbb{K} = \mathbb{Q}$. Unfortunately, the technique developed in [12], where the contradiction was derived from simple relations, resulting from matrix multiplication, cannot be used for a case of $\mathrm{SL}(3,\mathbb{Z})$ as it creates a large number of equations which do not directly limit the existence of φ. We found new techniques to show non-existence of φ by analysis of eigenvalues and the Jordan normal forms.

In the next theorem, we show that if we embed the generators of $\Sigma^* \times \Sigma^*$ into $\mathrm{SL}(3,\mathbb{Z})$, then, for each Jordan normal form, the matrices should satisfy additional equations.

▶ **Theorem 4.** *There is no injective morphism $\varphi : \Sigma^* \times \Sigma^* \to \mathrm{SL}(3,\mathbb{Z})$ for any $|\Sigma| \geq 2$.*

Proof (Sketch). Due to the obvious symmetries, it is enough to show that the claim holds for A. We conjugate the generators to transform A into Jordan normal form. Note that Jordan normal form J of an integer matrix A does not have to be integer or even real, but the contradictions we derive apply also to the original matrices. Also note that matrices J and A have integer trace and determinants are one. There are six possible Jordan normal forms for 3×3 matrices: $\left(\begin{smallmatrix}\lambda&0&0\\0&\mu&0\\0&0&\nu\end{smallmatrix}\right)$, $\left(\begin{smallmatrix}\lambda&0&0\\0&\mu&0\\0&0&\mu\end{smallmatrix}\right)$, $\left(\begin{smallmatrix}\lambda&0&0\\0&\mu&1\\0&0&\mu\end{smallmatrix}\right)$, $\left(\begin{smallmatrix}\lambda&0&0\\0&\lambda&0\\0&0&\lambda\end{smallmatrix}\right)$, $\left(\begin{smallmatrix}\lambda&1&0\\0&\lambda&0\\0&0&\lambda\end{smallmatrix}\right)$, or $\left(\begin{smallmatrix}\lambda&1&0\\0&\lambda&1\\0&0&\lambda\end{smallmatrix}\right)$. The first and the fourth normal forms can be ruled out as the matrices commute with diagonal matrices, and then we prove that C and D commute with each other. In the second form, it follows from the fact that A has an integer trace and the determinant is one, that the eigenvalues are $\lambda = 2$ and $\mu = \frac{1}{2}$. Then, we can rule this form out as also the trace of A^2 should be an integer which does not hold for these eigenvalues. The third form is ruled out as from the relations $AC = CA$ and $AD = DA$, we can solve C and D, and see that necessary also $CD = DC$ holds. The final form is similar to the third Jordan normal form and is ruled out in a similar manner. The fifth form requires additional considerations. We solve most of the elements of $C = \left(\begin{smallmatrix}a&b&c\\d&e&f\\g&h&\ell\end{smallmatrix}\right)$ and $D = \left(\begin{smallmatrix}a'&b'&c'\\d'&e'&f'\\g'&h'&\ell'\end{smallmatrix}\right)$ from equations $AC = CA$ and $AD = DA$. To solve the remaining elements, we need to do further case analysis and we prove that matrices C and D do not commute if and only if $ch' \neq c'h$. We further solve B from $BC = CB$ and $BD = DB$, and show that necessarily $CD = DC$, which is not a valid relation. This is a brief sketch of high-level steps of showing that the injective morphism does not exist; see the complete proof in [27]. ◀

Note that some of the Jordan normal forms of the previous theorem can be ruled out even without assuming that the original matrices were in $\mathrm{SL}(3,\mathbb{Z})$. Using these additional constraints on matrices, we are able to find an embedding into $\mathrm{SL}(3,\mathbb{Q})$.

▶ **Theorem 5.** *Let* $\Sigma = \{0,1\}$. *The morphism* $\varphi : \Sigma^* \times \Sigma^* \to \mathrm{SL}(3,\mathbb{Q})$, *defined by* $\varphi((0,\varepsilon)) = \begin{pmatrix} 4 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{pmatrix}$, $\varphi((1,\varepsilon)) = \begin{pmatrix} 9 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{3} \end{pmatrix}$, $\varphi((\varepsilon,0)) = \begin{pmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 4 \end{pmatrix}$ *and* $\varphi((\varepsilon,1)) = \begin{pmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & 9 \end{pmatrix}$ *is an embedding.*

We can further extend the non-existence result to words over group alphabets. The only known results for undecidability of the identity problem rely on embedding of pairs of group words into matrices (Theorem 13 and [6]) suggesting that the problem might be decidable in dimension three over integers.

▶ **Corollary 6.** *There is no injective morphism* $\varphi : \mathrm{FG}(\Gamma) \times \mathrm{FG}(\Gamma) \to \mathbb{Z}^{3\times 3}$ *for any binary group alphabet* Γ.

Proof. We proceed by contradiction. Assume that there exists such an injective morphism φ from the set of pairs of words over a group alphabet to the set of matrices in $\mathbb{Z}^{3\times 3}$. Suppose that $A = \varphi((a,\varepsilon))$, where $a \in \Gamma$. Then the inverse matrix A^{-1} corresponding to $(\bar{a},\varepsilon)$ must be in $\mathbb{Z}^{3\times 3}$. This implies that the determinant of A is ± 1 because otherwise the determinant of A^{-1} becomes a non-integer. Consider then a morphism ψ such that $\psi(x) = \varphi(x)\varphi(x)$ for each $x \in \mathrm{FG}(\Gamma) \times \mathrm{FG}(\Gamma)$. It is clear that also ψ is injective and that the determinant of the image is 1. By Theorem 4, such injective morphism ψ does not exist even from semigroup alphabets and hence neither does φ. ◀

4 Decidability of the identity problem in the Heisenberg group

In this section, we prove that the identity problem is decidable for the Heisenberg group which is an important subgroup of the special linear group. First, we provide more intuitive solution for dimension three, i.e., $\mathrm{H}(3,\mathbb{Q})$, which still requires a number of techniques to estimate possible values of elements under permutations in matrix products. In the end of the section, we generalize the result for $\mathrm{H}(n,\mathbb{Q})$ using analogies in the solution for $\mathrm{H}(3,\mathbb{Q})$.

We prove that the identity problem for the Heisenberg group over rationals is decidable by analysing the behaviour of multiplications especially in the upper-right coordinate of matrices. From Lemma 3, it follows that the matrix multiplication is commutative in the Heisenberg group if and only if matrices have pairwise parallel superdiagonal vectors. So we analyse two cases of products for matrices with pairwise parallel and none pairwise parallel superdiagonal vectors and then provide algorithms that solve the problem in polynomial time. The most difficult part is showing that only limited number of conditions must be checked to guarantee the existence of a product that results in the identity.

▶ **Lemma 7.** *Let* $G = \{M_1,\ldots,M_r\} \subseteq \mathrm{H}(3,\mathbb{Q})$ *be a set of matrices from the Heisenberg group such that superdiagonal vectors of matrices are pairwise parallel. If there exists a sequence of matrices* $M = M_{i_1}\cdots M_{i_k}$, *where* $i_j \in [1,r]$ *for all* $1 \le j \le k$, *such that* $\psi(M) = (0,0,c)$ *for some* $c \in \mathbb{Q}$, *then,* $c = \sum_{j=1}^{k}(c_{i_j} - \frac{q}{2}a_{i_j}^2)$ *for some* $q \in \mathbb{Q}$ *dependent only on* G.

Proof. Consider the sequence $M_{i_1}\cdots M_{i_k}$ and let $M_i = \begin{pmatrix} 1 & a_i & c_i \\ 0 & 1 & b_i \\ 0 & 0 & 1 \end{pmatrix}$ for each $i \in [1,r]$. Since the superdiagonal vectors are parallel, i.e., $a_i b_j = b_i a_j$ for any $i,j \in [1,r]$, we have $q = \frac{b_i}{a_i} \in \mathbb{Q}$

and thus $a_i q = b_i$ for all $i \in [1, r]$. Let us consider the product of the matrices. Then the value c is equal to

$$c = \sum_{j=1}^{k} c_{i_j} + \sum_{\ell=1}^{k-1} \left(\sum_{j=1}^{\ell} a_{i_j} \right) a_{i_{\ell+1}} q = \sum_{j=1}^{k} c_{i_j} + \frac{q}{2} \left(\sum_{\ell=1}^{k} \sum_{j=1}^{k} a_{i_\ell} a_{i_j} - \sum_{j=1}^{k} a_{i_j}^2 \right) = \sum_{j=1}^{k} (c_{i_j} - \frac{q}{2} a_{i_j}^2).$$

Note that if $a_{i_j} = 0$ for some $i_j \in [1, r]$, then due to superdiagonal vectors being parallel, $a_{i_j} = 0$ for all i_j and the value c is equal to $\sum_{j=1}^{k} c_{i_j}$. ◀

Note that the previous lemma also holds for $\mathrm{H}(3, \mathbb{R})$. From the previous lemma we further see that the value c is preserved if the matrices are reordered due to their commutativity.

It is worth mentioning that the identity problem in the Heisenberg group is decidable if any two matrices have pairwise parallel superdiagonal vectors since now the problem reduces to solving a system of two linear homogeneous Diophantine equations. Hence, it remains to consider the case when there exist two matrices with non-parallel superdiagonal vectors in the sequence generating the identity matrix. In the following, we prove that the identity matrix is always constructible if we can construct any matrix with the zero superdiagonal vector by using matrices with non-parallel superdiagonal vectors.

▶ **Lemma 8.** *Let $S = \langle M_1, \ldots, M_r \rangle \subseteq \mathrm{H}(3, \mathbb{Q})$ be a finitely generated matrix semigroup. Then the identity matrix exists in S if there exists a sequence of matrices $M_{i_1} \cdots M_{i_k}$, where $i_j \in [1, r]$ for all $1 \leq j \leq k$, satisfying the following properties:*

(i) *$\psi(M_{i_1} \cdots M_{i_k}) = (0, 0, c)$ for some $c \in \mathbb{Q}$, and*

(ii) *$\vec{v}(M_{i_{j_1}})$ and $\vec{v}(M_{i_{j_2}})$ are not parallel for some $j_1, j_2 \in [1, k]$.*

To prove Lemma 8, we show that from a matrix $M = M_{i_1} \cdots M_{i_k}$, such that $\psi(M) = (0, 0, c)$, satisfying the conditions of the lemma, we can construct a matrix M' such that $\psi(M') = (0, 0, c')$ and $cc' < 0$. Given that M_i is the ith generator and $\psi(M_i) = (a_i, b_i, c_i)$, we have $\sum_{j=1}^{k} a_{i_j} = 0$ and $\sum_{j=1}^{k} b_{i_j} = 0$. Without loss of generality, $c > 0$, and the following also holds:

$$c = \sum_{\ell=1}^{k-1} \sum_{j=1}^{\ell} a_{i_j} b_{i_{\ell+1}} + \sum_{j=1}^{k} c_{i_j} > 0. \tag{2}$$

If the matrix semigroup $S \subseteq \mathrm{H}(3, \mathbb{Q})$ has two different matrices N_1 and N_2 such that $\psi(N_1) = (0, 0, c_1)$ and $\psi(N_2) = (0, 0, c_2)$ and $c_1 c_2 < 0$, then the identity matrix exists in S. Indeed, let $\psi(N_1) = (0, 0, \frac{p_1}{q_1})$ and $\psi(N_2) = (0, 0, \frac{p_2}{q_2})$, where $p_1, q_1, q_2 \in \mathbb{Z}$ are positive and $p_2 \in \mathbb{Z}$ is negative. Then it is easy to see that the matrix $N_1^{-q_1 p_2} N_2^{q_2 p_1}$ exists in S and that $\psi(N_1^{-q_1 p_2} N_2^{q_2 p_1}) = (0, 0, 0)$.

To construct the matrix M', we first classify the matrices into four types as follows. A matrix with a superdiagonal vector (a, b) is classified as

1) the $(+,+)$-type if $a, b > 0$,
2) the $(+,-)$-type if $a \geq 0$ and $b \leq 0$,
3) the $(-,-)$-type if $a, b < 0$, and
4) the $(-,+)$-type if $a < 0$ and $b > 0$.

Let $G = \{M_1, \ldots, M_r\}$ be the generating set of the matrix semigroup S. Then $G = G_{(+,+)} \sqcup G_{(+,-)} \sqcup G_{(-,-)} \sqcup G_{(-,+)}$ such that $G_{(\xi_1, \xi_2)}$ is the set of matrices of the (ξ_1, ξ_2)-type, where $\xi_1, \xi_2 \in \{+, -\}$.

Recall that we assume $M = M_{i_1} \cdots M_{i_k}$ and $\psi(M) = (0, 0, c)$ for some $c > 0$. The main idea of the proof is to generate a matrix M' such that $\psi(M') = (0, 0, c')$ for some $c' < 0$ by duplicating the matrices in the sequence $M = M_{i_1} \cdots M_{i_k}$ multiple times and

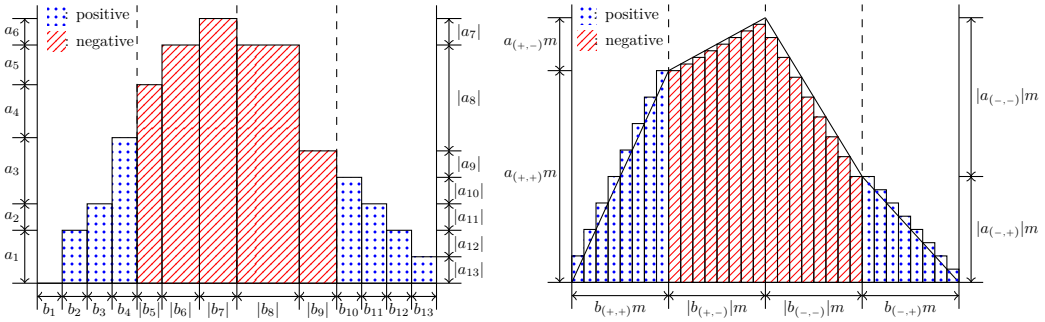


Figure 1 The histogram on the left describes how the upper-right corner of $M_1 \cdots M_{13}$ is computed by multiplications. The blue dotted (red lined) area implies the value which will be added to (subtracted from) the upper-right corner of the final matrix after multiplications of matrices in the sequence. The histogram on the right describes how the upper-right corner of $M^m_{(+,+)}M^m_{(+,-)}M^m_{(-,-)}M^m_{(-,+)}$ is computed by multiplications. Here $m = 8$.

reshuffling. Note that any permutation of the sequence generating the matrix M such that $\psi(M) = (0, 0, c)$ still generates matrices M' such that $\psi(M') = (0, 0, c')$ since the multiplication of matrices exchanges the first two coordinates in a commutative way. Also note that there exists a permutation such that $c \neq c'$ as we assumed that at least two matrices in the sequence do not commute. Moreover, we can still obtain matrices M'' such that $\psi(M'') = (0, 0, c'')$ for some $c'' \in \mathbb{Q}$ if we shuffle two different permutations of the sequence $M_{i_1} \cdots M_{i_k}$ by the same reason.

Let us illustrate the idea with the following example. See Figure 1 for pictorial descriptions of the idea. Let $\{M_i \mid 1 \leq i \leq 4\} \subseteq G_{(+,+)}$, $\{M_i \mid 5 \leq i \leq 7\} \subseteq G_{(+,-)}$, $\{M_i \mid 8 \leq i \leq 9\} \subseteq G_{(-,-)}$, and $\{M_i \mid 10 \leq i \leq 13\} \subseteq G_{(-,+)}$. Then assume that $M_1 \cdots M_{13} = \left(\begin{smallmatrix} 1 & 0 & x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{smallmatrix}\right)$, where x is computed by (2). As we mentioned above, x changes if we change the order of multiplicands. In this example, we first multiply $(+,+)$-type matrices and accumulate the values in the superdiagonal coordinates since these matrices have positive values in the coordinates. Indeed, the blue dotted area implies the value we add to the upper-right corner by multiplying such matrices. Then we multiply $(+,-)$-type matrices and still increase the 'a'-value. The 'b'-values in $(+,-)$-type matrices are negative thus, the red lined area is subtracted from the upper-right corner. We still subtract by multiplying $(-,-)$-type matrices since the accumulated 'a'-value is still positive and 'b'-values are negative. We finish the multiplication by adding exactly the last blue dotted area to the upper-right corner. It is easy to see that the total subtracted value is larger than the total added value.

However, we cannot guarantee that x is negative since $\sum_{i=1}^{13} c_i$ could be larger than the contribution from the superdiagonal coordinates. This is why we need to copy the sequence of matrices generating the matrix corresponding to the triple $(0, 0, c)$ for some $c \in \mathbb{Q}$. In Figure 1, we describe an example where we duplicate the sequence eight times and shuffle and permute it in order to minimize the value in the upper-right corner. Now the lengths of both axes are m ($m = 8$ in this example) times larger than before and it follows that the area also grows quadratically in m. Since the summation $m \cdot \sum_{i=1}^{13} c_i$ grows linearly in m, we have $x < 0$ for large enough m. In [27], we formally prove this by bounding contributions of each matrix type and showing that the coefficient of the highest power of m is negative.

It should be noted that there are some subcases where some matrix types are not present in the product. In each case we need to show that the same idea can be used to construct a matrix M' such that $\psi(M') = (0, 0, c')$, where $c' < 0$. The full analysis of all cases can be found in [27].

▶ **Theorem 9.** *The identity problem for a semigroup generated by matrices from* $\mathrm{H}(3,\mathbb{Q})$ *is in polynomial time.*

Proof. Let S be the matrix semigroup in $\mathrm{H}(3,\mathbb{Q})$ generated by the set $G = \{M_1, \ldots, M_r\}$. There are two possible cases of having the identity matrix in the matrix semigroup in $\mathrm{H}(3,\mathbb{Q})$. Either the identity matrix is generated by a product of matrices where all superdiagonal vectors are parallel or there are at least two matrices with non-parallel superdiagonal vectors.

Consider the first case. Lemma 7 provides a formula to compute the value in the top corner regardless of the order of the multiplications. That is, we need to solve a system of linear homogeneous Diophantine equations with solutions over non-negative integers. We partition the set G into several disjoint subsets $G_1, \ldots, G_s$, where s is at most r, and each subset contains matrices with parallel superdiagonal vectors. Since superdiagonal vectors being parallel is a transitive and symmetric property, each matrix needs to be compared to a representative of each subset. If there are no matrices with parallel superdiagonal vectors, then there are r subsets G_i containing exactly one matrix and $O(r^2)$ tests were done. Let us consider $G_i = \{M_{k_1}, \ldots, M_{k_{s_i}}\}$, i.e., one of the subsets containing s_i matrices and $\psi(M_{k_j}) = (a_{k_j}, b_{k_j}, c_{k_j})$. By Lemma 7, the value $c_{k_j} - \frac{q_i}{2}a_{k_j}^2$, for a fixed $q_i \in \mathbb{Q}$, is added to the top corner when matrix M_{k_j} is multiplied.

We solve the system of two linear homogeneous Diophantine equations $A\mathbf{y} = \mathbf{0}$, where

$$A = \begin{pmatrix} a_{k_1} & a_{k_2} & \cdots & a_{k_{s_i}} \\ c_{k_1} - \frac{q_i}{2}a_{k_1}^2 & c_{k_2} - \frac{q_i}{2}a_{k_2}^2 & \cdots & c_{k_{s_i}} - \frac{q_i}{2}a_{k_{s_i}}^2 \end{pmatrix}$$

and $\mathbf{y}^\mathsf{T} \in \mathbb{N}^{s_i}$. The first row is the constraint that guarantees that the first component of the superdiagonal is zero in the matrix product constructed from a solution. Since the superdiagonal vectors are parallel, it also implies that the whole vector is zero. The second row guarantees that the upper corner is zero.

It is obvious that the identity matrix is in the semigroup if we have a solution in the system of two linear homogeneous Diophantine equations for any subset G_i. That is, we need to solve at most r systems of two linear homogeneous Diophantine equations.

Next, we consider the second case, where by Lemma 8, it is enough to check whether there exists a sequence of matrices generating a matrix with zero superdiagonal vector and containing two matrices with non-parallel superdiagonal vectors. Let us say that $M_{i_1}, M_{i_2} \in G$, where $1 \leq i_1, i_2 \leq r$ are the two matrices. Recall that $G = \{M_1, \ldots, M_r\}$ is a generating set of the matrix semigroup and let $\psi(M_i) = (a_i, b_i, c_i)$ for all $1 \leq i \leq r$. We can see that there exists such a product containing the two matrices by solving a system of two linear homogeneous Diophantine equations of the form $B\mathbf{y} = \mathbf{0}$, where $B = \left(\begin{smallmatrix} a_1 & a_2 & \cdots & a_r \\ b_1 & b_2 & \cdots & b_r \end{smallmatrix}\right)$, with an additional constraint that the numbers in the solution $\mathbf{y}$ that correspond to M_{i_1} and M_{i_2} are non-zero since we must use these two matrices in the product. We repeat this process at most $r(r-1)$ times until we find a solution. Therefore, the problem reduces again to solving at most $O(r^2)$ systems of two linear homogeneous Diophantine equations.

Finally, we conclude the proof by mentioning that the identity problem for matrix semigroups in the Heisenberg group over rationals $\mathrm{H}(3,\mathbb{Q})$ can be decided in polynomial time as the problem of existence of a positive integer solution to a system of linear homogeneous Diophantine equations is in polynomial time. Note that if the system is non-homogeneous, then solvability of a system of linear Diophantine equations with solutions over positive integers is an NP-complete problem; see for example [38]. Indeed, a system of linear homogeneous Diophantine equations with solutions over non-negative integers can be converted to a linear programming problem with a solution over rationals which is known to be solvable in polynomial time; see e.g., [43]. It is easy to add additional constraints to the linear programming

that ensure that solutions are positive and non-zero. As the system is homogeneous, any solution can be converted to an integer solution by multiplying by the denominators. ◀

Next, we generalize the above algorithm for the identity problem in the Heisenberg group $\mathrm{H}(3,\mathbb{Q})$ to the domain of the Heisenberg groups for any dimension over the rational numbers. Similarly to the case of dimension three, we establish the following result for the case of matrices where multiplication is commutative.

▶ **Lemma 10.** *Let $G = \{M_1, \ldots, M_r\} \subseteq \mathrm{H}(n,\mathbb{Q})$ be a set of matrices from the Heisenberg group such that $\psi(M_i) = (\mathbf{a}_i, \mathbf{b}_i, c_i)$ and $\psi(M_j) = (\mathbf{a}_j, \mathbf{b}_j, c_j)$ and $\mathbf{a}_i \cdot \mathbf{b}_j = \mathbf{a}_j \cdot \mathbf{b}_i$ for any $1 \leq i \neq j \leq r$. If there exists a sequence of matrices $M = M_{i_1} \cdots M_{i_k}$, where $i_j \in [1,r]$ for all $1 \leq j \leq k$, such that $\psi(M) = (\mathbf{0}, \mathbf{0}, c)$ for some $c \in \mathbb{Q}$, then $c = \sum_{j=1}^{k}(c_{i_j} - \frac{1}{2}\mathbf{a}_{i_j} \cdot \mathbf{b}_{i_j})$.*

Lemma 8 does not generalize to $\mathrm{H}(n,\mathbb{Q})$ in the same way as we cannot classify matrices according to types to control the value in upper-right corner, so we use a different technique to prove that the value in the upper corner will be diverging to both positive and negative infinity quadratically as we repeat the same sequence generating any matrix M such that $\psi(M) = (\mathbf{0}, \mathbf{0}, c)$.

▶ **Lemma 11.** *Let $S = \langle M_1, \ldots, M_r\rangle \subseteq \mathrm{H}(n,\mathbb{Q})$ be a finitely generated matrix semigroup. Then the identity matrix exists in S if there exists a sequence of matrices $M_{i_1} \cdots M_{i_k}$, where $i_j \in [1,r]$ for all $1 \leq j \leq k$, satisfying the following properties:*

(i) *$\psi(M_{i_1} \cdots M_{i_k}) = (\mathbf{0}, \mathbf{0}, c)$ for some $c \in \mathbb{Q}$, and*

(ii) *$\mathbf{a}_{i_{j_1}} \cdot \mathbf{b}_{i_{j_2}} \neq \mathbf{a}_{i_{j_2}} \cdot \mathbf{b}_{i_{j_1}}$ for some $j_1, j_2 \in [1,k]$, where $\psi(M_i) = (\mathbf{a}_i, \mathbf{b}_i, c_i)$ for $1 \leq i \leq r$.*

Proof. From the first property claimed in the lemma, we know that any permutation of the sequence of matrix multiplications of $M_{i_1} \cdots M_{i_k}$ results in matrices M' such that $\psi(M') = (\mathbf{0}, \mathbf{0}, y)$ for some $y \in \mathbb{Q}$ since the multiplication of matrices in the $\mathrm{H}(n,\mathbb{Q})$ performs additions of vectors which is commutative in the top row and the rightmost column excluding the upper-right corner. From the commutative behaviour in the horizontal and vertical vectors of matrices in the Heisenberg group, we also know that if we duplicate the matrices in the sequence $M_{i_1} \cdots M_{i_k}$ and multiply the matrices in any order, then the resulting matrix has a non-zero coordinate in the upper triangular coordinates only in the upper right corner.

Now let $j_1, j_2 \in [1,k]$ be two indices such that $\mathbf{a}_{i_{j_1}} \cdot \mathbf{b}_{i_{j_2}} \neq \mathbf{a}_{i_{j_2}} \cdot \mathbf{b}_{i_{j_1}}$ as claimed in the lemma. Then consider the following matrix M_d that can be obtained by duplicating the sequence $M_{i_1} \cdots M_{i_k}$ of matrices into ℓ copies and shuffling the order as follows: $M_d = M_{i_{j_1}}^{\ell} M_{i_{j_2}}^{\ell} M_x^{\ell}$, where M_x is a matrix that is obtained by multiplying the matrices in $M_{i_1} \cdots M_{i_k}$ except the two matrices M_{j_1} and M_{j_2}. Then it is clear that $\psi(M_d) = (\mathbf{0}, \mathbf{0}, d)$ for some d. Let $\psi(M_x) = (\mathbf{a}_x, \mathbf{b}_x, c_x)$. Then it is easy to see that $\mathbf{a}_{i_{j_1}} + \mathbf{a}_{i_{j_2}} + \mathbf{a}_x = \mathbf{0}$ and $\mathbf{b}_{i_{j_1}} + \mathbf{b}_{i_{j_2}} + \mathbf{b}_x = \mathbf{0}$. Consider then a product, where order of M_{j_1} and M_{j_2} is swapped. That is, $M_e = M_{i_{j_2}}^{\ell} M_{i_{j_1}}^{\ell} M_x^{\ell}$ and let us denote $\psi(M_e) = (\mathbf{0}, \mathbf{0}, e)$ for some $e \in \mathbb{Q}$. By solving values d and e, we notice that the coefficient of ℓ^2 in d is $\mathbf{a}_{i_{j_1}} \cdot \mathbf{b}_{i_{j_2}} - \mathbf{a}_{i_{j_2}} \cdot \mathbf{b}_{i_{j_1}}$ and in e is $\mathbf{a}_{i_{j_2}} \cdot \mathbf{b}_{i_{j_1}} - \mathbf{a}_{i_{j_1}} \cdot \mathbf{b}_{i_{j_2}}$. As we assumed that $\mathbf{a}_{i_{j_1}} \cdot \mathbf{b}_{i_{j_2}} \neq \mathbf{a}_{i_{j_2}} \cdot \mathbf{b}_{i_{j_1}}$, the coefficients are of different sign. Hence, for sufficiently large ℓ, the values d and e have opposite signs. Then, as in the proof Lemma 8, the identity matrix always exists in the semigroup as we can multiply these two matrices correct number of times to have zero in the upper right coordinate as well. ◀

Next, we prove that the identity problem is decidable for n-dimensional Heisenberg matrices. In contrast to Theorem 9, we do not claim that the problem is decidable in polynomial time since one of the steps of the proof is to partition matrices according to dot products which cannot be extended to higher dimensions than three. For higher dimensions,

partitioning matrices according to dot products takes an exponential time in the number of matrices in the generating set. Note that if the size of the generating set is fixed, i.e., only the matrices are part of the input, then the problem remains in P.

▶ **Theorem 12.** *The identity problem for a semigroup generated by matrices from* $\mathrm{H}(n,\mathbb{Q})$ *is decidable.*

Proof. Similarly to the proof of Theorem 9, there are two ways the identity matrix can be generated. Either all the matrices commute or there are at least two matrices that do not commute. Let S be the matrix semigroup in $\mathrm{H}(n,\mathbb{Q})$ generated by the set $G = \{M_1,\ldots,M_r\}$. Consider matrices N_1, N_2 and N_3, such that $\psi(N_1) = (\mathbf{a}_1,\mathbf{b}_1,c_1)$, $\psi(N_2) = (\mathbf{a}_2,\mathbf{b}_2,c_2)$ and $\psi(N_3) = (\mathbf{a}_3,\mathbf{b}_3,c_3)$. If $\mathbf{a}_1\cdot\mathbf{b}_2 = \mathbf{a}_2\cdot\mathbf{b}_1$ and $\mathbf{a}_2\cdot\mathbf{b}_3 = \mathbf{a}_3\cdot\mathbf{b}_2$, it does not imply that $\mathbf{a}_1\cdot\mathbf{b}_3 = \mathbf{a}_3\cdot\mathbf{b}_1$. Therefore, the number of subsets of G, where each subset contains matrices that commute with other matrices in the same subset, is exponential in r as two subsets are not necessarily disjoint. Now we examine whether it is possible to generate the identity matrix by multiplying matrices in each subset by Lemma 10. If it is not possible, we need to consider the case of having two matrices that do not commute with each other in the product with zero values in the upper-triangular coordinates except the corner. Let us say that $M_{i_1}, M_{i_2} \in G$, where $1 \le i_1, i_2 \le r$ are the two matrices. Recall that $G = \{M_1,\ldots,M_r\}$ is a generating set of the matrix semigroup and let $\psi(M_i) = (\mathbf{a}_i,\mathbf{b}_i,c_i)$ for all $1 \le i \le r$.

Then we can see that there exists such a product by solving a system of $2(n-2)$ linear homogeneous Diophantine equations of the form $B\mathbf{y} = \mathbf{0}$, where $B = \begin{pmatrix} \mathbf{a}_1^{\mathsf{T}}\cdots\mathbf{a}_r^{\mathsf{T}} \\ \mathbf{b}_1^{\mathsf{T}}\cdots\mathbf{b}_r^{\mathsf{T}} \end{pmatrix}$, with an additional constraint that the values in the solution $\mathbf{y}$ that correspond to M_{i_1} and M_{i_2} are non-zero since we must use these two matrices in the product. We repeat this process at most $r(r-1)$ times until we find a solution. Hence, we can view the identity problem in $\mathrm{H}(n,\mathbb{Q})$ as the problem of solving systems of $2(n-2)$ linear homogeneous Diophantine equations with some constraints on the solution. As we can solve systems of linear homogeneous Diophantine equations, we conclude that the identity problem in $\mathrm{H}(n,\mathbb{Q})$ is also decidable. ◀

5 The identity problem in matrix semigroups in dimension four

Now we tighten the decidability gap proving undecidability for 4×4 matrices, when the generating set has eight matrices (reducing from 48), with a new technique exploiting the anti-diagonal entries.

▶ **Theorem 13.** *Given a semigroup S generated by eight 4×4 integer matrices with determinant one, determining whether the identity matrix belongs to S is undecidable.*

Proof. We prove the claim by reducing from the PCP. We shall use an encoding to embed an instance of the PCP into a set of 4×4 integer matrices. An *instance* of the PCP consists of two morphisms $g,h:\Sigma^*\to B^*$, where Σ and B are alphabets. A nonempty word $u\in\Sigma^*$ is a *solution* of an instance (g,h) if it satisfies $g(u) = h(u)$.

Let α be the mapping of Lemma 2. We also define a monomorphism $f:\mathrm{FG}(\Gamma_2)\to\mathbb{Z}^{2\times 2}$ as $f(a) = \left(\begin{smallmatrix}1&2\\0&1\end{smallmatrix}\right)$, $f(\bar{a}) = \left(\begin{smallmatrix}1&-2\\0&1\end{smallmatrix}\right)$, $f(b) = \left(\begin{smallmatrix}1&0\\2&1\end{smallmatrix}\right)$ and $f(\bar{b}) = \left(\begin{smallmatrix}1&0\\-2&1\end{smallmatrix}\right)$. Recall that the matrices $\left(\begin{smallmatrix}1&2\\0&1\end{smallmatrix}\right)$ and $\left(\begin{smallmatrix}1&0\\2&1\end{smallmatrix}\right)$ generate a free subgroup of $\mathrm{SL}(2,\mathbb{Z})$ [31]. The composition of two monomorphisms α and f gives us the embedding from an arbitrary group alphabet into $\mathrm{SL}(2,\mathbb{Z})$. We use the composition of two monomorphisms α and f to encode a set of pairs of words over an arbitrary group alphabet into a set of 4×4 integer matrices in $\mathrm{SL}(4,\mathbb{Z})$ and denote it by β.

Let (g,h) be an instance of the PCP, where $g,h:\{a_1,\ldots,a_n\}^*\to\Sigma_2^*$, where $\Sigma_2 = \{a,b\}$. Without loss of generality, we can assume that the solution starts with the letter

a_1. Moreover, we assume that this is the only occurence of a_1. We define the alphabet $\Gamma = \Sigma_2 \cup \Sigma_2^{-1} \cup \Sigma_B \cup \Sigma_B^{-1}$, where $\Sigma_B = \{q_0, q_1, p_0, p_1\}$ is the alphabet for the border letters that enforce the form of a solution.

Let us define the following sets of words $W_1 \cup W_2 \subseteq \mathrm{FG}(\Gamma) \times \mathrm{FG}(\Gamma)$, where

$$W_1 = \{(q_0 a \overline{q_0}, p_0 a \overline{p_0}),\ (q_0 b \overline{q_0}, p_0 b \overline{p_0}) \mid a, b \in \Sigma_2,\ \ q_0, p_0 \in \Sigma_B\} \text{ and}$$

$$W_2 = \left\{(q_0 \overline{g(a_1)} \overline{q_1}, p_0 \overline{h(a_1)} \overline{p_1}),\ (q_1 \overline{g(a_i)} \overline{q_1}, p_1 \overline{h(a_i)} \overline{p_1}) \mid 1 < i \leq n,\ \ q_0, q_1, p_0, p_1 \in \Sigma_B\right\}.$$

Intuitively, the words from set W_1 are used to construct words over Σ_2 and the words from set W_2 to cancel them according to the instance of the PCP.

Let us prove that $(q_0\overline{q_1}, p_0\overline{p_1}) \in \mathrm{FG}(W_1 \cup W_2)$ if and only if the PCP has a solution. It is easy to see that any pair of non-empty words in $\mathrm{FG}(W_1)$ is of the form $(q_0 w \overline{q_0}, p_0 w \overline{p_0})$ for $w \in \Sigma_2^+$. Then there exists a pair of words in $\mathrm{FG}(W_2)$ of the form $(q_0 \overline{w} \overline{q_1}, p_0 \overline{w} \overline{p_1})$ for some word $w \in \Sigma_2^+$ if and only if the PCP has a solution. Thus, $(q_0\overline{q_1}, p_0\overline{p_1})$ can be constructed by concatenating pairs of words in W_1 and W_2 if and only if the PCP has a solution.

For each pair of words $(u, v) \in \mathrm{FG}(W_1 \cup W_2)$, we define a matrix $A_{u,v}$ to be $\begin{pmatrix} \beta(u) & \mathbf{0}_2 \\ \mathbf{0}_2 & \beta(v) \end{pmatrix} \in \mathrm{SL}(4, \mathbb{Z})$, where $\mathbf{0}_2$ is the zero matrix in $\mathbb{Z}^{2\times 2}$. Moreover, we define the following matrix

$$B_{q_1\overline{q_0}, p_1\overline{p_0}} = \begin{pmatrix} \mathbf{0}_2 & \beta(q_1\overline{q_0}) \\ \beta(p_1\overline{p_0}) & \mathbf{0}_2 \end{pmatrix} \in \mathrm{SL}(4, \mathbb{Z}).$$

Let S be a matrix semigroup generated by the set $\{A_{u,v}, B_{q_1\overline{q_0}, p_1\overline{p_0}} \mid (u, v) \in W_1 \cup W_2\}$. We already know that the pair $(q_0\overline{q_1}, p_0\overline{p_1})$ of words can be generated by concatenating words in W_1 and W_2 if and only if the PCP has a solution. The matrix semigroup S has the corresponding matrix $A_{q_0\overline{q_1}, p_0\overline{p_1}}$ and thus, $\begin{pmatrix} \beta(q_0\overline{q_1}) & \mathbf{0}_2 \\ \mathbf{0}_2 & \beta(p_0\overline{p_1}) \end{pmatrix}\begin{pmatrix} \mathbf{0}_2 & \beta(q_1\overline{q_0}) \\ \beta(p_1\overline{p_0}) & \mathbf{0}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{0}_2 & \beta(\varepsilon) \\ \beta(\varepsilon) & \mathbf{0}_2 \end{pmatrix} \in S$. Now, the identity matrix $\boldsymbol{I}_4$ exists in the semigroup S by repeating this product twice.

Now we prove that the identity matrix does not exist in S if the PCP has no solution. It is easy to see that we cannot obtain the identity matrix only by multiplying 'A' matrices since there is no possibility of cancelling every border letter. We need to multiply the matrix $B_{q_1\overline{q_0}, p_1\overline{p_0}}$ with a product of 'A' matrices at some point to reach the identity matrix. Note that the matrix $B_{q_1\overline{q_0}, p_1\overline{p_0}}$ cannot be the first matrix of the product, followed by the 'A' matrices, because the upper right block of $B_{q_1\overline{q_0}, p_1\overline{p_0}}$, which corresponds to the first word of the pair, should be multiplied with the lower right block of 'A' matrix, which corresponds to the second word of the pair.

Suppose that the 'A' matrix is of form $\begin{pmatrix} \beta(q_0 u \overline{q_1}) & \mathbf{0}_2 \\ \mathbf{0}_2 & \beta(p_0 v \overline{p_1}) \end{pmatrix}$. Since the PCP instance has no solution, either u or v is not the empty word. We multiply $B_{q_1\overline{q_0}, p_1\overline{p_0}}$ to the matrix and then obtain the following matrix $\begin{pmatrix} \beta(q_0 u \overline{q_1}) & \mathbf{0}_2 \\ \mathbf{0}_2 & \beta(p_0 v \overline{p_1}) \end{pmatrix}\begin{pmatrix} \mathbf{0}_2 & \beta(q_1\overline{q_0}) \\ \beta(p_1\overline{p_0}) & \mathbf{0}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{0}_2 & \beta(q_0 u \overline{q_0}) \\ \beta(p_0 v \overline{p_0}) & \mathbf{0}_2 \end{pmatrix}$. We can see that either the upper right part or the lower left part cannot be $\beta(\varepsilon)$, which actually corresponds to the identity matrix in $\mathbb{Z}^{2\times 2}$. Now the only possibility of reaching the identity matrix is to multiply matrices which have $\mathrm{SL}(2, \mathbb{Z})$ matrices in the anti-diagonal coordinates like $B_{q_1\overline{q_0}, p_1\overline{p_0}}$. However, we cannot cancel the parts because the upper right block (the lower left block) of the left matrix is multiplied with the lower left block (the upper right block) of the right matrix as follows $\left(\begin{smallmatrix} \mathbf{0}_2 & A \\ B & \mathbf{0}_2 \end{smallmatrix}\right)\left(\begin{smallmatrix} \mathbf{0}_2 & C \\ D & \mathbf{0}_2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} AD & \mathbf{0}_2 \\ \mathbf{0}_2 & BC \end{smallmatrix}\right)$, where A, B, C and D are matrices in $\mathbb{Z}^{2\times 2}$. As the first word of the pair is encoded in the upper right block of the matrix and the second word is encoded in the lower left block, it is not difficult to see that we cannot cancel the remaining blocks.

Currently, the undecidability bound for the PCP is five [34] and thus the semigroup S is generated by eight matrices. Recall that in the beginning of the proof, we assumed that letter a_1 of the PCP is used exacly once and is the first letter of a solution. This property is in fact present in [34]. ◀

Theorem 13 implies smaller undecidability bounds for the *special diagonal membership problem* from 14 [24] to eight and for the identity problem in $\mathbb{H}(\mathbb{Q})^{2\times 2}$ from 48 to eight [6].

References

1. Andrei M. Akimenkov. Subgroups of the braid group B_4. *Mathematical notes of the Academy of Sciences of the USSR*, 50(6):1211–1218, 1991. doi:10.1007/BF01158260.
2. László Babai, Robert Beals, Jin-yi Cai, Gábor Ivanyos, and Eugene M. Luks. Multiplicative equations over commuting matrices. In *Proceedings of SODA 1996*, pages 498–507. SIAM, 1996. URL: http://dl.acm.org/citation.cfm?id=313852.314109.
3. Paul Bell, Vesa Halava, Tero Harju, Juhani Karhumäki, and Igor Potapov. Matrix equations and Hilbert's Tenth Problem. *International Journal of Algebra and Computation*, 18(08):1231–1241, dec 2008. doi:10.1142/s0218196708004925.
4. Paul C. Bell, Mika Hirvensalo, and Igor Potapov. The identity problem for matrix semigroups in SL$(2,\mathbb{Z})$ is NP-complete. In *Proceedings of SODA 2017*, pages 187–206. SIAM, 2017. doi:10.1137/1.9781611974782.13.
5. Paul C. Bell and Igor Potapov. Reachability problems in quaternion matrix and rotation semigroups. *Information and Computation*, 206(11):1353–1361, 2008. doi:10.1016/j.ic.2008.06.004.
6. Paul C. Bell and Igor Potapov. On the undecidability of the identity correspondence problem and its applications for word and matrix semigroups. *International Journal of Foundations of Computer Science*, 21(6):963–978, 2010. doi:10.1142/S0129054110007660.
7. Vladimir N. Bezverkhnii and Irina V. Dobrynina. Undecidability of the conjugacy problem for subgroups in the colored braid group R_5. *Matematicheskie Zametki*, 65(1):15–22, 1999. doi:10.1007/BF02675004.
8. Jean-Camille Birget and Stuart W. Margolis. Two-letter group codes that preserve aperiodicity of inverse finite automata. *Semigroup Forum*, 76:159–168, 2008. doi:10.1007/s00233-007-9024-6.
9. Kenneth R. Blaney and Andrey Nikolaev. A PTIME solution to the restricted conjugacy problem in generalized Heisenberg groups. *Groups Complexity Cryptology*, 8(1):69–74, 2016. doi:10.1515/gcc-2016-0003.
10. Vincent D. Blondel and Alexandre Megretski, editors. *Unsolved problems in mathematical systems and control theory*. Princeton University Press, 2004.
11. Jean-Luc Brylinski. *Loop spaces, characteristic classes, and geometric quantization*. Birkhäuser, 1993.
12. Julien Cassaigne, Tero Harju, and Juhani Karhumäki. On the undecidability of freeness of matrix semigroups. *International Journal of Algebra and Computation*, 9(03n04):295–305, 1999. doi:10.1142/S0218196799000199.
13. Émilie Charlier and Juha Honkala. The freeness problem over matrix semigroups and bounded languages. *Information and Computation*, 237:243–256, 2014. doi:10.1016/j.ic.2014.03.001.
14. Christian Choffrut. A remark on the representation of trace monoids. *Semigroup Forum*, 40(1):143–152, 1990. doi:10.1007/bf02573262.
15. Christian Choffrut and Juhani Karhumäki. Some decision problems on integer matrices. *RAIRO - Theoretical Informatics and Applications*, 39(1):125–131, 2005. doi:10.1051/ita:2005007.
16. Ventsislav Chonev, Joël Ouaknine, and James Worrell. The orbit problem in higher dimensions. In *Proceedings of STOC 2013*, pages 941–950. ACM, 2013. doi:10.1145/2488608.2488728.
17. Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the complexity of the orbit problem. *Journal of the ACM*, 63(3):23:1–23:18, 2016. doi:10.1145/2857050.

18 Marston Conder, Edmund Robertson, and Peter Williams. Presentations for 3-dimensional special linear groups over integer rings. *Proceedings of the American Mathematical Society*, 115(1):19–26, 1992. doi:10.2307/2159559.

19 Marston D. E. Conder. Some unexpected consequences of symmetry computations. In *SIGMAP 2014*, volume 159 of *PROMS*, pages 71–79. Springer, 2016. doi:10.1007/978-3-319-30451-9_3.

20 Jintai Ding, Alexei Miasnikov, and Alexander Ushakov. A linear attack on a key exchange protocol using extensions of matrix semigroups. *IACR Cryptology ePrint Archive*, 2015:18, 2015.

21 Esther Galby, Joël Ouaknine, and James Worrell. On matrix powering in low dimensions. In *Proceedings of STACS 2015*, volume 30 of *LIPIcs*, pages 329–340, 2015. doi:10.4230/LIPIcs.STACS.2015.329.

22 Razvan Gelca and Alejandro Uribe. From classical theta functions to topological quantum field theory. In *The influence of Solomon Lefschetz in geometry and topology*, volume 621 of *Contemprorary Mathematics*, pages 35–68. American Mathematical Society, 2014. doi:10.1090/conm/621.

23 Yuri Gurevich and Paul Schupp. Membership problem for the modular group. *SIAM Journal of Computing*, 37(2):425–459, 2007. doi:10.1137/050643295.

24 Vesa Halava, Tero Harju, and Mika Hirvensalo. Undecidability bounds for integer matrices using Claus instances. *International Journal of Foundations of Computer Science*, 18(5):931–948, 2007. doi:10.1142/S0129054107005066.

25 Juha Honkala. A Kraft-McMillan inequality for free semigroups of upper-triangular matrices. *Information and Computation*, 239:216–221, 2014. doi:10.1016/j.ic.2014.09.002.

26 Juha Honkala. Products of matrices and recursively enumerable sets. *Journal of Computer and System Sciences*, 81(2):468–472, 2015. doi:10.1016/j.jcss.2014.10.004.

27 Sang-Ki Ko, Reino Niskanen, and Igor Potapov. On the identity problem for the special linear group and the Heisenberg group. *CoRR*, abs/1706.04166, 2017. URL: https://arxiv.org/abs/1706.04166, arXiv:1706.04166.

28 Daniel König, Markus Lohrey, and Georg Zetzsche. Knapsack and subset sum problems in nilpotent, polycyclic, and co-context-free groups. *Algebra and Computer Science*, 677:138–153, 2016. doi:10.1090/conm/677/13625.

29 Bertram Kostant. Quantization and unitary representations. In *Lectures in Modern Analysis and Applications III*, pages 87–208. Springer, 1970. doi:10.1007/BFb0079068.

30 Alexei Lisitsa and Igor Potapov. Membership and reachability problems for row-monomial transformations. In *Proceedings of MFCS 2004*, volume 3153 of *LNCS*, pages 623–634. Springer, 2004. doi:10.1007/978-3-540-28629-5_48.

31 Roger C. Lyndon and Paul E. Schupp. *Combinatorial group theory*. Springer, 1977. doi:10.1007/978-3-642-61896-3.

32 Andrei A. Markov. On certain insoluble problems concerning matrices. *Doklady Akademii Nauk SSSR*, 57(6):539–542, 1947.

33 Alexei Mishchenko and Alexander Treier. Knapsack problem for nilpotent groups. *Groups Complexity Cryptology*, 9(1):87–98, 2017. doi:10.1515/gcc-2017-0006.

34 Turlough Neary. Undecidability in binary tag systems and the Post correspondence problem for five pairs of words. In *Proceedings of STACS 2015*, volume 30 of *LIPIcs*, pages 649–661. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.649.

35 Joël Ouaknine, João Sousa Pinto, and James Worrell. On termination of integer linear loops. In *Proceedings of SODA 2015*, pages 957–969. SIAM, 2015. doi:10.1137/1.9781611973730.65.

36 Joël Ouaknine and James Worrell. On the positivity problem for simple linear recurrence sequences. In *Proceedings of ICALP 2014*, volume 8573 of *LNCS*, pages 318–329. Springer, 2014. doi:10.1007/978-3-662-43951-7_27.

37 Joël Ouaknine and James Worrell. Ultimate positivity is decidable for simple linear recurrence sequences. In *Proceedings of ICALP 2014*, volume 8573 of *LNCS*, pages 330–341. Springer, 2014. doi:10.1007/978-3-662-43951-7_28.

38 Christos H. Papadimitriou. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768, 1981. doi:10.1145/322276.322287.

39 Michael S. Paterson. Unsolvability in 3×3 matrices. *Studies in Applied Mathematics*, 49(1):105, 1970. doi:10.1002/sapm1970491105.

40 Igor Potapov. Composition problems for braids. In *Proceedings of FSTTCS 2013*, volume 24 of *LIPIcs*, pages 175–187. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.FSTTCS.2013.175.

41 Igor Potapov and Pavel Semukhin. Decidability of the membership problem for 2×2 integer matrices. In *Proceedings of SODA 2017*, pages 170–186. SIAM, 2017. doi:10.1137/1.9781611974782.12.

42 Igor Potapov and Pavel Semukhin. Membership problem in $\mathrm{GL}(2,\mathbb{Z})$ extended by singular matrices. In *Proceedings of MFCS 2017*, LIPIcs, pages 44:1–44:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.MFCS.2017.44.

43 Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1998.

Gaifman Normal Forms for Counting Extensions of First-Order Logic

Dietrich Kuske
Technische Universität Ilmenau, Germany
dietrich.kuske@tu-ilmenau.de

Nicole Schweikardt[1]
Humboldt-Universität zu Berlin, Germany
schweikn@informatik.hu-berlin.de

Abstract

We consider the extension of first-order logic FO by unary counting quantifiers and generalise the notion of Gaifman normal form from FO to this setting. For formulas that use only ultimately periodic counting quantifiers, we provide an algorithm that computes equivalent formulas in Gaifman normal form. We also show that this is not possible for formulas using at least one quantifier that is not ultimately periodic.

Now let d be a degree bound. We show that for any formula φ with arbitrary counting quantifiers, there is a formula γ in Gaifman normal form that is equivalent to φ on all finite structures of degree $\leqslant d$. If the quantifiers of φ are decidable (decidable in elementary time, ultimately periodic), γ can be constructed effectively (in elementary time, in worst-case optimal 3-fold exponential time).

For the setting with unrestricted degree we show that by using our Gaifman normal form for formulas with only ultimately periodic counting quantifiers, a known fixed-parameter tractability result for FO on classes of structures of bounded local tree-width can be lifted to the extension of FO with ultimately periodic counting quantifiers (a logic equally expressive as FO+MOD, i.e., first-oder logic with modulo-counting quantifiers).

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases Finite model theory, Gaifman locality, modulo-counting quantifiers, fixed parameter tractable model-checking

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.133

1 Introduction

As database specialists know very well, when evaluating a query (i.e., a formula) in a database (i.e., a relational structure), it is often advantageous to first transform the formula into an equivalent one and then evaluate this new formula in the given structure. Using this approach, one also gets algorithmic meta-theorems stating that the evaluation of formulas from a certain logic in structures from a certain class is fixed-parameter tractable. For example, this is known for formulas from monadic second-order logic MSO and its extension CMSO with modulo-counting predicates and the class of labeled trees [5, 27] or classes of bounded tree-width [3, 1]. For first-order logic FO, it is known for classes of structures of bounded degree [25], for the class of planar graphs and, more generally, for classes of bounded local tree-width [9], for classes of locally bounded expansion [6], and for classes that are effectively nowhere dense [12].

[1] Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SCHW 837/5-1.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 133; pp. 133:1–133:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Gaifman's normal form theorem [10] provides an approach to this formula transformation, and it has been applied for obtaining several of the results on FO mentioned above. It is the aim of this paper to demonstrate that this approach does not only work for first-order logic, but also for its extension FO+MOD by modulo-counting quantifiers (these quantifiers allow to make statements of the form "the number of witnesses x for a formula φ is congruent r modulo m", for fixed integers r and m). The logic FO+MOD has been well-studied, see e.g., [26, 18, 24, 23, 14, 17]. Its expressivity lies strictly between that of FO and CMSO. It is known that all FO+MOD-queries are Gaifman-local [18]. But an extension of Gaifman's normal form theorem [10] from FO to FO+MOD has not been achieved in the literature.

Recall that a first-order formula is in Gaifman normal form if it is a Boolean combination of (1) first-order properties of the neighbourhood of the free variables and (2) statements that express the existence of mutually far-apart elements whose neighbourhoods share a first-order property. Gaifman's normal form theorem states that every first-order formula is effectively equivalent to such a formula in Gaifman normal form.

We propose a notion of Gaifman normal form for FO+MOD and, more generally, for the extension of first-order logic by unary counting quantifiers FO($\mathbf{Q}$): it is a Boolean combination of (1) FO($\mathbf{Q}$)-properties of the neighbourhood of the free variables, (2) statements that express the existence of mutually far-apart elements whose neighbourhoods share an FO($\mathbf{Q}$)-property, and (3) statements that depend on the total number of elements whose neighbourhoods share an FO($\mathbf{Q}$)-property. We show that if a formula uses only ultimately periodic counting quantifiers (and therefore is equivalent to a formula of FO+MOD), then Gaifman's theorem holds *mutatis mutandis*: any such formula can be transformed effectively into an equivalent formula in Gaifman normal form using the same counting quantifiers. The proof of this result extends the original proof for first-order logic from [10]; a crucial ingredient is an effective Feferman-Vaught decomposition [8] for FO($\mathbf{Q}$) that we prove first. Adapting [4], we show that the size of the resulting formula cannot be bounded by an elementary function. Furthermore, we prove that formulas with non-ultimately periodic counting quantifiers (e.g., the set of primes) do not have equivalent formulas in Gaifman normal form.

The situation changes when we restrict attention to classes of finite structures of bounded degree. Call two formulas "finitely d-equivalent" if they are equivalent on all finite structures of degree $\leqslant d$. We show that (1) for a formula with ultimately periodic counting quantifiers, one can compute in (worst-case optimal) 3-fold exponential time a finitely d-equivalent formula in Gaifman normal form; (2) from a formula with computable counting quantifiers, we can effectively compute a finitely d-equivalent formula in Gaifman normal form; and (3) if we allow arbitary counting quantifiers, then we get at least the existence of finitely d-equivalent Gaifman normal forms. In other words, by restricting the class of structures, the complexity drops from non-elementary to 3-fold exponential (for ultimately periodic counting quantifiers), from non-existent to computable (for computable quantifiers), and from non-existent to existent (for arbitrary quantifiers). The proofs of these results do not follow Gaifman's original proof, but generalise a proof for first-order logic from [16] that, in turn, builds on [2]. In the present setting of FO($\mathbf{Q}$), we first transform the original FO($\mathbf{Q}$)-formula in elementary time into a formula in "(weak) Hanf normal form" [17, 19], and afterwards we transform this formula into Gaifman normal form by a construction similar to the one in [16].

We also provide an algorithmic application that demonstrates the usefulness of our normal form: By applying our Gaifman normal form algorithm, we lift the result of [9] from FO to FO+MOD, showing that the model-checking problem for FO+MOD-sentences on classes of finite relational structures of bounded local tree-width is fixed-parameter tractable.

The rest of the paper is structured as follows. Section 2 provides the basic definitions and introduces our notion of Gaifman normal form. Section 3 provides effective Feferman-Vaught decompositions for the extension of first-order logic by ultimately periodic counting quantifiers. The sections 4 and 5 present our results in the setting without and with a degree bound, respectively. Section 6 provides an algorithmic application.

2 Preliminaries

We write $\mathcal{P}(S)$ to denote the power set of a set S. For an n-tuple $\overline{x} = (x_1, \ldots, x_n)$ we write $|\overline{x}|$ to denote the tuple's length n. We write $\mathbb{N}$ for the set of non-negative integers, and we let $\mathbb{N}_{\geqslant 1} = \mathbb{N} \setminus \{0\}$. For $m, n \in \mathbb{N}$ with $m \leqslant n$, we write $[m, n]$ for the set $\{i \in \mathbb{N} : m \leqslant i \leqslant n\}$. For a real number $r > 0$, we write $\log(r)$ to denote the logarithm of r with respect to base 2. We use the standard $\mathcal{O}$-notation, and by $\mathrm{poly}(n)$ we mean $n^{\mathcal{O}(1)}$. We say that a function f from $\mathbb{N}$ to the set $\mathbb{R}_{\geqslant 0}$ of non-negative reals is *at most k-fold exponential*, for some $k \in \mathbb{N}$, if there exists a number $c > 0$ such that for all sufficiently large $n \in \mathbb{N}$ we have $f(n) \leqslant \exp_k(n^c)$, where $\exp_k(m)$ is a tower of 2s of height k with an m on top, i.e., $\exp_0(m) = m$ and $\exp_{k+1}(m) = 2^{\exp_k(m)}$ for all $k, m \geqslant 0$. A function f is *elementary* if it is at most k-fold exponential for some $k \geqslant 0$. The function $\mathrm{tower}\colon \mathbb{N} \to \mathbb{N}$, defined via $\mathrm{tower}(h) := \exp_h(1)$ for all $h \in \mathbb{N}$, is not elementary.

Structures and formulas. A *signature* σ is a *finite* set of relation symbols and constant symbols. Associated with every relation symbol R is a positive integer $\mathrm{ar}(R)$ called the *arity* of R. We call a signature *relational* if it only contains relation symbols. A *σ-structure* $\mathcal{A}$ consists of a non-empty set A called the *universe* of $\mathcal{A}$, a relation $R^{\mathcal{A}} \subseteq A^{\mathrm{ar}(R)}$ for each relation symbol $R \in \sigma$, and an element $c^{\mathcal{A}} \in A$ for each constant symbol $c \in \sigma$. Note that according to this definition, all signatures considered in this paper are finite while structures can be infinite. We write $\mathcal{A} \cong \mathcal{B}$ to indicate that two σ-structures $\mathcal{A}$ and $\mathcal{B}$ are isomorphic.

We use the standard notation concerning first-order logic and extensions thereof, cf. [7, 20]. By $\mathrm{FO}[\sigma]$ we denote the class of all first-order formulas of signature σ, and by FO we denote the union of all $\mathrm{FO}[\sigma]$ for arbitrary signatures σ. By $\mathrm{free}(\varphi)$ we denote the set of all *free variables* of the formula φ. A *sentence* is a formula φ with $\mathrm{free}(\varphi) = \emptyset$. We write $\varphi(\overline{x})$, for $\overline{x} = (x_1, \ldots, x_n)$ with $n \geqslant 0$, to indicate that $\mathrm{free}(\varphi) \subseteq \{x_1, \ldots, x_n\}$. If $\mathcal{A}$ is a σ-structure and $\overline{a} = (a_1, \ldots, a_n) \in A^n$, by $\mathcal{A} \models \varphi(\overline{a})$ or $(\mathcal{A}, \overline{a}) \models \varphi$ we indicate that the formula $\varphi(\overline{x})$ is satisfied in $\mathcal{A}$ when interpreting the free occurrences of the variables $x_1, \ldots, x_n$ with the elements $a_1, \ldots, a_n$.

Unary counting quantifiers. In addition to the existential quantifier $\exists$ we consider *unary counting quantifiers* (for short: counting quantifiers), which are defined as subsets of $\mathbb{N}$. We will use the terms "set (of natural numbers)" and "counting quantifier" interchangeably. For a set $\mathbf{Q} \subseteq \mathcal{P}(\mathbb{N})$ of counting quantifiers we write $\mathrm{FO}(\mathbf{Q})[\sigma]$ to denote the *extension of* $\mathrm{FO}[\sigma]$ *with the quantifiers from* $\mathbf{Q}$. Precisely, we add the following formation rule for formulas:

If $\varphi(\overline{x}, y) \in \mathrm{FO}(\mathbf{Q})[\sigma]$, $\mathsf{Q} \in \mathbf{Q}$, and $k \in \mathbb{N}$, then also $(\mathsf{Q}{+}k)y\,\varphi$ belongs to $\mathrm{FO}(\mathbf{Q})[\sigma]$.

For $(\mathsf{Q}{+}0)y\,\varphi$ we write the more succinct $\mathsf{Q}y\,\varphi$. The formula $(\mathsf{Q}{+}k)y\,\varphi(\overline{x}, y)$ expresses that the number of witnesses y for $\varphi(\overline{x}, y)$ belongs to the set $(\mathsf{Q}{+}k) := \{q + k : q \in \mathsf{Q}\}$. Equivalently, this means that the formula $(\mathsf{Q}{+}k)y\,\varphi(\overline{x}, y)$ is satisfied by a σ-structure $\mathcal{A}$ and an interpretation $\overline{a}$ of the variables $\overline{x}$ iff $|\{b \in A : \mathcal{A} \models \varphi(\overline{a}, b)\}| - k \in \mathsf{Q}$. Here, for an *infinite* set B we use the convention that $|B| = \infty \notin \mathbb{N}$, where ∞ is larger than any integer,

and $\infty - k = \infty$ for all integers k. Every formula can be transformed into an equivalent exponentially larger *displacement-free* formula, meaning that counting quantifiers appear only in the form $(\mathsf{Q} + 0)$.

▶ **Example 2.1.** For $m \geqslant 2$, the quantifier $\mathsf{D}_m = m \cdot \mathbb{N}$ contains the multiples of m. Let $\mathbf{D} = \{\mathsf{D}_m : m \in \mathbb{N}, m \geqslant 2\}$ denote the collection of all these *divisibility quantifiers*. Then the logic FO($\mathbf{D}$) is equally expressive as the logic FO+MOD (cf. [26, 24]).

The quantifier rank $\mathrm{qr}(\varphi)$ of an FO($\mathbf{Q}$)-formula φ is defined as the maximal nesting depth of *all* quantifiers. For a number $\ell \in \mathbb{N}_{\geqslant 1}$ and a formula $\varphi(\overline{x}, y)$, we write $\exists^{\geqslant \ell} y\, \varphi$ to denote a formula expressing that there are at least ℓ witnesses y which satisfy φ.

Gaifman graph. The *Gaifman graph* $G_{\mathcal{A}}$ of a σ-structure $\mathcal{A}$ is the undirected, loop-free graph with vertex set A and an edge between two distinct vertices $a, b \in A$ iff there exist a relation symbol $R \in \sigma$ and a tuple $(a_1, \ldots, a_{\mathrm{ar}(R)}) \in R^{\mathcal{A}}$ such that $a, b \in \{a_1, \ldots, a_{\mathrm{ar}(R)}\}$. The *degree* of a σ-structure $\mathcal{A}$ is the degree of its Gaifman graph $G_{\mathcal{A}}$. If this degree is at most d, then we call $\mathcal{A}$ *d-bounded*. Two formulas $\varphi(\overline{x})$ and $\psi(\overline{x})$ of signature σ are called *finitely d-equivalent* if $\mathcal{A} \models \forall \overline{x}\, (\varphi \leftrightarrow \psi)$ holds for every finite d-bounded σ-structure $\mathcal{A}$.

The *distance* $\mathrm{dist}^{\mathcal{A}}(a, b)$ between two elements $a, b \in A$ is the minimal length (i.e., the number of edges) of a path from a to b in $G_{\mathcal{A}}$, and if no such path exists, we set $\mathrm{dist}^{\mathcal{A}}(a, b) = \infty$. For a tuple $\overline{a} \in A^m$ and an element $b \in A$, we let $\mathrm{dist}^{\mathcal{A}}(\overline{a}, b) = \min\{\mathrm{dist}^{\mathcal{A}}(a_i, b) : 1 \leqslant i \leqslant m\}$.

For any signature σ and any $k, r \in \mathbb{N}$, there exists a formula $\mathrm{dist}_{<r}(\overline{x}, y) \in \mathrm{FO}[\sigma]$ with $\overline{x} = (x_1, \ldots, x_k)$ such that for any σ-structure $\mathcal{A}$, any $\overline{a} \in A^k$, and any $b \in A$ we have $(\mathcal{A}, \overline{a}, b) \models \mathrm{dist}_{<r}(\overline{x}, y)$ iff $\mathrm{dist}^{\mathcal{A}}(\overline{a}, b) < r$. We write $\mathrm{dist}(\overline{x}, y) < r$ for the formula $\mathrm{dist}_{<r}(\overline{x}, y)$, and $\mathrm{dist}(\overline{x}, y) > r$ for the formula $\neg\, \mathrm{dist}_{<r+1}(\overline{x}, y)$.

Gaifman normal forms. Let $\mathbf{Q}$ be a set of counting quantifiers and σ a signature. A formula $\lambda(\overline{x}) \in \mathrm{FO}(\mathbf{Q})[\sigma]$ is *local* if all quantifications in λ are restricted to the neighborhood of the variables $\overline{x}$. The precise inductive definition proceeds as follows:

- Atomic formulas are *r-local* for any $r \in \mathbb{N}$.
- If $\lambda(\overline{x}, y)$ is r-local, $\overline{z}$ is a non-empty sub-tuple of $\overline{x}$, and $r' \in \mathbb{N}$, then the formula $\exists y \big(\mathrm{dist}(\overline{z}, y) \leqslant r' \wedge \lambda(\overline{x}, y)\big)$ is $(r' + r)$-*local*, and for all $\mathsf{Q} \in \mathbf{Q}$ and $k \in \mathbb{N}$, the formula $(\mathsf{Q}{+}k) y \big(\mathrm{dist}(\overline{z}, y) \leqslant r' \wedge \lambda(\overline{x}, y)\big)$ is $(r' + r)$-*local* as well.
- If $\lambda_1(\overline{x})$ and $\lambda_2(\overline{x})$ are r-local, then $(\lambda_1 \wedge \lambda_2)$ and $\neg\lambda_1$ are *r-local* as well.

An r-local formula is also $(r + 1)$-local. A formula is *local* if it is r-local for some $r \in \mathbb{N}$.

For a σ-structure $\mathcal{A}$, a tuple $\overline{a} = (a_1, \ldots, a_n) \in A^n$ and a number $r \in \mathbb{N}$, the *r-ball* $N_r^{\mathcal{A}}(\overline{a})$ of $\overline{a}$ is the set of all $b \in A$ with $\mathrm{dist}^{\mathcal{A}}(\overline{a}, b) \leqslant r$. If σ is a *relational* signature, then the *r-neighbourhood* $\mathcal{N}_r^{\mathcal{A}}(\overline{a})$ of $\overline{a}$ is the substructure of $\mathcal{A}$ induced on the set $N_r^{\mathcal{A}}(\overline{a})$.

Let $\lambda(\overline{x})$ be an r-local formula and let $\overline{a} \in A^{|\overline{x}|}$. When determining whether $\mathcal{A} \models \lambda(\overline{a})$, quantification is restricted to elements of distance at most r from $\overline{a}$, i.e., to elements in $N_r^{\mathcal{A}}(\overline{a})$. Hence we get $\mathcal{A} \models \lambda(\overline{a}) \iff \mathcal{N}_r^{\mathcal{A}}(\overline{a}) \models \lambda(\overline{a})$.

Let $\mathrm{L} \subseteq \mathrm{FO}(\mathbf{Q})$. A *counting sentence* over L is a sentence of the form $(\mathsf{Q}{+}k) x\, \lambda(x)$, where $\lambda \in \mathrm{L}$ is local, $\mathsf{Q} \in \mathbf{Q}$, and $k \in \mathbb{N}$. A *basic local sentence* over L is a sentence of the form $\exists x_1 \cdots \exists x_m \Big(\bigwedge_{1 \leqslant i < j \leqslant m} \mathrm{dist}(x_i, x_j) > 2r \ \wedge \bigwedge_{1 \leqslant i \leqslant m} \lambda(x_i) \Big)$, where $m \in \mathbb{N}_{\geqslant 1}$, $r \in \mathbb{N}$, and $\lambda \in \mathrm{L}$ is r-local. Such a basic local sentence expresses that there are m witnesses for λ of mutual distance at least $2r + 1$.

▶ **Definition 2.2.** A formula $\varphi(\overline{x}) \in \mathrm{FO}(\mathbf{Q})[\sigma]$ is *in Gaifman normal form* if it is a Boolean combination of local formulas $\lambda(\overline{x})$, of counting sentences $(\mathsf{Q}{+}k) y\, \lambda(y)$, and of basic local sentences over $\mathrm{FO}(\mathbf{Q})[\sigma]$.

If $\mathbf{Q} = \emptyset$, then a formula in Gaifman normal form consists of local formulas and basic local sentences over FO. In other words, our definition of Gaifman normal form for formulas from FO($\emptyset$) coincides with the traditional one for formulas from FO [10, 11]. The following is known about the existence and computability of formulas in Gaifman normal form:

▶ **Theorem 2.3.** *Let* $\mathbf{Q} = \emptyset$*, i.e.,* $\mathrm{FO}(\mathbf{Q}) = \mathrm{FO}$.

(1) *From a formula* $\varphi(\overline{x}) \in \mathrm{FO}$*, one can compute an equivalent formula* $\gamma(\overline{x}) \in \mathrm{FO}$ *in Gaifman normal form [10].*

(2) *The size of the equivalent formula in Gaifman normal form cannot be bounded by an elementary function in the size of the formula* $\varphi(\overline{x})$ *[4].*

(3) *From a formula* $\varphi(\overline{x}) \in \mathrm{FO}$ *and a degree bound* d*, one can compute in 3-fold exponential time a finitely* d*-equivalent formula* $\gamma(\overline{x})$ *in Gaifman normal form [16].*

It is the aim of this paper to study to what extent this theorem holds for the extension of first-order logic with unary counting quantifiers. A special role is played by ultimately periodic quantifiers that we introduce now.

Ultimately periodic sets. Let $p \in \mathbb{N}_{\geqslant 1}$ and $n_0 \in \mathbb{N}$. A set $\mathsf{Q} \subseteq \mathbb{N}$ is *ultimately* p*-periodic with offset* n_0 if $n \in \mathsf{Q} \iff n+p \in \mathsf{Q}$ holds for all $n \geqslant n_0$. A set Q is *ultimately* p*-periodic* if there exists an n_0 such that Q is ultimately p-periodic with offset n_0, and Q is *ultimately periodic* (cf. e.g. [22]) if it is ultimately p-periodic for some $p \in \mathbb{N}_{\geqslant 1}$. The *period* of Q is the *minimal* p for which Q is ultimately p-periodic.

We write $\mathbf{U}$ to denote the set of all ultimately periodic sets $\mathsf{Q} \subseteq \mathbb{N}$.

The *characteristic sequence* χ_{Q} of a set $\mathsf{Q} \subseteq \mathbb{N}$ is the ω-word $w = w_0 w_1 w_2 \cdots \in \{0,1\}^\omega$ with $\mathsf{Q} = \{n \in \mathbb{N} : w_n = 1\}$. Note that Q is ultimately periodic iff there are finite words $\alpha, \pi \in \{0,1\}^*$ with $\chi_{\mathsf{Q}} = \alpha\pi^\omega$. We represent an ultimately periodic set Q by the shortest word $\mathrm{rep}(\mathsf{Q}) := \alpha\#\pi$ satisfying $\chi_{\mathsf{Q}} = \alpha\,\pi^\omega$. The *size* $\|\mathsf{Q}\|$ of Q is defined as the length of the word $\mathrm{rep}(\mathsf{Q})$. The size of an FO($\mathbf{U}$)-formula φ of signature σ is its length when viewed as a word over the alphabet $\sigma \cup \mathrm{Var} \cup \{,\} \cup \{=, \exists, \neg, \vee, (,), 0, 1, +, \#\}$, where Var is a countable set of variables, each quantifier $\mathsf{Q} \in \mathbf{U}$ is represented by the word $\mathrm{rep}(\mathsf{Q})$, and each number k is given in binary (in subformulas of the form $(\mathsf{Q}{+}k)y\,\varphi$). The set $\mathbf{D}$ of all divisibility quantifiers (see Example 2.1) is a subset of $\mathbf{U}$. With every set of ultimately periodic quantifiers $\mathbf{Q} \subseteq \mathbf{U}$, we associate the set $\mathbf{D}_{\mathbf{Q}} \subseteq \mathbf{D}$ of divisibility quantifiers which consists of precisely those $\mathsf{D}_p = p \cdot \mathbb{N}$ for which $p \geqslant 2$ is the period of some $\mathsf{Q} \in \mathbf{Q}$.

▶ **Lemma 2.4** ([16, 17])**.** *Let* $\mathbf{Q} \subseteq \mathcal{P}(\mathbb{N})$ *be a set of counting quantifiers.*

(a) *Let* $\mathsf{Q} \in \mathbf{Q}$ *be ultimately* p*-periodic with offset* n_0*, and let* $k \in \mathbb{N}$*. Every formula from* $\mathrm{FO}(\mathbf{Q} \cup \mathbf{D}_{\mathbf{Q}})$ *of the shape* $(\mathsf{Q}{+}k)y\,\varphi$ *is equivalent to a Boolean combination of formulas of the form* $(\mathsf{D}_p{+}\ell)y\,\varphi$ *and* $\exists^{\geqslant m} y\,\varphi$ *with* $\ell < p$ *and* $m < n_0{+}k{+}p$*. This Boolean combination can be computed from* $\mathrm{rep}(\mathsf{Q}), k, \varphi$ *in polynomial time.*

(b) *Let* $\mathsf{Q} \in \mathbf{Q}$ *be ultimately periodic with period* $p \geqslant 2$*.*
Every formula from $\mathrm{FO}(\mathbf{Q} \cup \mathbf{D}_{\mathbf{Q}})$ *of the shape* $(\mathsf{D}_p{+}k)y\,\varphi$*, for a* $k \geqslant 0$*, is equivalent to a Boolean combination of formulas of the shape* $(\mathsf{Q}{+}\ell)y\,\varphi$ *and* $\exists^{\geqslant \ell} y\,\varphi$ *with* $\ell < \|\mathsf{Q}\| + p + k$*. This Boolean combination can be computed from* $\mathrm{rep}(\mathsf{Q}), k, \varphi$ *in polynomial time.*

(c) *Let* $\mathrm{L} \subseteq \mathrm{FO}(\mathbf{Q})$ *be a set of formulas that contains all atomic formulas and is closed under Boolean combinations and existential quantification.*
Every sentence $\exists^{\geqslant \ell} y\,\lambda(y)$*, where* $\lambda \in \mathrm{L}$ *is a local formula, is equivalent to a Boolean combination of basic local sentences over* L*.*
This Boolean combination can be computed from λ *and* ℓ *in time* $\mathcal{O}(\|\lambda\|) \cdot 2^{\mathcal{O}(\ell \log \ell)}$*.*

As an immediate consequence, we obtain:

▶ **Corollary 2.5.** *Let* $\mathbf{Q} \subseteq \mathbf{U}$ *and let* $\mathbf{D_Q}$ *be the associated set of divisibility quantifiers.*

(1) *For every* FO($\mathbf{Q}$)*-formula* φ*, we can compute an equivalent* FO($\mathbf{D_Q}$)*-formula and vice versa.*

(2) *For every* FO($\mathbf{D_Q}$)*-formula* ψ *in Gaifman normal form, we can compute an equivalent* FO($\mathbf{Q}$)*-formula in Gaifman normal form.*

Feferman-Vaught decompositions. A crucial tool in the construction of Gaifman normal forms for first-order logic (i.e., in the proof of Theorem 2.3(1)) is a result by Feferman and Vaught [8]. In its simplest form (which is all that is needed in this context) it expresses that the first-order theory of the disjoint sum of two structures is determined by the first-order theories of the two structures. Our proof of the generalisation of Theorem 2.3(1) to logics of the form FO($\mathbf{Q}$) will proceed similarly to Gaifman's proof for FO, and this requires us to first provide a generalisation of the result by Feferman and Vaught.

Let σ be a relational signature and let $\mathcal{A}$ and $\mathcal{B}$ be disjoint σ-structures (i.e., their universes A and B are disjoint). The *disjoint sum* $\mathcal{A} \oplus \mathcal{B}$ of $\mathcal{A}$ and $\mathcal{B}$ is the structure $\big(A \cup B,\, A,\, B,\, (R^{\mathcal{A}} \cup R^{\mathcal{B}})_{R\in\sigma}\big)$ over the signature σ_2 with two additional unary relation symbols (that we denote A and B). Since this is only defined for disjoint structures, the relations A and B form a partition of the universe of the σ_2-structure $\mathcal{A} \oplus \mathcal{B}$. Furthermore, no edge of the Gaifman graph of $\mathcal{A} \oplus \mathcal{B}$ connects nodes from A with nodes from B.

▶ **Definition 2.6.** Let $\mathbf{Q}$ be a set of counting quantifiers and $\varphi(\overline{x}, \overline{y}) \in \mathrm{FO}(\mathbf{Q})[\sigma_2]$ a formula with $\overline{x} = (x_1, \ldots, x_k)$ and $\overline{y} = (y_1, \ldots, y_\ell)$. Furthermore, let Δ be a finite set of pairs of formulas $(\alpha(\overline{x}), \beta(\overline{y}))$ from FO($\mathbf{Q}$)[σ]. The set Δ is a *decomposition of* φ *w.r.t.* $(\overline{x}; \overline{y})$ if

$$\mathcal{A} \oplus \mathcal{B} \;\models\; \varphi(\overline{a}, \overline{b}) \quad\Longleftrightarrow\quad \text{there exists } (\alpha, \beta) \in \Delta \text{ with } \mathcal{A} \models \alpha(\overline{a}) \text{ and } \mathcal{B} \models \beta(\overline{b})$$

holds for all disjoint σ-structures $\mathcal{A}$ and $\mathcal{B}$ and all tuples $\overline{a} \in A^k$ and $\overline{b} \in B^\ell$.

The following is known about the existence and computability of decompositions:

▶ **Theorem 2.7.** *Let* $\mathbf{Q} = \emptyset$*, i.e.,* FO($\mathbf{Q}$) = FO*.*

(1) *From a formula* $\varphi(\overline{x}, \overline{y}) \in \mathrm{FO}[\sigma_2]$*, one can compute a decomposition w.r.t.* $(\overline{x}; \overline{y})$ *[8].*

(2) *The size of the decomposition cannot be bounded by an elementary function in the size of the formula* $\varphi(\overline{x}, \overline{y})$ *[4].*

A more general definition of the notion "decomposition" replaces the condition "there exists $(\alpha, \beta) \in \Delta$ with ..." by a Boolean combination of statements of the form "$\mathcal{A} \models \alpha(\overline{a})$" and "$\mathcal{B} \models \beta(\overline{b})$"; and with this definition, a "decomposition" can be computed in 3-fold exponential time from any $\varphi \in \mathrm{FO}(\mathbf{D})[\sigma_2]$ and for any fixed $d \geqslant 0$; but this decomposition is only equivalent to φ provided $\mathcal{A}$ and $\mathcal{B}$ are finite and of degree at most d [15, Theorem 5.2.1] (see [13] for the first-order case). Another result in this direction is due to Courcelle who considers the extension CMSO of monadic second-order logic by predicates expressing the size of a set modulo some fixed number. In this context, Courcelle also proves a result analogous to Theorem 2.7(1) [3, Lemma 4.5]. More results in this vein can be found in [21].

3 Feferman-Vaught decompositions for FO(Q)

If $\mathbf{Q}$ is a set of counting quantifiers and $\mathsf{S} \in \mathbf{Q}$ is not ultimately periodic, then there is no decomposition for the sentence $\mathsf{S}x\, x{=}x$ [15, Theorem 8.5.2]. Here, we prove that if Q contains only ultimately periodic counting quantifiers, decompositions exist and can be computed:

▶ **Theorem 3.1.** *Let* $\mathbf{Q} \subseteq \mathbf{U}$ *and let* σ *be a relational signature. From a formula* $\varphi(\overline{x},\overline{y}) \in \mathrm{FO}(\mathbf{Q})[\sigma_2]$*, one can construct a decomposition for* φ *w.r.t.* $(\overline{x};\overline{y})$*.*

Proof. By Corollary 2.5(1), the logics $\mathrm{FO}(\mathbf{Q})$ and $\mathrm{FO}(\mathbf{D}_\mathbf{Q})$ for $\mathbf{Q} \subseteq \mathbf{U}$ are effectively equally expressive. Therefore, it suffices to prove the theorem for sets of divisibility quantifiers, i.e., for the case where $\mathbf{Q} \subseteq \mathbf{D}$. The proof proceeds by induction on the construction of the formula $\varphi(\overline{x},\overline{y})$. The cases of atomic formulas, Boolean combinations, and existential quantification are as in the first-order case, see e.g. [11, Lemma 2.3]. Here, we sketch the remaining case where $\varphi(\overline{x},\overline{y})$ is of the form $\mathsf{D}_m z\, \psi(\overline{x},\overline{y},z)$ for some $m \geqslant 2$. Let $\overline{x} = (x_1,\ldots,x_k)$ and $\overline{y} = (y_1,\ldots,y_\ell)$. For $n \in \{0,1,\ldots,m{-}1\}$, consider the formulas

$$\chi_n(\overline{x},\overline{y}) := (\mathsf{D}_m{+}n)z\,\big(A(z) \wedge \psi(\overline{x},\overline{y},z)\big) \quad \text{and} \quad \xi_n(\overline{x},\overline{y}) := (\mathsf{D}_m{+}n)z\,\big(B(z) \wedge \psi(\overline{x},\overline{y},z)\big).$$

Let $\varphi'(\overline{x},\overline{y})$ be the disjunction of all formulas $\chi_{n_1}(\overline{x},\overline{y}) \wedge \xi_{n_2}(\overline{x},\overline{y})$ where $n_1, n_2 \in \{0,\ldots,m{-}1\}$ and $n_1 + n_2 \equiv 0 \pmod m$. Clearly, $\mathcal{A} \oplus \mathcal{B} \models (\varphi \leftrightarrow \varphi')(\overline{a},\overline{b})$ holds for all disjoint structures $\mathcal{A}$ and $\mathcal{B}$ and all $\overline{a} \in A^k$ and $\overline{b} \in B^\ell$. Therefore, every decomposition of φ' is also a decomposition of φ. Furthermore, note that a decomposition for φ' can be computed from decompositions for the formulas $\chi_n(\overline{x},\overline{y})$ and $\xi_n(\overline{x},\overline{y})$ for $n \in \{0,\ldots,m{-}1\}$. All that remains to be done is to construct decompositions w.r.t. $(\overline{x};\overline{y})$ for each of the formulas $\chi_n(\overline{x},\overline{y})$ and $\xi_n(\overline{x},\overline{y})$. By symmetry, we only consider the formula ξ_n.

By the induction hypothesis, there is a decomposition $\{(\alpha_i(\overline{x}),\beta_i(\overline{y},z)) : i \in I\}$ of $\psi(\overline{x},\overline{y},z)$ w.r.t. $(\overline{x};\overline{y}z)$. We can, w.l.o.g., assume that the formulas $\alpha_i(\overline{x})$ are mutually exclusive, i.e., $\alpha_i(\overline{x}) \wedge \alpha_j(\overline{x})$ is unsatisfiable for $i \neq j$. Then, the set $\big\{\,\big(\,\alpha_i(\overline{x})\,,\,(\mathsf{D}_m{+}n)z\,\beta_i(\overline{y},z)\,\big) \;:\; i \in I\,\big\}$ is a decomposition of $\xi_n(\overline{x},\overline{y})$ w.r.t. $(\overline{x};\overline{y})$. ◀

In the inductive proof of Theorem 3.1, the size of the decomposition (i.e., the number of pairs) increases exponentially with every negation and every quantification. It follows that the size of the formulas in the resulting decomposition can be bounded by a tower of 2s whose height is proportional to the size of the formula. We can adopt and simplify the proof of [4, Theorem 3] to also obtain a non-elementary lower bound:

▶ **Proposition 3.2.** *Let* $\sigma = \{E\}$ *with* $\mathrm{ar}(E) = 2$*. There is a sequence* $(\varphi_h)_{h\geqslant 0}$ *of* $\mathrm{FO}[\sigma]$*-sentences of size* $\mathcal{O}(h)$ *such that for every elementary function* $f\colon \mathbb{N} \to \mathbb{N}$*, there is* $h \in \mathbb{N}$ *such that every decomposition* Δ_h *in* $\mathrm{FO}(\mathbf{U})$ *of* φ_h *contains some sentence of length* $> f(h)$*.*

We finish this section with a corollary to Theorem 3.1 that will be used in the construction of Gaifman normal forms in the next section.

▶ **Corollary 3.3.** *Let* $\mathbf{Q} \subseteq \mathbf{U}$ *be a set of ultimately periodic quantifiers and let* $r \in \mathbb{N}$*. From an* r*-local formula* $\lambda(\overline{x},y) \in \mathrm{FO}(\mathbf{Q})$*, one can compute a finite set* Δ' *of pairs of* r*-local* $\mathrm{FO}(\mathbf{Q})$*-formulas* $(\alpha'(\overline{x}),\beta'(y))$ *such that the following two formulas are equivalent:*

$$\mathrm{dist}(\overline{x},y) > 2r{+}1 \;\wedge\; \lambda(\overline{x},y) \qquad \textit{and} \qquad \mathrm{dist}(\overline{x},y) > 2r{+}1 \;\wedge \bigvee_{(\alpha',\beta')\in\Delta'} \big(\alpha'(\overline{x}) \wedge \beta'(y)\big)\,.$$

4 Equivalent Gaifman normal forms

The main result of this section is:

▶ **Theorem 4.1.** *Let* $\mathbf{Q}$ *be a set of unary counting quantifiers.*

(1) *If* $\mathbf{Q} \subseteq \mathbf{U}$ *contains only ultimately periodic quantifiers then, from a formula* $\varphi \in \mathrm{FO}(\mathbf{Q})$*, one can compute an equivalent formula* $\gamma \in \mathrm{FO}(\mathbf{Q})$ *in Gaifman normal form.*

(2) *If* $\mathbf{Q}$ *contains a quantifier* S *that is not ultimately periodic, then there is a sentence* $\eta \in \mathrm{FO}(\mathbf{Q})$ *such that no* $\mathrm{FO}(\mathbf{Q})$*-sentence in Gaifman normal form is equivalent to* η.

The two statements of the theorem are proved in the next two subsections.

4.1 Ultimately periodic quantifiers

Proof of Theorem 4.1(1)**.** In the light of Corollary 2.5, it suffices to consider sets $\mathbf{Q} \subseteq \mathbf{D}$ of divisibility quantifiers.

The construction of γ proceeds by structural induction following the construction of φ. The cases of atomic formulas φ as well as of Boolean combinations are trivial. If φ is of the form $\exists y\, \psi$, we can argue as in the first-order case ([10], see also [11, Sect. 4.1]), but since ψ is from $\mathrm{FO}(\mathbf{Q})$, we use Corollary 3.3 instead of Feferman-Vaught decompositions for FO (cf. [11, Lemma 2.3]). So it remains to consider the case where $\varphi(\overline{x})$ is of the form $\mathsf{D}_m y\, \psi(\overline{x}, y)$ for some $\mathsf{D_m} \in \mathbf{Q}$. By the induction hypothesis we can assume that ψ is in Gaifman normal form. Hence there are a finite set I, sentences χ_i in Gaifman normal form, and local formals $\lambda_i(\overline{x}, y)$ for all $i \in I$, such that φ is equivalent to the formula $\varphi' := \mathsf{D}_m\, y \Big(\bigvee_{i\in I} \big(\chi_i \wedge \lambda_i(\overline{x}, y) \big) \Big)$.

W.l.o.g. we can assume that the sentences χ_i are mutually exclusive, i.e., $\chi_i \wedge \chi_j$ is unsatisfiable for $i \neq j$.

Set $r' := 2r + 1$. Then, $\varphi(\overline{x})$ is equivalent to a Boolean combination of the formulas

$$\gamma_n(\overline{x}) \;:=\; (\mathsf{D}_m{+}n)\, y \,\Big(\operatorname{dist}(\overline{x}, y) \leqslant r' \;\wedge \bigvee_{i\in I} \big(\chi_i \wedge \lambda_i(\overline{x}, y) \big) \Big) \qquad \text{and}$$

$$\delta_n(\overline{x}) \;:=\; (\mathsf{D}_m{+}n)\, y \,\Big(\operatorname{dist}(\overline{x}, y) > r' \;\wedge \bigvee_{i\in I} \big(\chi_i \wedge \lambda_i(\overline{x}, y) \big) \Big)$$

with $n \in \{0, \ldots, m{-}1\}$, and it suffices to transform each of these formulas into Gaifman normal form. Since the sentences χ_i are mutually exclusive, $\gamma_n(\overline{x})$ is equivalent to

$$\bigvee_{i\in I} \big(\chi_i \;\wedge\; (\mathsf{D}_m{+}n)\, y \,\big(\operatorname{dist}(\overline{x}, y) \leqslant r' \;\wedge\; \lambda_i(\overline{x}, y) \big) \big)\,,$$

which is in Gaifman normal form. Similarly, $\delta_n(\overline{x})$ is equivalent to

$$\bigvee_{i\in I} \big(\chi_i \;\wedge\; (\mathsf{D}_m{+}n)\, y \,\big(\operatorname{dist}(\overline{x}, y) > r' \;\wedge\; \lambda_i(\overline{x}, y) \big) \big)\,.$$

Let $i \in I$. By Corollary 3.3, we can construct a finite set J and r-local formulas $\alpha_j(\overline{x})$ and $\beta_j(y)$ for $j \in J$, such that the formulas

$$\operatorname{dist}(\overline{x}, y) > r' \;\wedge\; \lambda_i(\overline{x}, y) \qquad \text{and} \qquad \operatorname{dist}(\overline{x}, y) > r' \;\wedge \bigvee_{j\in J} \big(\alpha_j(\overline{x}) \;\wedge\; \beta_j(y) \big)$$

are equivalent. Again, we can assume that the formulas $\alpha_j(\overline{x})$ are mutually exclusive. Then, $\delta_n(\overline{x})$ is equivalent to the formula

$$\bigvee_{(i,j)\in I\times J} \big(\chi_i \;\wedge\; \alpha_j(\overline{x}) \;\wedge\; (\mathsf{D}_m{+}n)\, y \,\big(\operatorname{dist}(\overline{x}, y) > r' \;\wedge\; \beta_j(y) \big) \big)\,.$$

Finally, the formula $(\mathsf{D}_m{+}n)\, y \big(\operatorname{dist}(\overline{x}, y) > r' \wedge \beta_j(y) \big)$ is equivalent to a Boolean combination of formulas of the form $(\mathsf{D}_m{+}n_1)\, y \big(\operatorname{dist}(\overline{x}, y) \leqslant r' \wedge \beta_j(y) \big)$ and $(\mathsf{D}_m{+}n_2)\, y\, \beta_j(y)$ with $n_1, n_2 \in \{0, \ldots, m{-}1\}$. Note that the first formula is $(r' + r)$-local, and the second formula is a counting sentence, i.e., both these formulas are in Gaifman normal form. ◀

In the inductive proof of Theorem 4.1(1), the size of the Gaifman normal form increases at least exponentially with every (existential or counting) quantifier, since we build the disjunctive normal form to get the formula φ' defined at the beginning of the proof. Consequently, we only obtain a non-elementary upper bound on the size of the formula in Gaifman normal form. We can complement this with a non-elementary lower bound (the sequence of formulas is the same as in Proposition 3.2):

▶ **Proposition 4.2.** *Let $\sigma = \{E\}$ with $\mathrm{ar}(E) = 2$. There is a sequence of $\mathrm{FO}[\sigma]$-sentences $(\varphi_h)_{h\geqslant 0}$ of size $\mathcal{O}(h)$ such that for every elementary function $f\colon \mathbb{N} \to \mathbb{N}$, there is $h \in \mathbb{N}$ such that no $\mathrm{FO}(\mathbf{U})$-sentence in Gaifman normal form of length $< f(h)$ is equivalent to φ_h.*

4.2 Non-ultimately periodic quantifiers

Theorem 4.1(2) is an immediate consequence of the following slightly stronger result.

▶ **Proposition 4.3.** *Let $\mathbf{Q}$ be a set of counting quantifiers, let $\mathsf{S} \in \mathbf{Q}$, let $\sigma_\sim = \{\sim\}$ be the signature with $\mathrm{ar}(\sim) = 2$, and let η be the $\mathrm{FO}(\mathbf{Q})$-sentence*

$$\exists x_1\, \exists x_2 \left(\begin{array}{ll} & \neg\,(x_1 \sim x_2) \;\wedge\; \mathsf{S}z\,(x_1 \sim z \;\vee\; x_2 \sim z) \\ \wedge & \forall y\,\big(\,\mathsf{S}z\,(\,x_1 \sim z \;\vee\; y \sim z\,) \;\vee\; \mathsf{S}z\,(\,x_2 \sim z \;\vee\; y \sim z\,)\,\big) \end{array} \right).$$

If S is not ultimately periodic, then no sentence from $\mathrm{FO}(\mathbf{Q})[\sigma_\sim]$ in Gaifman normal form is equivalent to η (not even on the class of finite equivalence structures).

A *finite equivalence structure* is a $\sigma_\sim$-structure $\mathcal{A} = (A, \sim^{\mathcal{A}})$ where A is finite and $\sim^{\mathcal{A}}$ is an equivalence relation on A. In such structures, the formula η expresses that there are two distinct equivalence classes $[a_1]$ and $[a_2]$ with $|[a_1] \cup [a_2]| \in \mathsf{S}$ such that $|[a_1] \cup [b]| \in \mathsf{S}$ or $|[a_2] \cup [b]| \in \mathsf{S}$ for any equivalence class $[b]$. We will prove that this property cannot be expressed by any sentence in Gaifman normal form.

Let $\lambda(x)$ be a local $\mathrm{FO}(\mathbf{Q})[\sigma_\sim]$-formula, let $\mathcal{A}$ be a finite equivalence structure, and let $a \in A$. Since λ is local, the question whether or not $\mathcal{A} \models \lambda(a)$ is determined solely by the size of the equivalence class $[a]$. In case that $\lambda(x)$ is a first-order formula, it is equivalent to a Boolean combination of statements of the form $|[a]| \geqslant k$ for $k \in \{0, \ldots, r\}$, where r is the formula's quantifier rank. If the formula $\lambda(x)$ also uses counting quantifiers from $\mathbf{Q}$, then we can also express that $|[a]| - \ell \in \mathsf{Q}$, for $\mathsf{Q} \in \mathbf{Q}$; but this is only possible for $\ell \in \{0, \ldots, r'\}$, where r' is a number that only depends on the formula λ, but not on the considered equivalence structure $\mathcal{A}$. These observations lead to the following limitation of the expressiveness of local formulas:

▶ **Lemma 4.4.** *For every local $\mathrm{FO}(\mathbf{Q})[\sigma_\sim]$-formula $\lambda(x)$ there exists an $r \in \mathbb{N}$ such that the following is true for all finite equivalence structures $\mathcal{A}$ and $\mathcal{B}$, all $a \in A$, and all $b \in B$.*
If the equivalences

$$|[a]| \geqslant \ell \iff |[b]| \geqslant \ell \qquad \text{and} \qquad |[a]| - \ell \in \mathsf{Q} \iff |[b]| - \ell \in \mathsf{Q}$$

are satisfied for all $\ell \in \{0, \ldots, r\}$ and all counting quantifiers $\mathsf{Q} \in \mathbf{Q}$ that appear in λ, then $\mathcal{A} \models \lambda(a) \iff \mathcal{B} \models \lambda(b)$.

Proof of Proposition 4.3. Suppose S is not ultimately periodic and suppose, for contradiction, that γ is an $\mathrm{FO}(\mathbf{Q})[\sigma_\sim]$-sentence in Gaifman normal form that is equivalent to η. Let $\mathbf{Q}_\gamma$ consist of all counting quantifiers that appear in γ. We apply Lemma 4.4 to all local formulas $\lambda(x)$ that occur in basic-local sentences or in counting sentences which are

subformulas of γ, and we let $\hat{r}$ be the maximum of all the numbers r provided by Lemma 4.4 for each such $\lambda(x)$.

Since $\mathsf{Q}_\gamma \cup \{\mathsf{S}\}$ is finite and S is not ultimately periodic, one can prove the existence of natural numbers $m, n \geqslant \hat{r}$ and $k \geqslant 1$ such that $m + k \notin \mathsf{S}$, $n + k \in \mathsf{S}$, and for all $\mathsf{Q} \in \mathbf{Q}_\gamma \cup \{\mathsf{S}\}$ and all $\ell \in \{0, \dots, \hat{r}\}$ we have $m - \ell \in \mathsf{Q} \iff n - \ell \in \mathsf{Q}$.

We use the numbers $m, n, k, \hat{r}$ to define two finite equivalence structures $\mathcal{B}$ and $\mathcal{C}$:

- $\mathcal{B}$ has $\hat{r}$ many equivalence classes of size k and m many equivalence classes of size n.
- $\mathcal{C}$ has $\hat{r}$ many equivalence classes of size k and n many equivalence classes of size m.

By Lemma 4.4, no local formula appearing in γ can distinguish equivalence classes of size m from equivalence classes of size n. It follows that $\mathcal{B} \models \gamma \iff \mathcal{C} \models \gamma$.

To prove that $\mathcal{B} \models \eta$, one chooses for x_1 and x_2 elements from equivalence classes of size k and n, respectively. When trying to satisfy η in $\mathcal{C}$, one has to choose for x_1 and x_2 elements from equivalence classes of the same size since $m + k \notin \mathsf{S}$. But there is some y whose equivalence class has different size, and hence $|[x_i]| + |[y]|$ cannot belong to S. Thus, η distinguishes $\mathcal{B}$ from $\mathcal{C}$, but γ does not. ◀

5 Finitely d-equivalent Gaifman normal forms

5.1 Ultimately periodic quantifiers

In Section 4.1 we obtained an algorithm that transforms a given FO(**U**)-formula over a relational signature into an equivalent FO(**U**)-formula in Gaifman normal form. Just as in Gaifman's original locality theorem, the algorithm's runtime is non-elementary in the size of the input formula; and from Proposition 4.2 we know that a non-elementary blow-up in formula size (and hence also runtime) cannot be avoided.

In [16] it was shown that for plain first-order logic FO, the non-elementary blow-up can be improved into a (worst-case optimal) 3-fold exponential running time if we drop the requirement that the Gaifman normal form formula has to be equivalent to the original formula on *all* structures and are content with a *finitely d-equivalent* formula in Gaifman normal form. We can generalise this result to FO(**U**) as follows.

▶ **Theorem 5.1.** *Upon input of a number $d \in \mathbb{N}$ and an* FO(**U**)*-formula φ over some relational signature σ, a finitely d-equivalent formula ψ in Gaifman normal form can be computed in time $2^{d^{2^{O(\|\varphi\|)}}}$ for $d \geqslant 3$, and in time $2^{2^{\mathrm{poly}(\|\varphi\|)}}$ for $d < 3$.*
Furthermore, ψ uses at most the quantifiers from φ and the quantifier $\exists$.

We proceed in the same way as the proof of [16], but instead of building upon the Hanf normal form algorithm for FO of [2] we build upon the Hanf normal form algorithm for FO(**U**) of [17]. For the precise statement of the result of [17], we need the following notation.

Let σ be a relational signature. For every $r \in \mathbb{N}$ and $n \in \mathbb{N}_{\geqslant 1}$, a *type with n centres and radius at most r* is structure of the form $\tau = (\mathcal{N}_r^{\mathcal{A}}(\overline{a}), \overline{a})$ where $\mathcal{A}$ is a σ-structure and $\overline{a} \in A^n$. Such a type is called d-bounded if the structure $\mathcal{N}_r^{\mathcal{A}}(\overline{a})$ is d-bounded.

The following is straightforward. The universe of a d-bounded type τ with n centres and radius $\leqslant r$ has size at most $n \cdot d^{r+1}$ (provided that $d \geqslant 2$). Given τ and r, one can construct an FO[σ]-formula[2] $\mathrm{sph}_\tau(\overline{x})$ with n free variables $\overline{x} = (x_1, \dots, x_n)$ such that for every σ-structure $\mathcal{A}$ and every tuple $\overline{a} \in A^n$ we have $\mathcal{A} \models \mathrm{sph}_\tau(\overline{a}) \iff (\mathcal{N}_r^{\mathcal{A}}(\overline{a}), \overline{a}) \cong \tau$. We

[2] The formula $\mathrm{sph}_\tau(\overline{x})$ also depends on r, although this is not reflected by the notation here.

can assume w.l.o.g. that the formula $\mathrm{sph}_\tau(\overline{x})$ is r-local and has size at most $(n{\cdot}d^{r+1})^{\mathcal{O}(\|\sigma\|)}$, where $\|\sigma\|$ is defined as the sum of the arities of the relation symbols in σ.

Formulas of the form $\mathrm{sph}_\tau(\overline{x})$ are called (d-bounded) *sphere-formulas* of signature σ. Let $\mathbf{Q}$ be a set of counting quantifiers. An FO($\mathbf{Q}$)-*Hanf-sentence* of signature σ is a sentence of the form $(\mathsf{Q}{+}k)y\,\mathrm{sph}_\rho(y)$ or of the form $\exists^{\geqslant k}y\,\mathrm{sph}_\rho(y)$, where $k \in \mathbb{N}$, $\mathsf{Q} \in \mathbf{Q}$, and ρ is a type of signature σ and with a single centre. An FO($\mathbf{Q}$)-formula in *Hanf normal form* and of signature σ is a Boolean combination of sphere-formulas and FO($\mathbf{Q}$)-Hanf-sentences of signature σ. The proof of Theorem 5.1 follows by combining Lemma 2.4(c) and:

▶ **Theorem 5.2** ([17]). *Upon input of a number $d \in \mathbb{N}$ and an* FO($\mathbf{U}$)*-formula φ over some relational signature σ, a finitely d-equivalent* FO($\mathbf{U}$)*-formula ψ in Hanf normal form and of signature σ can be computed in time $2^{d^{2^{\mathcal{O}(\|\varphi\|)}}}$ for $d \geqslant 3$, and in time $2^{2^{\mathrm{poly}(\|\varphi\|)}}$ for $d < 3$.*

5.2 General quantifiers

In Section 4.2 we showed that if a set $\mathbf{Q}$ contains a quantifier that is *not* ultimately periodic, then there is an FO($\mathbf{Q}$)-sentence that is not equivalent to any FO($\mathbf{Q}$)-sentence in Gaifman normal form (not even on the class of finite structures). Somewhat surprisingly, it turns out that if we drop the requirement that the Gaifman normal form formula has to be equivalent to the original formula on *all* structures and are content with a *finitely d-equivalent* formula, Gaifman normal forms do exist for *arbitrary* sets $\mathbf{Q}$ of counting quantifiers. Precisely, we obtain the following result, in which the size $\|\varphi\|$ of an FO($\mathbf{Q}$)-formula φ of signature σ is defined analogously as the size of FO($\mathbf{U}$)-formulas, but now each quantifier $\mathsf{Q} \in \mathbf{Q}$ is viewed as an abstract symbol of length 1.

▶ **Theorem 5.3.** *Let $\mathbf{Q}$ be an arbitrary set of counting quantifiers and let $d \in \mathbb{N}$. For every* FO($\mathbf{Q}$)*-formula φ over some relational signature σ, there exists a finitely d-equivalent* FO($\mathbf{Q}$)*-formula ψ in Gaifman normal form. Moreover, if the sets $\mathsf{Q} \in \mathbf{Q}$ are uniformly decidable (in elementary time), then ψ can be computed from φ and d (in elementary time).*

The proof proceeds in a similar way as the proof of Theorem 5.1, but instead of building upon Theorem 5.2, it uses a result of [19] that can be viewed as a generalisation of Theorem 5.2 to FO($\mathbf{Q}$) for arbitrary sets $\mathbf{Q}$ of unary counting quantifiers. For the precise statement of this result, we need the following notation. An FO($\mathbf{Q}$)-*weak-Hanf-sentence* of signature σ is a sentence of the form $(\mathsf{Q}{+}k)y\,\bigvee_{\theta\in T}\theta(y)$ or of the form $\exists^{\geqslant k}y\,\bigvee_{\theta\in T}\theta(y)$, where T is a finite set of sphere-formulas of signature σ, each of them with a single centre and all of the same radius r. An FO($\mathbf{Q}$)-formula in *weak Hanf normal form* and of signature σ is a Boolean combination of sphere-formulas and FO($\mathbf{Q}$)-weak-Hanf-sentences of signature σ. The proof of Theorem 5.3 follows by combining Lemma 2.4(c) and:

▶ **Theorem 5.4** ([19]). *Let $\mathbf{Q}$ be an arbitrary set of counting quantifiers and let $d \in \mathbb{N}$. For every* FO($\mathbf{Q}$)*-formula φ over some relational signature σ, there exists a finitely d-equivalent* FO($\mathbf{Q}$)*-formula ψ in weak Hanf normal form and of signature σ. Moreover, if the sets $\mathsf{Q} \in \mathbf{Q}$ are uniformly decidable (in elementary time), then ψ can be computed from φ and d (in elementary time).*

One may wonder if, analogously to the statement of Theorem 5.1, the last statement of Theorem 5.3 can be improved to a 3-fold exponential running time. To refute this, one observes that from a formula in Gaifman normal form, one can construct an equivalent formula in weak Hanf normal form with the same number of counting sentences. Then a lower bound result of [19] for weak Hanf normal forms implies the following:

▶ **Proposition 5.5.** *There exists a* $\mathsf{Q} \subseteq \mathbb{N}$ *such that for* $\mathbf{Q} := \{\mathsf{Q}\}$*, there is a sequence* $(\varphi_n)_{n\geqslant 1}$ *of* $\mathrm{FO}(\mathbf{Q})$*-sentences of the same relational signature and of size* $\mathcal{O}(n)$ *such that, for all* $n \geqslant 1$*, every* $\mathrm{FO}(\mathbf{Q})$*-sentence in Gaifman normal form that is finitely 3-equivalent to* φ_n *contains at least* $\exp_4(n)$ *distinct subformulas of the form* $(\mathsf{Q}{+}k)y\,\lambda(y)$*.*

From our proof it follows that $\{n^n : n \in \mathbb{N}\}$, $\{n! : n \in \mathbb{N}\}$, and $\{\lfloor 2^{n^c} \rfloor : n \in \mathbb{N}\}$ for all reals $c > 1$, are examples of sets Q for which the statement of Proposition 5.5 holds.

6 An algorithmic meta-theorem for FO(U)

The model-checking problem for a logic L and a class $\mathcal{C}$ of finite relational structures receives as input a sentence $\varphi \in \mathrm{L}$ and a structure $\mathcal{A} \in \mathcal{C}$, and the task is to decide if $\mathcal{A} \models \varphi$. This problem is said to be *fixed-parameter tractable* if it can be solved in time $f(\|\varphi\|)\cdot\mathrm{poly}(\|\mathcal{A}\|)$ where f is a computable function, $\|\varphi\|$ is the size of the formula, and $\|\mathcal{A}\|$ is the size of the structure (defined as $\|\mathcal{A}\| := |A| + \sum_{R\in\sigma} \mathrm{ar}(R)\cdot|R^{\mathcal{A}}|$). Recall from Section 1 the list of examples of logics L and classes $\mathcal{C}$ for which the model-checking problem is known to be fixed-parameter tractable. The aim of this section is to demonstrate that by using our Gaifman normal form result for $\mathrm{FO}(\mathbf{U})$ (Theorem 4.1(1)), the model-checking algorithm for classes of bounded local tree-width of [9] can be generalised from FO to $\mathrm{FO}(\mathbf{U})$.

To provide a precise formulation of the result, we need some further notation. We assume that the reader is familiar with the basic concept of a tree-decomposition and the tree-width $\mathrm{tw}(\mathcal{A})$ of a structure $\mathcal{A}$ (precise definitions can be found in [9] and will not be necessary for understanding the remainder of this section). The *local tree-width* of $\mathcal{A}$ is the function $\mathrm{ltw}^{\mathcal{A}}\colon \mathbb{N} \to \mathbb{N}$ defined by $\mathrm{ltw}^{\mathcal{A}}(r) := \max\{\mathrm{tw}(\mathcal{N}_r^{\mathcal{A}}(a)) : a \in A\}$ for all $r \in \mathbb{N}$. A class $\mathcal{C}$ of structures has *(effectively) bounded local tree-width* if there is a (computable) function $g\colon \mathbb{N} \to \mathbb{N}$ such that $\mathrm{ltw}^{\mathcal{A}}(r) \leqslant g(r)$ for all $\mathcal{A} \in \mathcal{C}$ and all $r \in \mathbb{N}$. As shown in [9], examples for classes of bounded local tree-width are classes of trees, classes of structures of tree-width at most w (for each fixed $w \in \mathbb{N}$), classes of degree at most d (for each fixed $d \in \mathbb{N}$), the class of planar graphs, and classes of graphs of genus at most g (for each fixed $g \in \mathbb{N}$).

The overall approach of [9] has been described in [12] as follows: *"Using Gaifman's theorem, the problem to decide whether a general first-order formula φ is true in a graph can be reduced to testing whether a formula is true in r-neighbourhoods in the graph, where the radius r only depends on φ, and solving a variant of the (distance d) independent set problem. Hence, if $\mathcal{C}$ is a class of graphs where r-neighbourhoods have a simple structure, such as the class of planar graphs or classes of bounded local tree-width, this method gives an easy way for deciding properties definable in first-order logic."*

Here, the "(distance d) independent set problem" corresponds to the essence of evaluating a basic local sentence. Our Gaifman normal form for $\mathrm{FO}(\mathbf{U})$-sentences consists of basic local sentences (which can be evaluated in the same way as described in [9]) and counting sentences of the form $(\mathsf{Q}{+}k)x\,\lambda(x)$, and evaluating these boils down to (1) computing the set of all nodes x whose r-neighbourhood satisfies $\lambda(x)$ and (2) checking if the size of this set belongs to $(\mathsf{Q}{+}k)$. The task (1) has been solved in [9] for r-local FO-formulas and can easily be gerneralised to r-local $\mathrm{FO}(\mathbf{U})$-formulas, and the task (2) is straightforward. In summary, by combining the approach of [9] with our Theorem 4.1(1) we obtain:

▶ **Corollary 6.1.** *Let $\mathcal{C}$ be a class of finite relational structures of bounded local tree-width and let φ be an $\mathrm{FO}(\mathbf{U})$-sentence. Then, for every $k \geqslant 1$, there is an algorithm deciding in time $\mathcal{O}(\|A\|^{1+(1/k)})$ whether a given structure $\mathcal{A} \in \mathcal{C}$ satisfies φ.*

To keep the runtime analysis of Corollary 6.1 simple, we formulated the corollary in a way which uses the $\mathcal{O}$-notation to hide factors that depend on the sentence φ or the number k. A closer inspection of the proof shows that, for any class of effectively bounded local tree-width, the algorithm's runtime can be bounded by $f(\|\varphi\|, k) \cdot \|\mathcal{A}\|^{1+(1/k)}$, for some computable function f. Thus, in particular, we obtain that for every class $\mathcal{C}$ of effectively bounded local tree-width, the model-checking problem for FO(**U**)-sentences on $\mathcal{C}$ is fixed-parameter tractable. To close this paper, let us mention that we believe that by a similar, but substantially more involved construction also the result of [12] for model-checking on nowhere dense classes can be lifted from FO to FO(**U**) – we plan to do this as future work.

References

1. S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991. doi:10.1016/0196-6774(91)90006-K.
2. B. Bollig and D. Kuske. An optimal construction of Hanf sentences. *J. Applied Logic*, 10(2):179–186, 2012. doi:10.1016/j.jal.2012.01.002.
3. B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.
4. A. Dawar, M. Grohe, S. Kreutzer, and N. Schweikardt. Model theory makes formulas large. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming, ICALP 2007, Wroclaw, Poland, July 9-13, 2007*, pages 913–924, 2007. doi:10.1007/978-3-540-73420-8_78.
5. J. Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5):406–451, 1970. doi:10.1016/S0022-0000(70)80041-1.
6. Z. Dvorak, D. Král, and R. Thomas. Deciding first-order properties for sparse graphs. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 133–142, 2010. doi:10.1109/FOCS.2010.20.
7. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer, 1995.
8. S. Feferman and R. L. Vaught. The first-order properties of products of algebraic systems. *Fundamenta Mathematicae*, 47:57–103, 1959.
9. M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001. doi:10.1145/504794.504798.
10. H. Gaifman. On local and nonlocal properties. In J. Stern, editor, *Logic Colloquium '81*, pages 105–135. North-Holland, 1982.
11. M. Grohe. Logic, graphs, and algorithms. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 357–422. Amsterdam University Press, 2008.
12. M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.
13. F. Harwath, L. Heimberg, and N. Schweikardt. Preservation and decomposition theorems for bounded degree structures. *Logical Methods in Computer Science*, 11(4), 2015. doi:10.2168/LMCS-11(4:17)2015.
14. F. Harwath and N. Schweikardt. On the locality of arb-invariant first-order formulas with modulo counting quantifiers. *Logical Methods in Computer Science*, 12(4), 2016. doi:10.2168/LMCS-12(4:8)2016.
15. L. Heimberg. *Complexity of normal forms on structures of bounded degree*. PhD thesis, Humboldt-Universität zu Berlin, 2017.

16 L. Heimberg, D. Kuske, and N. Schweikardt. An optimal Gaifman normal form construction for structures of bounded degree. In *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 63–72, 2013. doi:10.1109/LICS.2013.11.

17 L. Heimberg, D. Kuske, and N. Schweikardt. Hanf normal form for first-order logic with unary counting quantifiers. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2016, New York, NY, USA, July 5-8, 2016*, pages 277–286, 2016. doi:10.1145/2933575.2934571.

18 L. Hella, L. Libkin, and J. Nurmonen. Notions of locality and their logical characterizations over finite models. *J. Symb. Log.*, 64(4):1751–1773, 1999. doi:10.2307/2586810.

19 D. Kuske and N. Schweikardt. First-order logic with counting. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. (Full version available at https://arxiv.org/abs/1703.01122). doi:10.1109/LICS.2017.8005133.

20 L. Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

21 J. A. Makowsky. Algorithmic uses of the Feferman-Vaught Theorem. *Ann. Pure Appl. Logic*, 126(1-3):159–213, 2004. doi:10.1016/j.apal.2003.11.002.

22 A. B. Matos. Periodic sets of integers. *Theor. Comput. Sci.*, 127(2):287–312, 1994. doi:10.1016/0304-3975(94)90044-2.

23 H. Niemistö. *Locality and Order-Invariant Logics*. PhD thesis, Department of Mathematics and Statistics, University of Helsinki, 2007.

24 J. Nurmonen. Counting modulo quantifiers on finite structures. *Inf. Comput.*, 160(1-2):62–87, 2000. doi:10.1006/inco.1999.2842.

25 D. Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996.

26 H. Straubing. *Finite automata, formal logic, and circuit complexity*. Progress in Theoretical Computer Science. Birkhäuser Boston Inc., Boston, MA, 1994.

27 J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968. doi:10.1007/BF01691346.

Polynomial Vector Addition Systems With States

Jérôme Leroux[1]
Univ.Bordeaux, CNRS, Bordeaux-INP, Talence, France
jerome.leroux@labri.fr

Abstract

The reachability problem for vector addition systems is one of the most difficult and central problems in theoretical computer science. The problem is known to be decidable, but despite intense investigation during the last four decades, the exact complexity is still open. For some sub-classes, the complexity of the reachability problem is known. Structurally bounded vector addition systems, the class of vector addition systems with finite reachability sets from any initial configuration, is one of those classes. In fact, the reachability problem was shown to be polynomial-space complete for that class by Praveen and Lodaya in 2008. Surprisingly, extending this property to vector addition systems with states is open. In fact, there exist vector addition systems with states that are structurally bounded but with Ackermannian large sets of reachable configurations. It follows that the reachability problem for that class is between exponential space and Ackermannian. In this paper we introduce the class of polynomial vector addition systems with states, defined as the class of vector addition systems with states with size of reachable configurations bounded polynomially in the size of the initial ones. We prove that the reachability problem for polynomial vector addition systems is exponential-space complete. Additionally, we show that we can decide in polynomial time if a vector addition system with states is polynomial. This characterization introduces the notion of iteration scheme with potential applications to the reachability problem for general vector addition systems.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Vector additions system with states, Reachability problem, Formal verification, Infinite state system, Linear algebra, Kosaraju-Sullivan algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.134

1 Introduction

Vector addition systems or equivalently Petri nets are one of the most popular formal methods for the representation and the analysis of parallel processes [4]. The reachability problem is central since many computational problems (even outside the parallel processes) reduce to the reachability problem. In 1981, Mayr [13] provided the first decidability proof of the reachability problem. Later, that proof was first simplified by Kosaraju [7], and then ten years later by Lambert [9]. This last proof still remains difficult and the complexity upper bound of the corresponding algorithm is just known to be non-primitive recursive [11]. Nowadays, up to some details, there are only two different known reachability algorithms for general vector addition systems; one based on the Kosaraju-Lambert-Mayr (KLM) decomposition; and a recent one based on Presburger inductive invariants [10]. Despite intense investigation during the last four decades, it is still an open problem whether an elementary complexity upper bound for the reachability problem exists.

[1] The author was supported by the grant ANR-17-CE40-0028 of the French National Research Agency ANR (project BRAVAS)

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 134; pp. 134:1–134:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

When the reachability set of a vector addition system is finite, the KLM decomposition degenerates and it just corresponds to the regular language of all possible executions of the vector addition system from an initial configuration to a final one. Even in that case, the complexity of the KLM algorithm is Ackermannian and no better complexity upper bound are known.

In 2008, Praveen and Lodaya proved that the reachability problem for structurally bounded vector addition systems, the class of vector addition systems with finite reachability sets from any initial configuration is polynomial-space complete [17]. Surprisingly, extending this property to vector addition systems with states is open. In fact, there exist vector addition systems with states that are structurally bounded but with Ackermannian large sets of reachable configurations. It follows that the reachability problem for that class is between exponential space and Ackermannian.

Intuitively, for structurally bounded vector addition systems with states, the KLM algorithm fails to avoid enumerating all the possible reachable configurations since it tries to detect cycles of edges that can be iterated to obtain arbitrarily large components (such a cycle cannot exists due to the structurally bounded condition). Characterizing indexes that can be very large but not necessarily arbitrarily large should provide new insights on how to overcome the Ackermannian complexity of the KLM algorithm.

Our contributions

In this paper we introduce the class of *polynomial vector addition systems with states* defined as the vector addition systems with states such that reachable configurations have sizes polynomially bounded with respect to sizes of initial configurations. We prove that a vector addition system with states is not polynomial if, and only if, it contains a so-called *iteration scheme* that can increase some components. We prove that we can decide in polynomial time if a vector addition system with states is polynomial, and we show that the reachability problem for polynomial vector addition systems with states is exponential-space complete. Up to our knowledge, our notion of iteration scheme is new and provide a potential application to patch the KLM algorithm.

Outline

In Section 2 we introduce vector addition systems with states (VASS for short), and the subclass of polynomial VASS. Iteration schemes are defined in Section 3. Intuitively iteration schemes are sequences of cycles that can be iterated many times (at least an exponential number of times). Indexes that can be increased by an iteration scheme are called iterable indexes, and edges that occur in iteration schemes are called iterable edges. We show that reachable configurations cannot be polynomially bounded with respect to the size of the initial configurations on any iterable index. It follows that VASS with iterable indexes cannot be polynomial. In Section 4, we recall some general properties about the Kirchoff's functions and the Euler's lemma. Those definitions are used in Section 5 to prove the correctness of a polynomial-time algorithm inspired by the Kosaraju-Sullivan algorithm for computing the set of iterable indexes and the set of iterable edges. In Section 6 we show that reachable configurations are polynomially bounded on the non-iterable indexes with respect to the size of the initial configurations. Finally in Section 7 we show that we can decide in polynomial time if a VASS is polynomial and we prove that the reachability problem for polynomial VASS is exponential-space complete.

2 Polynomial Vector Addition Systems With States

In this section we first introduce the vector addition systems, and the structurally bounded ones. Then, we recall how the reachability problem for that subclass can be solved in polynomial space[17]. Next, we introduce the vector addition systems with states (VASS) and we show that the previous approach for VAS no longer apply to VASS. Finally, we introduce the class of polynomial VASS, the main class of VASS studied in this paper.

Concerning notations used in this paper, we denote by $\mathbb{Z}, \mathbb{N}, \mathbb{Q}$ the set of *integers*, *natural numbers*, and *rational numbers*. The absolute value of a *rational* $\lambda \in \mathbb{Q}$ is denoted by $|\lambda|$. Let $d \in \mathbb{N}$ be a natural number. *Vectors* in $\mathbb{Q}^d$ are denoted in bold face, and we denote by $\mathbf{v}[1], \ldots, \mathbf{v}[d]$ the components of $\mathbf{v}$, i.e. $\mathbf{v} = (\mathbf{v}[1], \ldots, \mathbf{v}[d])$. Every operations are performed component-wise on the vectors; for instance the sum $\mathbf{x} + \mathbf{y}$ of two vectors in $\mathbb{Q}^d$ is the vector $\mathbf{z}$ in $\mathbb{Q}^d$ satisfying $\mathbf{z}[i] = \mathbf{x}[i] + \mathbf{y}[i]$ for every $i \in \{1, \ldots, d\}$. We write $\mathbf{x} \leq \mathbf{y}$ if $\mathbf{x}[i] \leq \mathbf{y}[i]$ for every $1 \leq i \leq d$, and we write $\mathbf{x} < \mathbf{y}$ if $\mathbf{x} \leq \mathbf{y}$ and $\mathbf{x} \neq \mathbf{y}$. We denote by $\mathbf{u}_i$ the ith *unit vector* of $\mathbb{Q}^d$ defined by $\mathbf{u}_i[j] = 0$ if $j \neq i$ and $\mathbf{u}_i[i] = 1$. Notice that $\mathbf{x} = \sum_{i=1}^{d} \mathbf{x}[i]\mathbf{u}_i$ with our notations. The *norm* of a vector $\mathbf{v} \in \mathbb{Q}^d$ is the rational number $||\mathbf{v}|| = \max_i |\mathbf{v}[i]|$. The *norm* of a finite set $\mathbf{V} \subseteq \mathbb{Q}^d$ is defined as $||\mathbf{V}|| = \max_{\mathbf{v} \in \mathbf{V}} ||\mathbf{v}||$. We introduce the sets $|\mathbf{v}|^+ = \{i \mid \mathbf{v}[i] > 0\}$ and $|\mathbf{v}|^- = \{i \mid \mathbf{v}[i] < 0\}$. The *dot product* of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{Q}^d$ is the rational number $\sum_i \mathbf{x}[i] \cdot \mathbf{y}[i]$ denoted as $\mathbf{x} \cdot \mathbf{y}$.

2.1 Vector Addition Systems

A *vector addition system* (*VAS* for short) is a non-empty finite set $\mathbf{A} \subseteq \mathbb{Z}^d$ of *actions*. A vector in $\mathbb{N}^d$ is called a *configuration* of the VAS $\mathbf{A}$. The *semantics* is defined thanks to the binary relation $\to$ over the configurations by $\mathbf{x} \to \mathbf{y}$ if $\mathbf{y} = \mathbf{x} + \mathbf{a}$ for some action $\mathbf{a} \in \mathbf{A}$. The reflexive and transitive closure of $\to$ is denoted as $\xrightarrow{*}$ and it is called the *reachability relation*. If $\mathbf{x} \xrightarrow{*} \mathbf{y}$, we say that $\mathbf{y}$ is *reachable* from $\mathbf{x}$.

The reachability problem consists in deciding for a triple $(\mathbf{x}, \mathbf{A}, \mathbf{y})$ where $\mathbf{x}, \mathbf{y}$ are two configurations of a VAS $\mathbf{A}$, if $\mathbf{x} \xrightarrow{*} \mathbf{y}$. The problem is decidable [14] but its complexity remains elusive; the problem is known to be exponential-space hard [12], and the best known upper bound is non-primitive recursive [11].

A VAS $\mathbf{A}$ is said to be *bounded* from an initial configuration $\mathbf{x}$ if the set of configurations reachable from $\mathbf{x}$ is finite. The boundedness problem is known to be exponential-space complete [18]. Since the size of reachable configurations are at most Ackermannian in that case [15, 5], it follows that the reachability problem can be decided in Ackermannian complexity (space and time are equivalent for that class of complexity). This is the best known upper bound, far from the exponential-space lower bound [12].

When enforcing the VAS to be bounded for any initial configuration, we obtain the so-called structurally bounded VAS. More formally, a VAS is said to be *structurally bounded* if it is bounded from any initial configuration. In polynomial time, one can decide if a VAS is structurally bounded. In fact, a VAS $\mathbf{A}$ is not structurally bounded if, and only if, the following linear system is satisfiable over the non negative rational numbers: $(\lambda_{\mathbf{a}})_{\mathbf{a} \in \mathbf{A}}$:

$$\sum_{\mathbf{a} \in \mathbf{A}} \lambda_{\mathbf{a}} \mathbf{a} > \mathbf{0}$$

The previous observation combined with the Farkas Lemma [19] shows that a VAS is structurally bounded if, and only if, there exists a vector $\mathbf{v}$ in $\mathbb{N}^d$, called a *place invariant* such that $\mathbf{v}[i] > 0$ for every i, and such that $\mathbf{v} \cdot \mathbf{a} \leq 0$ for every action $\mathbf{a}$ in $\mathbf{A}$.

▶ **Example 1.** The VAS $\mathbf{A} = \{2\mathbf{u}_{i+1} - \mathbf{u}_i \mid 1 \leq i < d\}$ admits the place invariant $\mathbf{v} \in \mathbb{N}^d$ defined by $\mathbf{v}[i] = 2^{d-i}$ for every i.

Notice that if $\mathbf{x} \xrightarrow{*} \mathbf{y}$ then $\mathbf{v} \cdot \mathbf{y} \leq \mathbf{v} \cdot \mathbf{x}$ for any place invariant $\mathbf{v}$. We deduce that $||\mathbf{y}|| \leq (\sum_{i=1}^{d} \mathbf{v}[i])||\mathbf{x}||$. The norm of reachable configurations is therefore bounded linearly in the norm of the initial one. As observed by Praveen and Lodaya [17], the norm of the vector $\mathbf{v}$ can be bounded thanks to a small solution theorem of Borosh and Treybig [1] in such a way the reachability problem for structurally bounded VAS is decidable in polynomial space. Based on the reduction of QBF to the reachability problem of structurally bounded VAS, Praveen and Lodaya deduced the following result.

▶ **Theorem 2** (Theorem 3.10 and Theorem 3.11 of [17])**.** *The reachability problem for structurally bounded VAS is polynomial-space complete.*

2.2 Vector Addition Systems With States

The previous approach no longer apply for structurally bounded vector addition systems with states. Formally, a *vector addition systems with states* (*VASS* for short) is a graph $G = (Q, \mathbf{A}, E)$ where Q is a non empty finite set of states, $\mathbf{A}$ is a VAS, and $E \subseteq Q \times \mathbf{A} \times Q$ is a finite set of *edges*. A *configuration* is a pair $(q, \mathbf{x})$ in $Q \times \mathbb{N}^d$ denoted as $q(\mathbf{x})$ in the sequel. The semantics of an edge e is defined thanks to the binary relation $\xrightarrow{e}$ over the configurations by $p(\mathbf{x}) \xrightarrow{e} q(\mathbf{y})$ if $e = (p, \mathbf{y} - \mathbf{x}, q)$. We associate to a word $\pi = e_1 \dots e_k$ of edges the binary relation $\xrightarrow{\pi}$ over the configurations defined as the following composition:

$$\xrightarrow{e_1} \cdots \xrightarrow{e_k}$$

Notice that $\xrightarrow{\varepsilon}$ is the identity binary relation. The *reachability relation* of a VASS G is the binary relation $\xrightarrow{*}$ defined as the union $\bigcup_{\pi \in E^*} \xrightarrow{\pi}$. A configuration $q(\mathbf{y})$ is said to be *reachable* from a configuration $p(\mathbf{x})$ if $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$.

The following lemma states the so-called *VASS monotony property*. We refer to that lemma when we mention a monotony property in the sequel.

▶ **Lemma 3.** *We have $p(\mathbf{x} + \mathbf{c}) \xrightarrow{\pi} q(\mathbf{y} + \mathbf{c})$ for every $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$ and for every $\mathbf{c} \in \mathbb{N}^d$.*

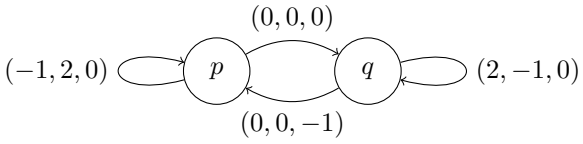
Proof. By induction on the length of π. ◀

We associate to a VASS $G = (Q, \mathbf{A}, E)$ the functions $\mathrm{src}, \mathrm{tgt} : E \to Q$ and $\Delta : E \to \mathbf{A}$ satisfying $e = (\mathrm{src}(e), \Delta(e), \mathrm{tgt}(e))$ for every $e \in E$. The states $\mathrm{src}(e)$ and $\mathrm{tgt}(e)$ are respectively called the *source* and *target* states. The vector $\Delta(e)$ is called the *displacement* of e. We extend the displacement function to words $\pi = e_1 \dots e_k$ of edges by $\Delta(\pi) = \Delta(e_1) + \cdots + \Delta(e_k)$. Given a sequence $\pi_1, \dots, \pi_k$ of words of edges, the vector $\Delta(\pi_1) + \cdots + \Delta(\pi_k)$ is also called the *displacement* of the sequence. A word $\pi = e_1 \dots e_k$ of edges is called a *path* of G from a state p to a state q, if there exists a sequence $q_0, \dots, q_k$ of states with $q_0 = p$ and $q_k = q$ such that $(\mathrm{src}(e_j), \mathrm{tgt}(e_j)) = (q_{j-1}, q_j)$ for every $1 \leq j \leq k$. A path is said to be *simple* if $q_i = q_j$ implies $i = j$. A *cycle* on a state q is a path from q to q. A cycle is said to be *elementary* if $q_i = q_j$ and $i < j$ implies $i = 0$ and $j = k$.

Let $T \subseteq E$ be a subset of edges. An edge e of T is said to be *recurrent* for T if there exists a path from $\mathrm{tgt}(e)$ to $\mathrm{src}(e)$ in T, otherwise, it is said to be *transient* for T. We denote by $\mathrm{rec}(T)$ the set of edges of T that are recurrent for T. The set T is said to be *reccurent* if every edge in T is recurrent, i.e. $\mathrm{rec}(T) = T$. We observe $\mathrm{rec}(T)$ is recurrent for any set

T. We associate to a subset T the equivalence relation $\sim_T$ over $\mathrm{rec}(T)$ defined by $e \sim_T e'$ if there exists a path from $\mathrm{tgt}(e)$ to $\mathrm{src}(e')$ and a path from $\mathrm{tgt}(e')$ to $\mathrm{src}(e)$. The equivalence classes of $\sim_T$ are called the strongly connected components of T, and they are denoted as $\mathcal{SCC}(T)$. The set T is said to be *strongly connected* if $\mathcal{SCC}(T) = \{T\}$. We also denote by $\mathcal{SCC}(G)$ the set $\mathcal{SCC}(E)$, and we say that G is strongly connected if E is strongly connected.

▶ **Example 4.** We adapt the VASS introduced in [6] by introducing the following VASS:



Notice that $p(1,0,n) \xrightarrow{*} p(4^n,0,0)$ for every natural number n since for every $n, m \in \mathbb{N}$ with $n \geq 1$, we have:

$$p(m,0,n) \xrightarrow{(p,(-1,2,0),p)^m (p,(0,0,0),q)(q,(2,-1,0),q)^{2m} (q,(0,0,-1),p)} p(4m,0,n-1)$$

A VASS G is said to be *bounded* from an initial configuration $p(\mathbf{x})$ if the reachability set from that configuration is finite. Let us recall that the boundedness problem is decidable in exponential space since the boundedness problem for VASS is logspace reducible to the boundedness for VAS using for instance the encoding of control states as additional counters [6]. A VASS is said to be *structurally bounded* if it is bounded from any initial configuration. Let us recall that a VASS G is not structurally bounded if, and only if, there exists a *cycle* σ such that $\Delta(\sigma) > \mathbf{0}$. Moreover, this property is decidable in polynomial time using the Kosaraju-Sullivan algorithm [8].

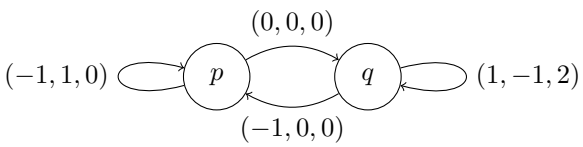
Example 4 shows that reachable configurations of structurally bounded VASS can be exponentially larger than the initial configuration. Unfortunately, it can even be larger by observing that the Ackermannian VASS introduced in [15] are structurally bounded. It follows that the best complexity upper bound for the reachability problem for structurally bounded VASS is Ackermannian. Concerning the lower bound, by observing that Lipton's construction [12, 3] also produces structurally bounded VASS, it follows that the reachability problem is exponential-space hard.

2.3 Polynomial VASS

In this paper we consider a subclass of the structurally bounded VASS, called the polynomial VASS.

▶ **Definition 5.** A VASS is said to be *polynomial* if there exists a polynomial function f such that $||\mathbf{y}|| \leq f(||\mathbf{x}||)$ for every $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$.

▶ **Example 6.** We introduce the following VASS:



We have $p(n,0,0) \xrightarrow{*} p(0,0,n^2+n))$ for every $n \in \mathbb{N}$ since for every $n, m \in \mathbb{N}$ with $m \geq 1$, we have:

$$p(m,0,n) \xrightarrow{(p,(-1,1,0),p)^m (p,(0,0,0),q)(q,(1,-1,2),q)^m (q,(-1,0,0),p)} p(m-1,0,n+2m)$$

We will prove in Example 13 that the VASS is polynomial.

▶ Remark. The VASS given in Example 4 is not polynomial.

We notice that since the Lipton's construction [12, 3] produces polynomial VASS, it follows that the reachability problem is exponential-space hard for polynomial VASS. In this paper we show that (1) we can decide in polynomial time if a VASS is polynomial, (2) the reachability problem is exponential-space complete for polynomial VASS.

3 Iteration Schemes

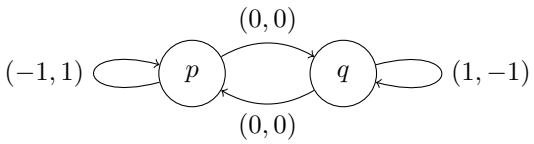
An *iteration scheme* of a VASS G is a finite sequence $\sigma_1, \ldots, \sigma_k$ of cycles such that:

$$\bigwedge_{j=1}^{k} \left[|\Delta(\sigma_j)|^- \subseteq |\Delta(\sigma_1) + \cdots + \Delta(\sigma_k)|^+ \right]$$

Observe that the displacement of an iteration scheme is necessarily a vector in $\mathbb{N}^d$. An index $i \in \{1, \ldots, d\}$ such that there exists an iteration scheme with a displacement strictly positive on i is called an *iterable index*. An edge t that occurs in an iteration scheme is called an *iterable edge*. By concatenating iteration schemes, notice that there exists an iteration scheme with a displacement strictly positive on every iterable index, and such that every iterable edge occurs in the scheme.

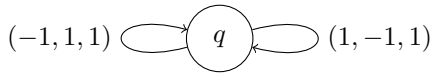
▶ **Example 7.** Let us come back to the VASS introduced in Example 4. Notice that the cycles $(p,(-1,2,0),p)$ and $(q,(2,-1,0),q)$ forms an iteration scheme with a displacement equal to $(1,1,0)$. It follows that the two first indexes are iterable.

▶ **Example 8.** We introduce the following VASS:



Notice that the cycles $(p,(-1,1),p)$ and $(q,(1,-1),q)$ do not form an iteration scheme.

▶ **Example 9.** We cannot restrict iteration schemes to sequences of *elementary* cycles. In fact, let us introduce the following VASS:



Notice that the cycle $(q,(-1,1,1),q)(q,(1,-1,1),q)$ is an iteration scheme proving that the third index is iterable. However notice that any non-empty sequence of elementary cycles is not an iteration scheme.

The following lemma shows that if a strongly connected VASS admits an iteration scheme with a non-zero displacement, then the VASS is not polynomial.

▶ **Lemma 10.** *For every strongly-connected VASS, there exists a rational number $\lambda > 1$ such that for every $n \in \mathbb{N}$ there exists an execution $p_n(\mathbf{x}_n) \xrightarrow{\pi_n} q_n(\mathbf{y}_n)$ such that:*

- $||\mathbf{x}_n|| \geq n$.
- $\mathbf{y}_n[i] \geq \lambda^{||\mathbf{x}_n||}$ *for every iterable index* i.
- *Every iterable edge occurs in* π_n *at least* $\lambda^{||\mathbf{x}_n||}$ *times.*

Proof. The proof is given in a technical report. Intuitively, iteration schemes can be iterated an exponential number of times. ◀

In section 6, we prove that conversely, indexes that are not iterable can be bounded with a polynomial, as well as non iterable edges occur at most a bounded polynomial number of times.

4 Kirchoff's Functions and Euler's Lemma

We recall some classical results about Kirchoff's functions and Euler's lemma. We assume that $G = (Q, \mathbf{A}, E)$ is a VASS.

A *multiset of edges* is a function $\phi : E \to \mathbb{N}$. The set $\{e \in E \mid \phi(e) \neq 0\}$ is called the *domain* of ϕ and it is denoted by $\mathrm{dom}(\phi)$. Given a subset $T \subseteq E$ and a multiset of edges ϕ, we denote by $\phi \cap T$ the multiset of edges defined as follows:

$$\phi \cap T(t) = \begin{cases} \phi(t) & \text{if } t \in T \\ 0 & \text{otherwise} \end{cases}$$

The *Parikh image* of a word π of edges is the multiset of edges ϕ such that $\phi(e)$ is the number of occurrences in π of e for every edge $e \in E$.

The displacement function Δ is extended over the multiset of edges ϕ by $\Delta(\phi) = \sum_{e \in E} \phi(e)\Delta(e)$. The vector $\Delta(\phi)$ are called the *displacement* of ϕ. Notice that if ϕ is the Parikh image of a word π of edges then $\Delta(\phi) = \Delta(\pi)$.

A *Kirchoff* function ϕ is a multiset of edges such that for every $q \in Q$:

$$\sum_{e \in E \mid \mathrm{tgt}(e) = q} \phi(e) = \sum_{e \in E \mid \mathrm{src}(e) = q} \phi(e)$$

Let us recall that a finite sum of Parikh images of cycles is a Kirchoff function and every Kirchoff function is a finite sum of Parikh images of elementary cycles. It follows that the domain of a Kirchoff function is recurrent. The *Euler's Lemma* claims that a Kirchoff function is the Parikh image of a cycle if, and only if, its domain is strongly connected.

5 Computing the Set of Iterable Indexes and Edges

In this section, we show that the set of iterable indexes and the set of iterable edges of a VASS $G = (Q, \mathbf{A}, E)$ are computable in polynomial time. Given a pair (I, T) where I is a subset of $\{1, \ldots, d\}$ and T is a subset of edges, a sequence $\sigma_1, \ldots, \sigma_k$ of cycles of T is called an (I, T)-constrained iteration scheme if

$$\bigwedge_{j=1}^{k} |\Delta(\sigma_j)|^- \subseteq |\Delta(\sigma_1) + \cdots + \Delta(\sigma_k)|^+ \subseteq I$$

We denote by $\Gamma(I,T)$ the pair (I',T') where I' is the set of indexes $i \in I$ such that there exists an (I,T)-constrained iteration scheme with a displacement strictly positive on i, and where T' is the set of edges that occurs in an (I,T)-constrained iteration scheme. Observe that $\Gamma(\{1,\dots,d\},E)$ is the pair (I',T') where I' is the set of iterable indexes and T' is the set of iterable edges.

We are going to compute $\Gamma(I,T)$ inductively by reducing the pair (I,T) into a pair (I',T') such that $\Gamma(I,T) = \Gamma(I',T')$ and such that $\Gamma(I,T) = (I,T)$ if it is not possible to reduce (I,T) anymore. Such an approach is similar to the one used by the Kosaraju-Sullivan algorithm [8, 2] for computing from a VASS the set of edges occurring in cycles with zero displacements. The pair (I',T') is obtained by computing $\Omega(I,T)$ defined as follows.

▶ **Definition 11.** Let (I,T) be a pair such that $I \subseteq \{1,\dots,d\}$ and $T \subseteq E$, and let us consider the following linear system over the variables $(\mu_i)_{i\in I}$ and $(\lambda_t)_{t\in T}$ ranging over the non-negative rational numbers:

$$\bigwedge_{q\in Q} \sum_{t\in T\mid \mathrm{tgt}(t)=q} \lambda_t = \sum_{t\in T\mid \mathrm{src}(t)=q} \lambda_t$$

$$\bigwedge_{S\in \mathcal{SCC}(T)} \bigwedge_{i\notin I} \sum_{t\in S} \lambda_t \Delta(t)[i] = 0$$

$$\bigwedge_{i\in I} \sum_{t\in T} \lambda_t \Delta(t)[i] = \mu_i$$

Then $\Omega(I,T)$ is defined as the pair (I',T') where:

- I' is the set of indexes $i \in I$ satisfying the previous linear system and $\mu_i > 0$, and
- T' is the set of edges $t \in T$ satisfying the previous linear system and $\lambda_t > 0$.

▶ **Example 12.** Let us come back to Example 4. We observe that $\Omega^n(\{1,2,3\},E)$ is equal to $(\{1,2\},E)$ if $n=1$, and it is equal to $(\{1,2\},\{(p,(-1,2,0),p),(q,(2,-1,0),q)\})$ if $n \geq 2$.

▶ **Example 13.** Let us come back to Example 6. We observe that $\Omega^n(\{1,2,3\},E)$ is equal to $(\{3\},E)$ if $n=1$, $(\{3\},\{(p,(-1,1,0),p),(q,(1,-1,2),q)\})$ if $n=2$, and $(\emptyset,\emptyset)$ if $n \geq 3$.

▶ **Example 14.** Let us come back to Example 8. We observe that $\Omega^n(\{1,2\},E)$ is equal to $(\emptyset,E)$ for every $n \geq 1$.

Notice that $\Omega(I,T)$ is computable in polynomial time and $\Omega(I,T)$ reduces the pair (I,T). The following lemma shows that this reduction let unchanged the value of $\Gamma(I,T)$.

▶ **Lemma 15.** *We have* $\Gamma(I,T) = \Gamma(\Omega(I,T))$.

Proof. Let $(I',T') = \Omega(I,T)$. Since $(I',T') \subseteq (I,T)$ we get $\Gamma(I',T') \subseteq \Gamma(I,T)$. For the converse inclusion, we just have to prove that any (I,T)-constrained iteration scheme $\sigma_1,\dots,\sigma_k$ is an (I',T')-constrained iteration scheme. Let ϕ_j be the Parikh image of σ_j, and let $\phi = \sum_{j=1}^{k} \phi_j$. Notice that $(\mu_i)_{i\in I}$ and $(\lambda_t)_{t\in T}$ defined as $\mu_i = \Delta(\phi)[i]$ and $\lambda_t = \phi(t)$ is a solution of the linear system introduced in Definition 11. It follows that $\mu_i > 0$ implies $i \in I'$ and $\lambda_t > 0$ implies $t \in T'$. Hence $\sigma_1,\dots,\sigma_k$ is an (I',T')-constrained iteration scheme. ◀

Finally, the following lemma shows that $\Gamma(I,T)$ is equal to (I,T) if (I,T) cannot be reduced anymore.

▶ **Lemma 16.** *We have* $\Gamma(I,T) = (I,T)$ *if, and only if,* $\Omega(I,T) = (I,T)$.

Proof. Assume that $\Gamma(I,T) = (I,T)$. Lemma 15 shows that $\Omega(I,T) = (I,T)$. Conversely, assume that $\Omega(I,T) = (I,T)$. By adding many solutions of the linear system introduced in Definition 11, we obtain a solution $(\mu_i)_{i\in I}$ and $(\lambda_t)_{t\in T}$ such that $\mu_i > 0$ for every $i \in I$ and $\lambda_t > 0$ for every $t \in T$. By multiplying that solution by a large natural number, we can assume that the solution is over the natural numbers. Now, just notice that the multiset of edges ϕ defined by $\phi(t) = \lambda_t$ if $t \in T$ and $\phi(t) = 0$ otherwise is a Kirchoff function satisfying $\mathrm{dom}(\phi) = T$, $|\Delta(\phi)|^+ = I$, and such that $\Delta(\phi \cap S)[i] = 0$ for every $S \in \mathcal{SCC}(T)$ and for every $i \notin I$, and such that $\Delta(\phi)[i] \geq 0$ for every $i \in I$. Euler's Lemma shows that for every $S \in \mathcal{SCC}(T)$, there exists a cycle σ_S with a Parikh image equal to $\phi \cap S$. Notice that $|\Delta(\sigma_S)|^- = |\Delta(\phi \cap S)|^- \subseteq I$. Moreover $|\sum_S \Delta(\sigma_S)|^+ = I$. It follows that $(\sigma_S)_{S\in\mathcal{SCC}(T)}$ is an (I,T)-constrained iteration scheme. This scheme shows that $(I,T) \subseteq \Gamma(I,T)$. Therefore $\Gamma(I,T) = (I,T)$. ◀

Now, let us introduce $\Omega^\infty(I,T) = \bigcap_{n\in\mathbb{N}} \Omega^n(I,T)$. Since $(\Omega^n(I,T))_{n\in\mathbb{N}}$ is a non-increasing sequence, there exists $n \leq |I|.|T|$ such that $\Omega^{n+1}(I,T) = \Omega^n(I,T)$ and for such an n, we have $\Omega^\infty(I,T) = \Omega^n(I,T)$. We deduce that $\Omega^\infty(I,T)$ is computable in polynomial time. Moreover, from Lemma 15 and Lemma 16, we deduce that $\Gamma(I,T) = \Omega^\infty(I,T)$. We have proved the following theorem.

▶ **Theorem 17.** *Iterable indexes and iterable edges are computable in polynomial time.*

6 Non-iterable case

In this section we prove the following theorem.

▶ **Theorem 18.** *Let $G = (Q, \mathbf{A}, E)$ be a strongly connected VASS. For every $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$, the values $\mathbf{y}[i]$ where i is a non iterable index, and the number of occurrences of non iterable edges in π are bounded by:*

$$[(1 + ||\mathbf{x}||)^2 d^2 (3||\mathbf{A}||\,|Q|)^{15d^4}]^{4^{d|E|}}$$

We are now ready for proving the following lemma.

▶ **Lemma 19.** *Let $G = (Q, \mathbf{A}, E)$ be a VASS such that $||\mathbf{A}|| \geq 1$, $I \subseteq \{1, \ldots, d\}$, and T be a recurrent set of edges. We consider a path π such that $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$. Let $m \geq 1$ satisfying:*

- *The number of occurrences in π of edges not in T is bounded by m, and*
- *$\mathbf{x}[i] + \Delta(\pi')[i] \leq m$ for every every prefix π' of π and for every $i \notin I$.*

Then m', I', T' defined as follows:

$$m' = m^4(1 + ||\mathbf{x}||)^2 d^2 (3\mu)^{15d^4}$$
$$(I', T') = \Omega(I, T)$$

Where $\mu = ||\mathbf{A}||\,|Q|$ satisfies:

- *The number of occurrences in π of edges not in T' is bounded by m', and*
- *$\mathbf{x}[i] + \Delta(\pi')[i] \leq m'$ for every every prefix π' of π and for every $i \notin I'$.*

Proof. Notice that it is sufficient to prove that the number of occurrences in π of edges not in T' is bounded by m', and $\mathbf{y}[i] \leq m'$ for every $i \notin I'$. In fact, the more general bound $\mathbf{x}[i] + \Delta(\pi')[i] \leq m'$ for every prefix π' of π and for every $i \notin I'$ can be obtained as a corollary.

Let k be the number of occurrences in π of edges not in T. It follows that π can be decomposed into: $\pi = \pi_0 t_1 \pi_1 \ldots t_k \pi_k$ where $\pi_0, \ldots, \pi_k$ are paths in T and $t_1, \ldots, t_k$ are not in T. We introduce the sequences $(p_j(\mathbf{x}_j))_{1\leq j\leq k}$ and $(q_j(\mathbf{y}_j))_{0\leq j\leq k}$ of configurations such

that $p_j(\mathbf{x}_j) \xrightarrow{\pi_j} q_j(\mathbf{y}_j)$ for every $0 \leq j \leq k$, $q_{j-1}(\mathbf{y}_{j-1}) \xrightarrow{t_j} p_j(\mathbf{x}_j)$ for every $1 \leq j \leq k$, and such that $p_0(\mathbf{x}_0) = p(\mathbf{x})$ and $q_k(\mathbf{y}_k) = q(\mathbf{y})$. Notice that $\mathbf{x}_j[i], \mathbf{y}_j[i] \leq m$ for every $i \notin I$ and for every $0 \leq j \leq k$ since $\mathbf{x}_j$ and $\mathbf{y}_j$ can be obtained trivially as vectors of the form $\mathbf{x} + \Delta(\pi')$ for some prefixes π' of π.

Let us observe that since π_j is a path in T and since T is recurrent, it follows that there exists $S_j \in \mathcal{SCC}(T)$ such that π_j is a path in S_j. We decompose the Parikh image α_j of π_j into $\alpha_j = \phi_j + \sum_{\ell \in L_j} \psi_{j,\ell}$ where ϕ_j is the Parikh image of a simple path in S_j, and $\psi_{j,\ell}$ is a Parikh image of an elementary cycle in S_j for every ℓ in a finite set L_j.

We introduce the set $\mathbf{Z}$ of displacements of elementary cycles. Notice that $||\mathbf{Z}|| \leq \mu$. We associate to each $S \in \mathcal{SCC}(T)$ and each $\mathbf{z} \in \mathbf{Z}$ the cardinal $n_{S,\mathbf{z}}$ of the set $\bigcup_{j|S_j=S}\{(j,\ell) \mid \ell \in L_j \wedge \Delta(\psi_{j,\ell}) = \mathbf{z}\}$. Observe that for every S, we have:

$$\begin{aligned}\sum_{\mathbf{z}\in\mathbf{Z}} n_{S,\mathbf{z}}\mathbf{z} &= \sum_{j|S_j=S} \sum_{\ell\in L_j} \Delta(\psi_{j,\ell}) \\ &= \sum_{j|S_j=S} \Delta(\alpha_j) - \Delta(\phi_j) \\ &= \sum_{j|S_j=S} (\mathbf{y}_j - \mathbf{x}_j - \Delta(\phi_j))\end{aligned}$$

It follows that for every $i \notin I$, we have:

$$|\sum_{\mathbf{z}\in\mathbf{Z}} n_{S,\mathbf{z}}\mathbf{z}[i]| \leq (k+1)(m + |Q|||\mathbf{A}||) \leq 2m(2m\mu) \leq 4m^2\mu(1 + ||\mathbf{x}||)$$

Moreover, we have:

$$\begin{aligned}\sum_{S\in\mathcal{SCC}(T)} \sum_{\mathbf{z}\in\mathbf{Z}} n_{S,\mathbf{z}}\mathbf{z} &= \sum_{j=0}^{k} (\mathbf{y}_j - \mathbf{x}_j - \Delta(\phi_j)) \\ &= \sum_{j=0}^{k} (\mathbf{y}_j - \mathbf{x}_j) + \sum_{j=1}^{k} \Delta(t_j) - \sum_{j=0}^{k} \Delta(\phi_j) - \sum_{j=1}^{k} \Delta(t_j) \\ &= \mathbf{y} - \mathbf{x} - \sum_{j=0}^{k} \Delta(\phi_j) - \sum_{j=1}^{k} \Delta(t_j)\end{aligned}$$

If follows that for every i, we have:

$$\begin{aligned}\sum_{S\in\mathcal{SCC}(T)} \sum_{\mathbf{z}\in\mathbf{Z}} n_{S,\mathbf{z}}\mathbf{z}[i] &\geq -(||\mathbf{x}|| + (k+1)(|Q|-1)||\mathbf{A}|| + k||\mathbf{A}||) \\ &\geq -3m\mu(1 + ||\mathbf{x}||) \geq -4m^2\mu(1 + ||\mathbf{x}||)\end{aligned}$$

Let us introduce $\delta = |\mathcal{SCC}(T)|(4m^2\mu(1 + ||\mathbf{x}||)d)^2(3\mu)^{9d^4}$. Notice that $\delta \leq m^4(1 + ||\mathbf{x}||)^2 d^2 (3\mu)^{12d^4}$. Lemma 20 shows that there exists a sequence of natural numbers $(m_{S,\mathbf{z}})_{S,\mathbf{z}}$ such that:

- $\sum_{\mathbf{z}\in\mathbf{Z}} m_{S,\mathbf{z}}\mathbf{z}[i] = \mathbf{0}$ for every S, and every $i \notin I$.
- $\sum_S \sum_{\mathbf{z}\in\mathbf{Z}} m_{S,\mathbf{z}}\mathbf{z} \geq \mathbf{0}$
- $n_{S,\mathbf{z}} > \delta$ implies $m_{S,\mathbf{z}} > 0$,
- $\sum_S \sum_{\mathbf{z}\in\mathbf{Z}} n_{S,\mathbf{z}}\mathbf{z}[i] > \delta$ implies $\sum_S \sum_{\mathbf{z}\in\mathbf{Z}} m_{S,\mathbf{z}}\mathbf{z}[i] > 0$.

It follows that:

- if $n_{S,\mathbf{z}} > \delta$ then every simple cycle of S with a displacement equal to $\mathbf{z}$ is a simple cycle of T', and

- if $\mathbf{y}[i] > ||\mathbf{x}|| + \delta + m + (k+1)(|Q|-1)||\mathbf{A}|| + k||\mathbf{A}||$ then $i \in I'$.

It follows that $\mathbf{y}[i] \leq ||\mathbf{x}|| + \delta + 3m\mu \leq 3\delta \leq m'$ for every $i \notin I'$. Moreover, it follows that for every S, the number of occurrences of cycles $\psi_{j,\ell}$ with j such that $S_j = S$, and $\ell \in L_j$ such that $\mathrm{dom}(\psi_{j,\ell}) \not\subseteq T'$ is bounded by $|\mathbf{Z}|\delta \leq m^4(1+||\mathbf{x}||)^2 d^2 (3\mu)^{13d^4}$. It follows that the number of occurrences of edges not in T' in π is bounded by $k+|\mathcal{SCC}(T)|.|Q|.m^4(1+||\mathbf{x}||)^2 d^2 (3\mu)^{13d^4} \leq m^4(1+||\mathbf{x}||)^2 d^2 (3\mu)^{15d^4} = m'$. ◀

The proof of the previous lemma was based on the following one, a kind of "small solution" theorem [16].

▶ **Lemma 20.** *Let $(n_{s,\mathbf{z}})_{s,\mathbf{z}}$ be a sequence of natural numbers indexes by s in a non-empty finite set S, and by $\mathbf{z}$ in a finite subset $\mathbf{Z} \subseteq \{-\mu, \ldots, \mu\}^d$ for some $\mu \geq 1$. Let $I \subseteq \{1, \ldots, d\}$ and $m \geq 1$ such that:*

- $|\sum_{\mathbf{z}\in\mathbf{Z}} n_{s,\mathbf{z}}\mathbf{z}[i]| \leq m$ *for every* $s \in S$ *and for every* $i \notin I$, *and*
- *The vector* $\mathbf{v}$ *defined as* $\sum_{s\in S}\sum_{\mathbf{z}\in\mathbf{Z}} n_{s,\mathbf{z}}\mathbf{z}$ *satisfies* $\mathbf{v}[i] \geq -m$ *for every* $i \in \{1, \ldots, d\}$.

There exists a sequence $(m_{s,\mathbf{z}})_{\mathbf{z}\in\mathbf{Z},s\in S}$ of natural numbers such that:

- $\sum_{\mathbf{z}\in\mathbf{Z}} m_{s,\mathbf{z}}\mathbf{z}[i] = 0$ *for every* $s \in S$ *and for every* $i \notin I$, *and*
- *The vector* $\mathbf{w}$ *defined as* $\sum_{s\in S}\sum_{\mathbf{z}\in\mathbf{Z}} m_{s,\mathbf{z}}\mathbf{z}$ *satisfies* $\mathbf{w} \geq \mathbf{0}$

and such that $\delta = |S|(md)^2(3\mu)^{9d^4}$ satisfies:

- *If* $n_{s,\mathbf{z}} > \delta$ *then* $m_{s,\mathbf{z}} > 0$, *and*
- *If* $\mathbf{v}[i] > \delta$ *then* $\mathbf{w}[i] > 0$.

Proof. The proof is based on an application of a "small solution" theorem of Pottier [16] on each $s \in S$, and then, on the resulting solutions we apply again the "small solution" theorem for extracting solutions satisfying $\mathbf{w} \geq \mathbf{0}$. ◀

Let us consider $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$ and let $(I_n, T_n) = \Omega^n(\{1, \ldots, d\}, E)$. We introduce for every n the minimal number $m_n \geq 1$ such that the number of occurrences of edges in π that are not in T_n is bounded by m_n, and $\mathbf{x}[i] + \Delta(\pi')[i] \leq m_n$ for every $i \notin I_n$ and for every prefix π' of π. Notice that $m_0 = 1$, and Lemma 19 shows that for every $n \geq 0$:

$$m_{n+1} \leq m_n^4(1+||\mathbf{x}||)^2 d^2 (3\mu)^{15d^4}$$

Let us introduce the sequence $(s_n)_{n\geq 0}$ defined by $s_0 = 1$, and the induction for every $n \in \mathbb{N}$:

$$s_{n+1} = s_n^4.(1+||\mathbf{x}||)^2 d^2 (3\mu)^{15d^4}$$

Observe that $m_n \leq s_n$ for every n. Moreover, we have:

$$s_n = [(1+||\mathbf{x}||)^2.d^2(3\mu)^{15d^4}]^{4^n-1}$$

Since $\Gamma(\{1, \ldots, d\}, E) = \Omega^\infty(\{1, \ldots, d\}, E) = \Omega^{d|E|}(\{1, \ldots, d\}, E)$, we have proved Theorem 18.

7 Applications

Theorem 18 shows that a strongly connected VASS without iterable indexe is polynomial. Combined with Lemma 10, we deduce the following characterization.

▶ **Theorem 21.** *A strongly connected VASS is polynomial if, and only if, its set of iterable indexes is empty.*

As a direct consequence of Theorems 21 and 17 and the following Lemma 22, we get the following Theorem 23. Note that the restriction of a VASS $G = (Q, \mathbf{A}, E)$ to a subset $T \subseteq E$ of edges is defined as the VASS $G|_T = (Q, \mathbf{A}, T)$.

▶ **Lemma 22.** *A VASS is polynomial if, and only if, its restriction to every SCC is polynomial.*

Proof. The proof is given in a technical report. ◀

▶ **Theorem 23.** *We can decide in polynomial-time if a VASS is polynomial.*

Moreover, since Theorem 18 shows that reachable configurations are bounded exponentially in space, we derive the following result.

▶ **Theorem 24.** *The reachability problem for polynomial VASS is exponential-space complete.*

Proof. Theorem 18 shows that reachable configurations are bounded exponentially in space. It follows that the reachability problem is decidable in exponential space. We have already observed the lower bound in Section 2.2. ◀

8 Conclusion

In this paper we introduced the class of polynomial VASS and showed that the membership problem of a VASS in that class is decidable in polynomial time. Moreover, we proved that the reachability problem for polynomial VASS is exponential-space complete. Our characterization of polynomial VASS is based on the notion of iteration schemes. Intuitively, whereas a cycle of a VASS with a non-negative displacement can be iterated an arbitrarily number of times to obtain arbitrarily large values on indexes that are strictly increased by the cycle, iteration schemes can be iterated an exponential number of times and provide a way to increase by an exponential number every index that is increased by the iteration scheme. As a future work, we are interested in using iteration schemes rather than iterable cycles in the KLM algorithm to hopefully obtain better complexity upper bound for the reachability problem for general vector addition systems.

References

1. I. Borosh and L. B. Treybig. Bounds on positive integral solutions of linear diophantine equations. *Proc. Amer. Math. Soc.*, 55(2):299–304, 1976. doi:10.1090/S0002-9939-1976-0396605-3.
2. E. Cohen and N. Megiddo. Strongly polynomial-time and NC algorithms for detecting cycles in dynamic graphs. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, pages 523–534, New York, NY, USA, 1989. ACM. doi:10.1145/73007.73057.
3. Javier Esparza. Decidability and complexity of Petri net problems — an introduction. In *Advances in Petri Nets '98*, volume 1491 of *LNCS*, pages 374–428. Springer, 1998. doi:10.1007/3-540-65306-6_20.
4. Javier Esparza and Mogens Nielsen. Decidability issues for Petri nets - a survey. *Bulletin of the EATCS*, 52:244–262, 1994.
5. Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson's lemma. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 269–278. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.39.

6 John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979. doi:10.1016/0304-3975(79)90041-0.

7 S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 267–281. ACM, 1982. doi:10.1145/800070.802201.

8 S. Rao Kosaraju and Gregory F. Sullivan. Detecting cycles in dynamic graphs in polynomial time (preliminary version). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 398–406. ACM, 1988. doi:10.1145/62212.62251.

9 Jean-Luc Lambert. A structure to decide reachability in Petri nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992. doi:10.1016/0304-3975(92)90173-D.

10 Jérôme Leroux. Vector addition systems reachability problem (A simpler solution). In *Turing-100*, volume 10 of *EPiC Series in Computing*, pages 214–228. EasyChair, 2012.

11 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 56–67. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.16.

12 R. J. Lipton. The reachability problem requires exponential space. Technical Report 63, Department of Computer Science, Yale University, 1976.

13 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*, pages 238–246. ACM, 1981. doi:10.1145/800076.802477.

14 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984. doi:10.1137/0213029.

15 Ernst W. Mayr and Albert R. Meyer. The complexity of the finite containment problem for Petri nets. *J. ACM*, 28(3):561–576, 1981. doi:10.1145/322261.322271.

16 Loic Pottier. Minimal solutions of linear diophantine systems: Bounds and algorithms. In *Proceedings of the 4th International Conference on Rewriting Techniques and Applications*, RTA '91, pages 162–173, London, UK, UK, 1991. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=647192.720494.

17 M. Praveen and Kamal Lodaya. Analyzing reachability for some Petri nets with fast growing markings. *Electr. Notes Theor. Comput. Sci.*, 223:215–237, 2008. doi:10.1016/j.entcs.2008.12.041.

18 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

19 Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, New York, 1987.

Reducing CMSO Model Checking to Highly Connected Graphs

Daniel Lokshtanov[1]
University of Bergen, Norway
daniello@ii.uib.no

M. S. Ramanujan[2]
University of Warwick, United Kingdom
R.Maadapuzhi-Sridharan@warwick.ac.uk

Saket Saurabh[3]
The Institute of Mathematical Sciences, HBNI, Chennai, India
saket@imsc.res.in

Meirav Zehavi
Ben-Gurion University, Israel
Zehavimeirav@gmail.com

Abstract

Given a Counting Monadic Second Order (CMSO) sentence ψ, the CMSO[ψ] problem is defined as follows. The input to CMSO[ψ] is a graph G, and the objective is to determine whether $G \models \psi$. Our main theorem states that for every CMSO sentence ψ, if CMSO[ψ] is solvable in polynomial time on "globally highly connected graphs", then CMSO[ψ] is solvable in polynomial time (on general graphs). We demonstrate the utility of our theorem in the design of parameterized algorithms. Specifically we show that technical problem-specific ingredients of a powerful method for designing parameterized algorithms, recursive understanding, can be replaced by a black-box invocation of our main theorem. We also show that our theorem can be easily deployed to show fixed parameterized tractability of a wide range of problems, where the input is a graph G and the task is to find a connected induced subgraph of G such that "few" vertices in this subgraph have neighbors outside the subgraph, and additionally the subgraph has a CMSO-definable property.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Fixed Parameter Tractability Model Checking Recursive Understanding

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.135

Related Version A full version of the paper is available at `https://arxiv.org/abs/1802.01453`.

1 Introduction

Algorithmic meta-theorems are general algorithmic results applicable to a whole range of problems. Many prominent algorithmic meta-theorems are about model checking; such theorems state that for certain kinds of logic L, and all classes $\mathcal{C}$ that have a certain property,

[1] Supported by *Pareto-Optimal Parameterized Algorithms*, ERC Starting Grant 715744
[2] Supported by *BeHard*, Bergen Research Foundation and *X-Tract*, Austrian Science Fund (FWF, project P26696)
[3] Supported by *Parameterized Approximation*, ERC Starting Grant 306992.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 135; pp. 135:1–135:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

there is an algorithm that takes as input a formula $\phi \in L$ and a structure $S \in \mathcal{C}$ and efficiently determines whether $S \models \phi$. Results in this direction include the seminal theorem of Courcelle [8, 7, 9] for model checking Monadic Second Order Logic (MSO) on graphs of bounded treewidth (see also [1, 2, 4, 10, 14]), as well as a large body of work on model checking first-order (FO) logic [5, 12, 16, 18, 19, 21, 20, 23, 28].

Another kind of algorithmic meta-theorems *reduce* the task of designing one type of algorithm for a problem, to one of designing a different kind of algorithm for the same problem. The hope is, of course, that the second type of algorithms are significantly easier to design than the first. A prototype example of such results is *Bidimensionality* [13], which reduces the design of sub-exponential time parameterized algorithms for a problem on planar (or H-minor free) graphs, to the design of single exponential time algorithms for the same problem when parameterized by the treewidth of the input graph.

In this paper we prove a result of the second type for model checking Counting Monadic Second Order Logic (CMSO), an extension of MSO with atomic sentences for determining the cardinality of vertex and edge sets modulo any (fixed) integer. For every CMSO sentence ψ define the CMSO[ψ] problem as follows. The input is a graph G on n vertices, and the task is to determine whether $G \models \psi$.

Our main result states that for every CMSO sentence ψ, if there is a $\mathcal{O}(n^d)$ time algorithm ($d > 4$) for CMSO[ψ] for the special case when the input graph is required to be "highly connected everywhere", then there is a $\mathcal{O}(n^d)$ time algorithm for CMSO[ψ] without any restrictions. In other words, our main theorem reduces CMSO model checking to model checking the same formula on graphs which are "highly connected everywhere".

In order to complete the description of our main result we need to define what we mean by "highly connected everywhere". For two integers s and c, we say that a graph G is (s, c)-*unbreakable* if there does not exist a partition of the vertex set into three sets X, C, and Y such that

- C is a separator: there are no edges from X to Y,
- C is small: $|C| \leq c$, and
- X and Y are large: $|X|, |Y| \geq s$.

For example, the set of $(1, c)$-unbreakable graphs contains precisely the $(c + 1)$-connected graphs, i.e. the connected graphs for which removing any set of at most c vertices leaves the graph connected. We can now state our main result:

▶ **Theorem 1.** *Let ψ be a CMSO sentence. For all $c \in \mathbb{N}$, there exists $s \in \mathbb{N}$ such that if there exists an algorithm that solves* CMSO[ψ] *on (s, c)-unbreakable graphs in time $\mathcal{O}(n^d)$ for some $d > 4$, then there exists an algorithm that solves* CMSO[ψ] *on general graphs in time $\mathcal{O}(n^d)$.*

For Theorem 1 to be useful, there must exist problems that can be formulated in CMSO, for which it is easier to design algorithms for the special case when the input graphs are unbreakable, than it is to design algorithms that work on general graphs. Such problems can be found in abundance in *parameterized complexity*. Indeed, the *recursive understanding* technique, which has been used to solve several problems [6, 22, 24, 25, 27, 26] in parameterized complexity, is based precisely on the observation that for many graph problems it is much easier to design algorithms if the input graph can be assumed to be unbreakable.

Designing algorithms using the recursive understanding technique typically involves a technical and involved argument akin to doing dynamic programming on graphs of bounded treewidth (see Chitnis et al. [6] for an exposition). These arguments reduce the original problem on general graphs to a generalized version of the problem on (s, c)-unbreakable

graphs, for appropriate values of s and c. Then an algorithm is designed for this generalized problem on (s, c)-unbreakable graphs, yielding an algorithm for the original problem.

For all applications of the recursive understanding technique known to the authors [6, 22, 24, 25, 27, 26], the problem in question (in which recursive understanding has been applied) can be formulated as a CMSO model checking problem, and therefore, the rather cumbersome application of recursive understanding can be completely replaced by a black box invocation of Theorem 1. Using Theorem 1 in place of recursive understanding has the additional advantage that it reduces problems on general graphs to *the same* problem on unbreakable graphs, facilitating also the last step of designing an algorithm on unbreakable graphs.

As an example of the power of Theorem 1 we use it to give a fixed parameter tractable (FPT) algorithm for the Vertex Multiway Cut Uncut problem. Details can be found in the full version of the paper on arXiv.org. Here, we are given a graph G together with a set of terminals $T \subseteq V(G)$, an equivalence relation $\mathcal{R}$ on the set T, and an integer k, and the objective is to test whether there exists a set $S \subseteq V(G) \setminus T$ of at most k vertices such that for any $u, v \in T$, the vertices u and v belong to the same connected component of $G \setminus S$ if and only if $(u, v) \in \mathcal{R}$. Since finding the desired set S satisfying the above property can be formulated in CMSO, we are able to completely sidestep the necessity to define a technically involved annotated version of our problem, and furthermore, we need only focus on the base case where the graph is unbreakable. To solve the base case, a simple procedure that is based on the enumeration of connected sets with small neighborhood is sufficient. For classification purposes, our approach is significantly simpler than the problem-specific algorithm in [6]. Finally, we show how Theorem 1 can be effortlessly deployed to show fixed parameterized tractability of a wide range of problems, where the input is a graph G and the task is to find a connected induced subgraph of G of bounded treewidth such that "few" vertices outside this subgraph have neighbors inside the subgraph, and additionally the subgraph has a CMSO-definable property.

Our techniques. The proof of Theorem 1 is based heavily on the idea of graph replacement, which dates back to the work of Fellows and Langston [17]. We combine this idea with Courcelle's theorem [8, 7, 9], which states that every CMSO-definable property σ has finite state on a bounded-size separation/boundary. In other words, for any CMSO-definable property σ and fixed $t \in \mathbb{N}$, there is an equivalence relation defined on the set of all t-boundaried graphs (graphs with a set of at most t distinguished vertices) with a finite number, say ζ (where ζ depends only on σ and t) of equivalence classes such that if we replace any t-boundaried subgraph H of the given graph G with another t-boundaried graph, say H', from the same equivalence class to obtain a graph G', then G has the property σ if and only if G' has the property σ. In case of (s, c)-unbreakable graphs, $t = 2c$. Let $R_1, \ldots, R_\zeta$ denote a set containing one " minimal" $2c$-boundaried graph from each equivalence class (for the fixed CMSO-definable property σ). Let r denote the size of the largest among these minimal representatives.

The main technical content of our paper is in the description of an algorithm for a generalization of our question. To be precise, we will describe how one can, given a $2c$-boundaried graph G, locate the precise equivalence class in which G is contained and how one could compute the corresponding smallest representative from the set $\{R_1, \ldots, R_\zeta\}$. We refer to this task as "understanding" G.

In order to achieve our objective, we first give an algorithm $\mathcal{A}$ that allows one to understand $2c$-boundaried $(s - r, c)$-unbreakable graphs (for a choice of s which is sufficiently

large compared to r and c). This algorithm is built upon the following observation. The equivalence class of any $2c$-boundaried graph G is determined exactly by the subset of $\{G \oplus R_1, G \oplus R_2, \ldots, G \oplus R_\zeta\}$ on which σ evaluates to true. Here, the graph $G \oplus R_i$ is the graph obtained by taking the disjoint union of the graphs G and R_i and then identifying the vertices of the boundaries of these graphs with the same label. Since s is chosen to be sufficiently large compared to c and r, it follows that for every $i \in \{1, \ldots, \zeta\}$, the graph $G \oplus R_i$ is (s, c)-unbreakable and we can use the assumed algorithm for CMSO$[\psi]$ on (s, c)-unbreakable graphs to design an algorithm that *understands* $2c$-boundaried $(s - r, c)$-unbreakable graphs. This constitutes the 'base case' of our main algorithm.

In order to understand a general ($(s - r, c)$-breakable) $2c$-boundaried graph, we use known algorithms from [6] to compute a partition of the vertex set of G into X, C, and Y such that C is a separator, $|C| \leq c$ and $|X|, |Y| \geq \frac{s-r}{2^c}$. Let $G_1 = G[X \cup C]$ and let $G = G[Y \cup C]$. Without loss of generality, we may assume that at most half the vertices in the boundary of G lie in $X \cup C$. Consequently, the graph G_1 is a $2c$-boundaried graphs where the boundary vertices are the vertices in C along with the boundary vertices of G contained in $X \cup C$. We then recursively understand the strictly smaller $2c$-boundaried graph G_1 to find its representative $\hat{R} \in \{R_1, \ldots, R_\zeta\}$. Since the evaluation of σ on G is the same as the evaluation of σ on $G_2 \oplus \hat{R}$ (where the gluing happens along C), we only need to understand the $2c$-boundaried graph $G_2 \oplus \hat{R}$ (where the boundary is carefully defined from that of G and $\hat{R}$) and we do this by recursively executing the "understand" algorithm on this graph.

At this point we also need to remark on two drawbacks of Theorem 1. The first is that Theorem 1 is *non-constructive*. Given an algorithm for CMSO$[\psi]$ on (s, c)-unbreakable graphs, Theorem 1 allows us to infer the existence of an algorithm for CMSO$[\psi]$ on general graphs, but it does not provide us with the actual algorithm. This is due to the subroutine $\mathcal{S}$ requiring a representative $2c$-boundaried subgraph for each equivalence class, to be part of its 'source code'. Thus, the parameterized algorithms obtained using Theorem 1 are *non-uniform* (see Section 4), as opposed to the algorithms obtained by recursive understanding.

The second drawback is that Theorem 1 incurs a gargantuan constant factor overhead in the running time, where this factor depends on the formula ψ and the cut size c. We leave removing these two drawbacks as intriguing open problems.

2 Preliminaries

In order to present a rigorous proof of our lemmas in a way that is consistent with existing notation used in related work, we follow the notation from the paper [3].

Graphs and treewidth. Throughout this paper, we use the term "graph" to refer to a multigraph rather than only a simple graph. Given a graph G, we let $V(G)$ and $E(G)$ denote the vertex and edge sets of G, respectively. When G is clear from the context, we denote $n = |V(G)|$ and $m = |E(G)|$. Given two subsets of $V(G)$, A and B, we let $E(A, B)$ denote the set of edges of G with one endpoint in A and the other endpoint in B. Given $U \subseteq V(G)$, we let $G[U]$ denote the subgraph of G induced by U, and we let $N(U)$ and $N[U]$ denote the open and closed neighborhoods of U, respectively. Moreover, we denote $G \setminus U = G[V(G) \setminus U]$. Given $v \in V(G)$, we denote $N(v) = N(\{v\})$ and $N[v] = N[\{v\}]$. Given $E \subseteq E(G)$, we denote $G \setminus E = (V(G), E(G) \setminus E)$. Moreover, we let $V[E]$ denote the set of every vertex in $V(G)$ that is incident to at least one edge in E, and we define $G[E] = (V[E], E)$. A graph G is a *cluster graph* if there exists a partition $(V_1, V_2, \ldots, V_r)$ of $V(G)$ for some $r \in \mathbb{N}_0$ of $V(G)$ such that for all $i \in [r]$, $G[V_i]$ is a clique, and for all $j \in [r] \setminus \{i\}$, $E(V_i, V_j) = \emptyset$.

▶ **Definition 2.** A *tree decomposition* of a graph G is a pair (T, β) of a tree T and $\beta : V(T) \to 2^{V(G)}$, such that (a) $\bigcup_{t \in V(T)} \beta(t) = V(G)$, and (b) for any edge $e \in E(G)$, there exists a node $t \in V(T)$ such that both endpoints of e belong to $\beta(t)$, and (c) for any vertex $v \in V(G)$, the subgraph of T induced by the set $T_v = \{t \in V(T) : v \in \beta(t)\}$ is a tree.

The *width* of (T, β) is $\max_{v \in V(T)}\{|\beta(v)|\} - 1$. The *treewidth* of G is the minimum width of a tree decomposition of G.

Unbreakability. To formally introduce the notion of unbreakability, we rely on the definition of a separation:

▶ **Definition 3** (Separation)**.** A pair (X, Y) where $X \cup Y = V(G)$ is a *separation* if $E(X \setminus Y, Y \setminus X) = \emptyset$. The order of (X, Y) is $|X \cap Y|$.

Roughly speaking, a graph is breakable if it is possible to "break" it into two large parts by removing only a small number of vertices. Formally,

▶ **Definition 4** ((s, c)-Unbreakable graph)**.** Let G be a graph. If there exists a separation (X, Y) of order at most c such that $|X \setminus Y| > s$ and $|Y \setminus X| > s$, called an (s, c)-*witnessing separation*, then G is (s, c)-*breakable*. Otherwise, G is (s, c)-*unbreakable*.

The following lemma implies that it is possible to determine (approximately) whether a graph is unbreakable or not, and lemmata similar to it can be found in [6].

▶ **Lemma 5.** *There exists an algorithm,* Break-ALG*, that given $s, c \in \mathbb{N}$ and a graph G, in time $2^{\mathcal{O}(c \log(s+c))} \cdot n^3 \log n$ either returns an $(\frac{s}{2^c}, c)$-witnessing separation or correctly concludes that G is (s, c)-unbreakable.*

Boundaried Graphs. Roughly speaking, a boundaried graph is a graph where some vertices are labeled. Formally,

▶ **Definition 6** (Boundaried graph)**.** A *boundaried graph* is a graph G with a set $\delta(G) \subseteq V(G)$ of distinguished vertices called *boundary vertices*, and an injective labeling $\lambda_G : \delta(G) \to \mathbb{N}$. The set $\delta(G)$ is the *boundary* of G, and the *label set* of G is $\Lambda(G) = \{\lambda_G(v) \mid v \in \delta(G)\}$.

We remark that we also extend the definition of (s, c)-(un)breakability from graphs, to boundaried graphs in the natural way. That is, we ignore the boundary vertices when considering the existence of an (s, c)-witnessing separation. For ease of presentation, we sometimes abuse notation and treat equally-labeled vertices of different boundaried graphs, as well as the vertex that is the result of the identification of two such vertices, as the same vertex. Given a finite set $I \subseteq \mathbb{N}$, $\mathcal{F}_I$ denotes the class of all boundaried graphs whose label set is I, and $\mathcal{F}_{\subseteq I} = \bigcup_{I' \subseteq I} \mathcal{F}_{I'}$. A boundaried graph in $\mathcal{F}_{\subseteq [t]}$ is called a t-*boundaried* graph. Finally, $\mathcal{F}$ denotes the class of all boundaried graphs. The main operation employed to unite two boundaried graphs is the one that glues their boundary vertices together. Formally,

▶ **Definition 7** (Gluing by $\oplus$)**.** Let G_1 and G_2 be two boundaried graphs. Then, $G_1 \oplus G_2$ is the (not-boundaried) graph obtained from the disjoint union of G_1 and G_2 by identifying equally-labeled vertices in $\delta(G_1)$ and $\delta(G_2)$.[4]

[4] Each edge in G_1 (or G_2) whose endpoints are boundaried vertices in G_1 (or G_2) is preserved as a unique edge in $G_1 \oplus G_2$.

Structures. We first define the extension of graphs to *structures* in the context of our paper.

▶ **Definition 8** (Structure)**.** A *structure* α is a tuple whose first element is a graph, denoted by G_α, and each of the remaining elements is a subset of $V(G_\alpha)$, a subset of $E(G_\alpha)$, a vertex in $V(G_\alpha)$ or an edge in $E(G_\alpha)$. The number of elements in the tuple is the *arity* of the structure.

Given a structure α of arity p and an integer $i \in [p]$, we let $\alpha[i]$ denote the i'th element of α. Note that $\alpha[1] = G_\alpha$. By *appending* a subset S of $V(G_\alpha)$ (or $E(G_\alpha)$) to a structure α of arity p, we produce a new structure, denoted by $\alpha' = \alpha \diamond S$, of arity $p+1$ with the first p elements of α' being the elements of α and $\alpha'[p+1] = S$. For example, consider the structure $\alpha = (G_\alpha, S, e)$ of arity 3, where $S \subseteq V(G_\alpha)$ and $e \in E(G_\alpha)$. Let S' be some subset of $V(G_\alpha)$. Then, appending S' to α results in the structure $\alpha' = \alpha \diamond S' = (G_\alpha, S, e, S')$.

Next, we define the notions of a *type* of a structure and a *property* of structures.

▶ **Definition 9** (Type)**.** Let α be a structure of arity p. The *type* of α is a tuple of arity p, denoted by $\mathbf{type}(\alpha)$, where the first element, $\mathbf{type}(\alpha)[1]$, is graph, and for every $i \in \{2, 3, \ldots, p\}$, $\mathbf{type}(\alpha)[i]$ is vertex if $\alpha[i] \in V(G_\alpha)$, edge if $\alpha[i] \in E(G_\alpha)$, vertex set if $\alpha[i] \subseteq V(G_\alpha)$, and edge set otherwise.[5]

▶ **Definition 10** (Property)**.** A *property* is a function σ from the set of all structures to {true, false}.

Finally, we extend the notion of unbreakability to structures.

▶ **Definition 11** ((s,c)-Unbreakable structure)**.** Let α be a structure. If G_α is an (s,c)-unbreakable graph, then we say that α is an (s,c)-*unbreakable* structure, and otherwise we say that α is an (s,c)-*breakable* structure.

Counting Monadic Second Order Logic. The syntax of Monadic Second Order Logic (MSO) of graphs includes the logical connectives $\vee$, $\wedge$, $\neg$, $\Leftrightarrow$, $\Rightarrow$, variables for vertices, edges, sets of vertices and sets of edges, the quantifiers $\forall$ and $\exists$, which can be applied to these variables, and five binary relations: (a) $u \in U$, where u is a vertex variable and U is a vertex set variable; (b) $d \in D$, where d is an edge variable and D is an edge set variable; (c) $\mathbf{inc}(d, u)$, where d is an edge variable, u is a vertex variable, and the interpretation is that the edge d is incident to u; (d) $\mathbf{adj}(u, v)$, where u and v are vertex variables, and the interpretation is that u and v are adjacent; (e) equality of variables representing vertices, edges, vertex sets and edge sets.

Counting Monadic Second Order Logic (CMSO) extends MSO by including atomic sentences testing whether the cardinality of a set is equal to q modulo r, where q and r are integers such that $0 \leq q < r$ and $r \geq 2$. That is, CMSO is MSO with the following atomic sentence: $\mathbf{card}_{q,r}(S) = \mathbf{true}$ if and only if $|S| \equiv q \pmod r$, where S is a set. We refer to [2, 8, 9] for a detailed introduction to CMSO.

Evaluation. To evaluate a CMSO-formula ψ on a structure α, we instantiate the free variables of ψ by the elements of α. In order to determine which of the free variables of ψ are instantiated by which of the elements of α, we introduce the following conventions. First, each free variable x of a CMSO-formula ψ is associated with a rank, $r_x \in \mathbb{N} \setminus \{1\}$. Thus, a

[5] Note that we distinguish between a set containing a single vertex (or edge) and a single vertex (or edge).

CMSO-formula ψ can be viewed as a string accompanied by a tuple of integers, where the tuple consists of one integer r_x for each free variable x of ψ.

Given a structure α and a CMSO-formula ψ, we say that $\mathbf{type}(\alpha)$ *matches* ψ if **(i)** the arity of α is at least $\max r_x$, where the maximum is taken over each free variable x of ψ, and **(ii)** for each free variable x of ψ, $\mathbf{type}(\alpha)[r_x]$ is compatible with the type of x. For example, if x is a vertex set variable, then $\mathbf{type}(\alpha)[r_x] =$ vertex set. Finally, we say that α *matches* ψ if $\mathbf{type}(\alpha)$ matches ψ. Given a free variable x of a CMSO sentence ψ and a structure α that matches ψ, the element corresponding to x in α is $\alpha[r_x]$.

▶ **Definition 12.** [Property σ_ψ] Given a CMSO-formula ψ, the property σ_ψ is defined as follows. Given a structure α, if α does not match ψ, then $\sigma_\psi(\alpha)$ equals false, and otherwise $\sigma_\psi(\alpha)$ equals the result of the evaluation of ψ where each free variable x of ψ is instantiated by $\alpha[r_x]$.

Note that some elements of α may not correspond to any variable of ψ. However, ψ may still be evaluated on the structure α—in this case, the evaluation of ψ does not depend on all the elements of the structure. If the arity of α is 1, then we use $\sigma_\psi(G_\alpha)$ to denote $\sigma_\psi(\alpha)$.

▶ **Definition 13.** [CMSO-definable property] A property σ is *CMSO-definable* if there exists a CMSO-formula ψ such that $\sigma = \sigma_\psi$. In this case, we say that ψ *defines* σ.

Structures. The notion of a *boundaried structure* is an extension of the notion of a *boundaried graph* and is defined as follows.

▶ **Definition 14** (Boundaried structure)**.** A *boundaried structure* is a tuple whose first element is a boundaried graph G, denoted by G_α, and each of the remaining elements is a subset of $V(G)$, a subset of $E(G)$, a vertex in $V(G)$, an edge in $E(G)$, or the symbol $\star$. The number of elements in the tuple is the *arity* of the boundaried structure.

Given a boundaried structure α of arity p and an integer $i \in [p]$, we let $\alpha[i]$ denote the i'th element of α. We remark that we extend the definition of (s, c)-(un)breakability of structures, to boundaried structures.

▶ **Definition 15** (Type)**.** Let α be a boundaried structure of arity p. The *type* of α is a tuple of arity p, denoted by $\mathbf{type}(\alpha)$, where the first element, $\mathbf{type}(\alpha)[1]$, is boundaried graph, and for every $i \in \{2, 3, \ldots, p\}$, $\mathbf{type}(\alpha)[i]$ is vertex if $\alpha[i] \in V(G_\alpha)$, edge if $\alpha[i] \in E(G_\alpha)$, vertex set if $\alpha[i] \subseteq V(G_\alpha)$, edge set if $\alpha[i] \subseteq E(G_\alpha)$ and $\star$ otherwise.

Now, given a boundaried structure and a CMSO-formula ψ, we say that $\mathbf{type}(\alpha)$ *matches* ψ if **(i)** the arity of α is at least $\max r_x$, where the maximum is taken over each free variable x of ψ, and **(ii)** for each free variable x of ψ, $\mathbf{type}(\alpha)[r_x]$ is compatible with the type of x. Moreover, we say that α matches ψ if $\mathbf{type}(\alpha)$ *matches* ψ.

Given $p \in \mathbb{N}$, $\mathcal{A}^p$ denotes the class of all boundaried structures of arity p, and given a finite set $I \subseteq \mathbb{N}$, $\mathcal{A}^p_I$ ($\mathcal{A}^p_{\subseteq I}$) denotes the class of all boundaried structures of arity p whose boundaried graph belongs to $\mathcal{F}_I$ (resp. $\mathcal{F}_{\subseteq I}$). A boundaried structure in $\mathcal{A}^p_{\subseteq [t]}$ is called a *t-boundaried structure*. Finally, we let $\mathcal{A}$ denote the class of all boundaried structures.

▶ **Definition 16** (Compatiblity)**.** Two boundaried structures α and β are *compatible* (notationally, $\alpha \sim_c \beta$) if the following conditions are satisfied.

- α and β have the same arity p.
- For every $i \in [p]$:
  - $\mathbf{type}(\alpha)[i] = \mathbf{type}(\beta)[i] \neq \star$, or

  - $\mathbf{type}(\alpha)[i] \in \{\mathsf{vertex}, \mathsf{edge}\}$ and $\mathbf{type}(\beta)[i] = \star$, or
  - $\mathbf{type}(\beta)[i] \in \{\mathsf{vertex}, \mathsf{edge}\}$ and $\mathbf{type}(\alpha)[i] = \star$.
- For every $i \in [p]$ such that both $\alpha[i]$ and $\beta[i]$ are vertices: $\alpha[i] \in \delta(G_\alpha)$, $\beta[i] \in \delta(G_\beta)$ and $\lambda_{G_\alpha}(\alpha[i]) = \lambda_{G_\beta}(\beta[i])$.
- For every $i \in [p]$ such that both $\alpha[i]$ and $\beta[i]$ are edges: $\alpha[i] \in E(G_\alpha[\delta(G_\alpha)])$, $\beta[i] \in E(G_\beta[\delta(G_\beta)])$ and $\{\lambda_{G_\alpha}(x_{\alpha[i]}), \lambda_{G_\alpha}(y_{\alpha[i]})\} = \{\lambda_{G_\beta}(x_{\beta[i]}), \lambda_{G_\beta}(y_{\beta[i]})\}$, where $\alpha[i] = (x_{\alpha[i]}, y_{\alpha[i]})$ and $\beta[i] = (x_{\beta[i]}, y_{\beta[i]})$. That is, x_j and y_j are the endpoints of the edge $j \in \{\alpha[i], \beta[i]\}$.

▶ **Definition 17** (Gluing by $\oplus$)**.** Given two compatible boundaried structures α and β of arity p, the operation $\alpha \oplus \beta$ is defined as follows.

- $\alpha \oplus \beta$ is a structure γ of arity p.
- $G_\gamma = G_\alpha \oplus G_\beta$.
- For every $i \in [p]$:
  - if $\alpha[i]$ and $\beta[i]$ are sets, $\gamma[i] = \alpha[i] \cup \beta[i]$;
  - if $\alpha[i]$ and $\beta[i]$ are vertices/edges, $\gamma[i] = \alpha[i] = \beta[i]$;
  - if $\alpha[i] = \star$, $\gamma[i] = \beta[i]$;
  - if $\beta[i] = \star$, $\gamma[i] = \alpha[i]$.

State. This subsection states a variant of the classical Courcelle's Theorem [8, 7, 9] (see also [10]), which is a central component in the proof of our main result. To this end, we first define the *compatibility equivalence relation* $\equiv_c$ on boundaried structures as follows. We say that $\alpha \equiv_c \beta$ if $\Lambda(G_\alpha) = \Lambda(G_\beta)$ and for every boundaried structure γ, $\alpha \sim_c \gamma \iff \beta \sim_c \gamma$. Now, we define the *canonical equivalence relation* $\equiv_\sigma$ on boundaried structures.

▶ **Definition 18** (Canonical equivalence)**.** Given a property σ of structures, the *canonical equivalence relation* $\equiv_\sigma$ on boundaried structures is defined as follows. For two boundaried structures α and β, we say that $\alpha \equiv_\sigma \beta$ if **(i)** $\alpha \equiv_c \beta$, and **(ii)** for all boundaried structures γ compatible with α (and thus also with β), we have $\sigma(\alpha \oplus \gamma) = \mathsf{true} \Leftrightarrow \sigma(\beta \oplus \gamma) = \mathsf{true}$.

It is easy to verify that $\equiv_\sigma$ is indeed an equivalence relation. Given a property σ of structures, $p \in \mathbb{N}$ and $I \subseteq \mathbb{N}$, we let $\mathcal{E}_{\equiv_\sigma}[\mathcal{A}^p_{\subseteq I}]$ denote the set of equivalence classes of $\equiv_\sigma$ when restricted to $\mathcal{A}^p_{\subseteq I}$.

▶ **Definition 19** (Finite state)**.** A property σ of structures is *finite state* if, for every $p \in \mathbb{N}$ and $I \subseteq \mathbb{N}$, $\mathcal{E}_{\equiv_\sigma}[\mathcal{A}^p_{\subseteq I}]$ is finite.

Given a CMSO sentence ψ, the canonical equivalence relation associated with ψ is $\equiv_{\sigma_\psi}$, and for the sake of simplicity, we denote this relation by $\equiv_\psi$. We are now ready to state the variant of Courcelle's Theorem which was proven in [3] (see also [8, 7, 9]) and which we use in this paper.

▶ **Lemma 20** ([3])**.** *Every CMSO-definable property on structures has finite state.*

Parameterized Complexity. An instance of a parameterized problem is a pair of the form (x, k), where k is a non-negative integer called the *parameter*. Thus, a parameterized problem Π is a subset of $\Sigma^* \times \mathbb{N}_0$, for some finite alphabet Σ.

Two central notions in parameterized complexity are those of *uniform fixed-parameter tractability* and *non-uniform fixed-parameter tractability*. In this paper, we are interested in the second notion, which is defined as follows.

▶ **Definition 21** (Non-uniform fixed-parameter tractability (FPT)). Let Π be a parameterized problem. We say that Π is *non-uniformly fixed-parameter tractable (*FPT*)* if there exists a fixed d such that for every fixed $k \in \mathbb{N}_0$, there exists an algorithm A_k that for every $x \in \Sigma^*$, determines whether $(x, k) \in \Pi$ in time $\mathcal{O}(|x|^d)$.

Note that in Definition 21, d is independent of k. We refer the reader to the books [15, 11] for a detailed introduction to parameterized complexity.

3 CMSO Model Checking

Given a CMSO formula ψ, the CMSO$[\psi]$ problem is defined as follows. The input of CMSO$[\psi]$ is a structure α that matches ψ, and the objective is to output $\sigma_\psi(\alpha)$. In this section, we prove the following result, which then implies Theorem 1.

▶ **Theorem 22.** *Let ψ be a CMSO formula. For all $c \in \mathbb{N}$, there exists $s \in \mathbb{N}$ such that if there exists an algorithm that solves* CMSO$[\psi]$ *on (s, c)-unbreakable structures in time $\mathcal{O}(n^d)$ for some $d > 4$, then there exists an algorithm that solves* CMSO$[\psi]$ *on general structures in time $\mathcal{O}(n^d)$.*

In the context of parameterized complexity, MIN-CMSO$[\psi]$ (MIN-EDGE-CMSO$[\psi]$) is defined as follows. The input of MIN-CMSO$[\psi]$ is a structure α, where for all $S \subseteq V(G_\alpha)$ (resp. $S \subseteq E(G_\alpha)$), $\alpha \diamond S$ matches ψ, and a parameter k. The objective is to determine whether there exists $S \subseteq V(G_\alpha)$ (resp. $S \subseteq E(G_\alpha)$) of size at most k such that $\sigma_\psi(\alpha \diamond S)$ is true. Similarly, we define MAX-CMSO$[\psi]$ (resp. MAX-EDGE-CMSO$[\psi]$), where the size of S should be at least k, and EQ-CMSO$[\psi]$ (resp. EQ-EDGE-CMSO$[\psi]$), where the size of S should be exactly k. Then, as a consequence of Theorem 22, we derive the following result.

▶ **Theorem 23.** *Let* X$\in$ {MIN,MAX,EQ,MIN-EDGE,MAX-EDGE,EQ-EDGE}*, and let $\widehat{\psi}$ be a CMSO sentence. For all $\widehat{c} : \mathbb{N}_0 \to \mathbb{N}_0$, there exists $\widehat{s} : \mathbb{N}_0 \to \mathbb{N}_0$ such that if* X-CMSO$[\widehat{\psi}]$ *parameterized by k is* FPT *on $(\widehat{s}(k), \widehat{c}(k))$-unbreakable structures, then* X-CMSO$[\widehat{\psi}]$ *parameterized by k is* FPT *on general structures.*

From now on, to prove Theorem 22, we assume a fixed CMSO formula ψ and a fixed $c \in \mathbb{N}$. Moreover, we fix p as the number of free variables of ψ, and $I = [2c]$. We also let $s \in \mathbb{N}$ be fixed, where its exact value (that depends only on ψ and c) is determined later. Finally, we assume that there exists an algorithm, Solve-Unbr-ALG, that solves CMSO$[\psi]$ on (s, c)-unbreakable structures in time $\mathcal{O}(n^d)$ for some $d > 4$.

3.1 Understanding an instance of the CMSO$[\psi]$ Problem

To solve CMSO$[\psi]$, we consider a generalization of CMSO$[\psi]$, called UNDERSTAND$[\psi]$. The definition of this generalization is based on an examination of $\mathcal{E}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$. Given a boundaried structure $\alpha \in \mathcal{A}^p_{\subseteq I}$, we let E_α denote the equivalence class in $\mathcal{E}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$ that contains α. For every equivalence class $E_q \in \mathcal{E}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$, let α_{E_q} denote some boundaried structure in E_q such that there is no boundaried structure $\alpha \in E_q$ where the length of the string encoding α is smaller than the length of the string encoding α_{E_q}. Accordingly, denote $\mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}] = \{\alpha_{E_q} : E_q \in \mathcal{E}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]\}$. These will be the representatives of the equivalence classes induced by $\equiv_\psi$. By Lemma 20, there is a fixed $r \in \mathbb{N}$ (that depends only on ψ and c) such that both $|\mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]|$ and the length of encoding of any boundaried structure in $\mathcal{R}_{\equiv_\psi}$ are upper bounded by r as well as $c \leq r$. Note that the encoding explicitly lists all vertices and edges. By initially choosing s appropriately, we ensure that $s \geq 2r2^c + r$.

The UNDERSTAND[ψ] problem is defined as follows. The input is a boundaried structure $\alpha \in \mathcal{A}^p_{\subseteq I}$ that matches ψ, and the objective is to output a boundaried structure $\beta \in \mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$ such that $E_\alpha = E_\beta$. We proceed by showing that to prove Theorem 22, it is sufficient to prove that there exists an algorithm that solves UNDERSTAND[ψ] on general boundaried structures in time $\mathcal{O}(n^d)$.

▶ **Lemma 24.** *If there exists an algorithm that solves* UNDERSTAND[ψ] *on general boundaried structures in time $\mathcal{O}(n^d)$, then there exists an algorithm that solves* CMSO[ψ] *on general structures in time $\mathcal{O}(n^d)$.*

In light of Lemma 24, the rest of this section focuses on the proof of the following result.

▶ **Lemma 25.** *There exists an algorithm that solves* UNDERSTAND[ψ] *on general boundaried structures in time $\mathcal{O}(n^d)$.*

3.2 Understand[ψ] on Unbreakable Structures

Recall that $s \geq 2r2^c + r$. In this subsection, we show that Algorithm Solve-Unbr-ALG can be used as a subroutine in order to efficiently solve UNDERSTAND[ψ] on $(s-r, c)$-unbreakable boundaried structures. For this, we follow the method of test sets (see for example, [Section 12.5, [15]]). The high level idea here is as follows. We first enumerate the relevant subset of the finite set of minimal representatives. In other words, we simply list those minimal representatives which can be glued in a meaningful way to the structure under consideration, call it α. We now observe that gluing each of these representatives to α results in an (s, c)-unbreakable structure, which is what we need to call Solve-Unbr-ALG. In this way we solve the instance obtained by gluing α to each minimal representative.

Now, for every (not necessarily distinct) pair of minimal representatives, we glue them together and do the same. This way, we can identify the specific minimal representative whose behaviour when glued with *every* minimal representative, precisely resembles that of the structure α when we do the same with α. Consequently, we obtain a solution for UNDERSTAND[ψ]. We now formalize this intuition in the following lemma.

▶ **Lemma 26.** *There exists an algorithm* Understand-Unbr-ALG*, that solves* UNDERSTAND[ψ]*, where it is guaranteed that inputs are $(s-r, c)$-unbreakable boundaried structures, in time $\mathcal{O}(n^d)$.*[6]

Proof. We design the algorithm Understand-Unbr-ALG as follows. Let α be an input, which is an $(s-r, c)$-unbreakable boundaried structure. Moreover, let $\mathcal{C} = \{\gamma \in \mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}] : \gamma \equiv_c \alpha\}$, and let $\mathcal{T}$ denote the set of boundaried structures in $\mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$ that are compatible with α. In the first phase, the algorithm performs the following computation. Notice that for every $\beta \in \mathcal{T}$, since $|V(G_\beta)| \leq r$, it holds that $\alpha \oplus \beta$ is an (s, c)-unbreakable structure. Thus, for every $\beta \in \mathcal{T}$, Understand-Unbr-ALG can call Solve-Unbr-ALG with $\alpha \oplus \beta$ as input, and it lets $\mathsf{ans}(\alpha, \beta)$ denote the result.

In the second phase, the algorithm performs the following computation. Notice that for every $\gamma \in \mathcal{C}$ and $\beta \in \mathcal{T}$, since $|V(G_\beta)|, |V(G_\gamma)| \leq r$, it holds that $\gamma \oplus \beta$ is a $(2r, c)$-unbreakable structure. Thus, since $s \geq 2r2^c + r$, for all $\beta \in \mathcal{C}$ and $\gamma \in \mathcal{T}$, Understand-Unbr-ALG can call Solve-Unbr-ALG with $\gamma \oplus \beta$ as input, and it lets $\mathsf{ans}(\gamma, \beta)$ denote the result.

[6] Here, Understand-Unbr-ALG is not required to verify whether the input is an $(s-r, c)$-unbreakable boundaried structure.

Finally, in the third phase, for every $\beta \in \mathcal{C}$, the algorithm performs the following computation. It checks whether for every $\gamma \in \mathcal{T}$ it holds that $\mathsf{ans}(\alpha, \gamma) = \mathsf{ans}(\beta, \gamma)$, and if the answer is positive, then it outputs β. Since $\alpha \in \mathcal{A}^p_{\subseteq I}$, there exists $\beta' \in \mathcal{C}$ such that $E_\alpha = E_{\beta'}$, and therefore, at the latest, when $\beta = \beta'$, the algorithm terminates. Thus, the algorithm is well defined, and it is clear that it runs in time $\mathcal{O}(n^d)$.

To conclude that the algorithm is correct, it remains to show that for all $\beta \in \mathcal{C} \setminus \{\beta'\}$, there exists $\gamma \in \mathcal{T}$ such that $\mathsf{ans}(\alpha, \gamma) \neq \mathsf{ans}(\beta, \gamma)$, as this would imply that the algorithm necessarily outputs β'. For this purpose, suppose by way of contradiction that there exists $\beta \in \mathcal{C} \setminus \{\beta'\}$ such that for all $\gamma \in \mathcal{T}$ it holds that $\mathsf{ans}(\alpha, \gamma) = \mathsf{ans}(\beta, \gamma)$. We now argue that $E_\beta = E_{\beta'}$ which leads to a contradiction since each boundaried structure in $\mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$ belongs to a different equivalence class.

For all $\gamma \in \mathcal{T}$, since it holds that $\mathsf{ans}(\alpha, \gamma) = \mathsf{ans}(\beta, \gamma)$, it also holds that $\mathsf{ans}(\beta', \gamma) = \mathsf{ans}(\beta, \gamma)$. This implies that $\sigma_\psi(\beta' \oplus \gamma) = \sigma_\psi(\beta \oplus \gamma)$. Consider some boundaried structure γ (not necessarily in $\mathcal{T}$) that is compatible with β' (and thus also with β). We claim that $\sigma_\psi(\beta' \oplus \gamma) = \sigma_\psi(\beta \oplus \gamma)$. Indeed, let γ' be the (unique) boundaried structure in $\mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$ such that $E_{\gamma'} = E_\gamma$. Then, $\sigma_\psi(\beta' \oplus \gamma') = \sigma_\psi(\beta' \oplus \gamma)$ and $\sigma_\psi(\beta \oplus \gamma') = \sigma_\psi(\beta \oplus \gamma)$. Note that since γ' is compatible with β', it is also compatible with α, and hence $\gamma' \in \mathcal{T}$. Therefore, $\sigma_\psi(\beta' \oplus \gamma') = \sigma_\psi(\beta \oplus \gamma')$. Overall, we obtain that indeed $\sigma_\psi(\beta' \oplus \gamma) = \sigma_\psi(\beta \oplus \gamma)$.

Note that $\beta \equiv_c \beta'$, and thus, since we have shown that for every boundaried structure γ compatible with β' it holds that $\sigma_\psi(\beta' \oplus \gamma) = \sigma_\psi(\beta \oplus \gamma)$, we derive that $E_\beta = E_{\beta'}$. However, each boundaried structure in $\mathcal{R}_{\equiv_\psi}[\mathcal{A}^p_{\subseteq I}]$ belongs to a different equivalence class, and thus we have reached the desired contradiction. ◀

3.3 Understand[ψ] on General Structures

The Algorithm Understand-ALG. We start by describing an algorithm called Understand-ALG, which is based on recursion. Given an input to Understand[ψ] on general boundaried structures, which is a boundaried structure α, the algorithm works as follows. First, it calls Break-ALG (given by Lemma 5) with G_α as input to either obtain an $(\frac{s-r}{2^c}, c)$-witnessing separation (X, Y) or correctly conclude that G_α is $(s-r, c)$-unbreakable. In the second case or if $n < 2(s-r)$, it calls Understand-Unbr-ALG (given by Lemma 26), and returns its output. Next, suppose that Understand-ALG obtained an $(\frac{s-r}{2^c}, c)$-witnessing separation (X, Y) and that $n \geq 2(s-r)$. Without loss of generality, assume that $|X \cap \delta(G_\alpha)| \leq |Y \cap \delta(G_\alpha)|$. Denote $\Delta = \{v \in X \cap Y : v \notin \delta(G_\alpha)\}$.

Now, we define a boundaried structure, $\beta \in \mathcal{A}^p_{\subseteq I}$, which can serve as an instance of Understand[ψ]. First, we let the graph G_β be $G_\alpha[X]$, and we define $\delta(G_\beta) = (X \cap \delta(G_\alpha)) \cup \Delta$. Now, for all $v \in X \cap \delta(G_\alpha)$, we define $\lambda_{G_\beta}(v) = \lambda_{G_\alpha}(v)$. Since $|X \cap \delta(G_\alpha)| \leq |Y \cap \delta(G_\alpha)|$, $\alpha \in \mathcal{A}^p_{\subseteq I}$ and $|X \cap Y| \leq c$, we have that $|(X \cap \delta(G_\alpha)) \cup \Delta| \leq 2c$. Thus, to each $v \in \Delta$, we can let $\lambda_{G_\beta}(v)$ assign some unique integer from $I \setminus \lambda_{G_\alpha}(X \cap \delta(G_\alpha))$. Hence, $G_\beta \in \mathcal{F}_{\subseteq I}$. Now, for all $i \in \{2, \ldots, p\}$, we set $\beta[i]$ as follows. If $\mathbf{type}(\alpha)[i] \in \{\mathsf{vertex}, \mathsf{edge}\}$: If $\alpha[i] \in V(G_\beta) \cup E(G_\beta)$, then $\beta[i] = \alpha[i]$, and otherwise $\beta[i] = \star$. Else: $\beta[i] = \alpha[i] \cap (V(G_\beta) \cup E(G_\beta))$.

Understand-ALG proceeds by calling itself recursively with β as input, and it lets β' be the output of this call. Now, we define another boundaried structure, $\gamma \in \mathcal{A}^p_{\subseteq I}$, which can serve as an instance of Understand[ψ]. First, we define the boundaried graph G_γ as follows. Let H be the disjoint union of $G_{\beta'}$ and $G[Y]$, where both $G_{\beta'}$ and $G[Y]$ are treated as not-boundaried graphs. For all $v \in X \cap Y$, identify (in H) the vertex v of $G[Y]$ with the vertex u of $G_{\beta'}$ that satisfies $\lambda_{G_{\beta'}}(u) = \lambda_{G_\beta}(v)$, and for the sake of simplicity, let v and u also denote the identity of the resulting (unified) vertex. The graph G_γ is the result

of this process. Moreover, let Δ' denote the set of vertices in $G_{\beta'}$ whose labels belong to $G_\beta(\Delta)$. Next, set $\delta(G_\gamma) = (Y \cap \delta(G_\alpha)) \cup (\delta(G_{\beta'}) \setminus \Delta')$. Now, for all $v \in Y \cap \delta(G_\alpha)$, we define $\lambda_{G_\gamma}(v) = \lambda_{G_\alpha}(v)$, and for all $v \in \delta(G_{\beta'}) \setminus \Delta'$, we define $\lambda_{G_\gamma}(v) = \lambda_{G_{\beta'}}(v)$ (note that if a vertex belongs to both $Y \cap \delta(G_\alpha)$ and $\delta(G_{\beta'}) \setminus \Delta'$, we still assign it the same label). Hence, $G_\gamma \in \mathcal{F}_{\subseteq I}$. For the sake of simplicity, if two vertices have the same label (one in G_α and the other in G_γ), we let the identity of one of them also refer to the other and vice versa. For all $i \in \{2, \ldots, p\}$, we set $\gamma[i]$ to have the same type as $\alpha[i]$, and define it as follows. If $\mathbf{type}(\alpha)[i] \in \{\mathsf{vertex}, \mathsf{edge}\}$: If $\alpha[i] \in V(G_\gamma) \cup E(G_\gamma)$, then $\gamma[i] = \alpha[i]$, and otherwise $\gamma[i] = \star$. Else: $\gamma[i] = \alpha[i] \cap (V(G_\gamma) \cup E(G_\gamma))$. Finally, Understand-ALG calls itself recursively with γ as input, and it returns γ', the output of this call.

Correctness and running time. Finally, we prove the following two results. Along with Lemma 27, these complete the proof of Lemma 25.

▶ **Lemma 27.** *If* Understand-ALG *terminates, then it correctly solves* Understand$[\psi]$ *on general boundaried structures.*

▶ **Lemma 28.** Understand-ALG *runs in time* $\mathcal{O}(n^d)$.

4 Applications

In this section, we first show how Theorem 23 can be easily deployed to show the fixed parameter tractability of a wide range of problems of the following kind. The input is a graph G and the task is to find a connected induced subgraph of G of bounded treewidth such that "few" vertices outside this subgraph have neighbors inside the subgraph, and additionally the subgraph has a CMSO-definable property. Then, we show that technical problem-specific ingredients of a powerful method for designing parameterized algorithms called recursive understanding, can be replaced by a black-box invocation of Theorem 23.

4.1 "Pendant" Subgraphs with CMSO-Definable Properties

Formally, given a CMSO sentence ψ and a non-negative integer t, the t-Pendant$[\psi]$ problem is defined as follows. The input of t-Pendant$[\psi]$ is a graph G and a parameter k, and the objective is to determine whether there exists $U \subseteq V(G)$ such that $G[U]$ is a connected graph of treewidth at most t, $|N(U)| \leq k$ and $\sigma_\psi(G[U])$ is true.

It is straightforward to define a CMSO formula φ with free variable S such that the t-Pendant$[\psi]$ problem is equivalent to min-CMSO$[\varphi]$ as follows.

▶ **Observation 4.1.** *Let G be a graph, and let k be a parameter. Then, (G, k) is a* Yes*-instance of t-*Pendant$[\psi]$ *if and only if $((G), k)$ is a* Yes*-instance of* min-CMSO$[\varphi]$.

Next, we solve t-Pendant$[\psi]$ on unbreakable graphs with the appropriate parameters. Define $c : \mathbb{N}_0 \to \mathbb{N}_0$ as follows. For all $k \in \mathbb{N}_0$, let $\widehat{c}(k) = k + t$. Let $s : \mathbb{N}_0 \to \mathbb{N}_0$ be the function $\widehat{s}$ in Theorem 23 with $\widehat{\psi} = \varphi$ and $\widehat{c} = c$. We first prove the following lemma.

▶ **Lemma 29.** *Let (G, k) be a* Yes*-instance of t-*Pendant$[\psi]$ *parameterized by k on $(s(k), k + t)$-unbreakable graphs. Then, there exists $U \subseteq V(G)$ such that $G[U]$ is a connected graph of treewidth at most t, $|N(U)| \leq k$, $\sigma_\psi(G[U])$ is* true *and $|U| < 3(s(k) + t)$.*

Proof. Since (G, k) is a Yes-instance, there exists $U \subseteq V(G)$ such that $G[U]$ is a connected graph of treewidth at most t, $|N(U)| \leq k$ and $\sigma_\psi(G[U])$ is true. Moreover, since the

treewidth of $G[U]$ is at most t, it is easy to see that there exists a separation (X, Y) of order at most t of $G[U]$ such that $|X|, |Y| \geq |U|/3$ (see, e.g., [11]). Then, set $X' = X \cup N(U)$ and $Y' = (V(G) \setminus X) \cup (X \cap Y)$. Note that (X', Y') is a separation of order $|X \cap Y| + |N(U)| \leq k+t$. Moreover, $X \setminus Y \subseteq X' \setminus Y'$ and $Y \setminus X \subseteq Y' \setminus X'$. Thus, (X', Y') is a $(|U|/3 - t, k + t)$-witnessing separation. Since G is $(s(k), k+t)$-unbreakable graph, we have that $|U|/3 - t < s(k)$. Therefore, $|U| < 3(s(k) + t)$, which concludes the correctness of the lemma. ◀

▶ **Lemma 30.** *t-Pendant$[\psi]$ parameterized by k is* FPT *on $(s(k), k + t)$-unbreakable graphs.*

Finally, by Theorem 23, Observation 4.1 and Lemma 30, we derive the following result.

▶ **Theorem 31.** *t-Pendant$[\psi]$ parameterized by k is* FPT *on general graphs.*

References

1. Karl Abrahamson and Michael Fellows. Finite automata, bounded treewidth and well-quasiordering. In *Graph structure theory (Seattle, WA, 1991)*, volume 147 of *Contemp. Math.*, pages 539–563, Providence, RI, 1993. Amer. Math. Soc. doi:10.1090/conm/147/01199.
2. Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.
3. Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) kernelization. *J. ACM*, 63(5):44:1–44:69, 2016. URL: http://dl.acm.org/citation.cfm?id=2973749, doi:10.1145/2973749.
4. Richard B. Borie, R. Gary Parker, and Craig A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–581, 1992.
5. Simone Bova, Robert Ganian, and Stefan Szeider. Model checking existential logic on partially ordered sets. *ACM Trans. Comput. Log.*, 17(2):10:1–10:35, 2016. doi:10.1145/2814937.
6. Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM J. Comput.*, 45(4):1171–1229, 2016. doi:10.1137/15M1032077.
7. B. Courcelle. The monadic second-order logic of graphs. III. Tree-decompositions, minors and complexity issues. *RAIRO Inform. Théor. Appl.*, 26(3):257–286, 1992.
8. Bruno Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Inform. and Comput.*, 85:12–75, 1990.
9. Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In *Handbook of graph grammars and computing by graph transformation, Vol. 1*, pages 313–400. World Sci. Publ, River Edge, NJ, 1997.
10. Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Cambridge University Press, 2012.
11. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
12. A. Dawar, M. Grohe, and S. Kreutzer. Locally excluding a minor. In *LICS'07*, pages 270–279. IEEE Computer Society, 2007.
13. Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.
14. Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, Berlin, 1998.

15 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

16 Zdenek Dvorak, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36:1–36:24, 2013. doi:10.1145/2499483.

17 Michael R. Fellows and Michael A. Langston. An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations (extended abstract). In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS 1989)*, pages 520–525. IEEE, 1989.

18 J. Flum and M. Grohe. Fixed-parameter tractability, definability, and model-checking. *SIAM J. Comput.*, 31(1):113–145, 2001.

19 M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001.

20 Jakub Gajarský, Petr Hlinený, Jan Obdrzálek, and Sebastian Ordyniak. Faster existential FO model checking on posets. *Logical Methods in Computer Science*, 11(4), 2015. doi:10.2168/LMCS-11(4:8)2015.

21 Robert Ganian, Petr Hlinený, Daniel Král, Jan Obdrzálek, Jarett Schwartz, and Jakub Teska. FO model checking of interval graphs. *Logical Methods in Computer Science*, 11(4), 2015. doi:10.2168/LMCS-11(4:11)2015.

22 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 479–488, 2011. doi:10.1145/1993636.1993700.

23 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

24 Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 160–169, 2011. doi:10.1109/FOCS.2011.53.

25 Eun Jung Kim, Christophe Paul, Ignasi Sau, and Dimitrios M. Thilikos. Parameterized algorithms for min-max multiway cut and list digraph homomorphism. *J. Comput. Syst. Sci.*, 86:191–206, 2017. doi:10.1016/j.jcss.2017.01.003.

26 Ashutosh Rai and M. S. Ramanujan. Strong parameterized deletion: Bipartite graphs. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, pages 21:1–21:14, 2016. doi:10.4230/LIPIcs.FSTTCS.2016.21.

27 Ashutosh Rai, M. S. Ramanujan, and Saket Saurabh. A parameterized algorithm for mixed-cut. In *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, pages 672–685, 2016. doi:10.1007/978-3-662-49529-2_50.

28 D. Seese. Linear time computable problems and first-order descriptions. *Math. Structures Comput. Sci.*, 6(6):505–526, 1996.

An Optimal Bound on the Solution Sets of One-Variable Word Equations and its Consequences

Dirk Nowotka[1]
Department of Computer Science, Kiel University, 24098 Kiel, Germany
dn@informatik.uni-kiel.de

Aleksi Saarela
Department of Mathematics and Statistics, University of Turku, 20014 Turku, Finland
amsaar@utu.fi
https://orcid.org/0000-0002-6636-2317

Abstract

We solve two long-standing open problems on word equations. Firstly, we prove that a one-variable word equation with constants has either at most three or an infinite number of solutions. The existence of such a bound had been conjectured, and the bound three is optimal. Secondly, we consider independent systems of three-variable word equations without constants. If such a system has a nonperiodic solution, then this system of equations is at most of size 17. Although probably not optimal, this is the first finite bound found. However, the conjecture of that bound being actually two still remains open.

2012 ACM Subject Classification Mathematics of computing → Combinatorics on words

Keywords and phrases combinatorics on words, word equations, systems of equations

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.136

Related Version A full version of the paper is available at `http://arxiv.org/abs/1805.09535`.

1 Introduction

If n words satisfy a nontrivial relation, they can be written as products of $n-1$ words. This folklore result is known as the defect theorem, and it can be seen as analogous to the simple fact of linear algebra that the dimension of the solution space of a homogeneous n-variable linear equation is $n-1$. If an independent equation is added to a system of linear equations, the dimension of the solution space decreases, which gives an upper bound n for the size of independent systems of linear equations, but no such results are known for word equations. In fact, the maximal size of independent systems of constant-free word equations has been one of the biggest open questions in combinatorics on words for many decades. In 1983, Culik and Karhumäki [4] pointed out that a conjecture of Ehrenfeucht about test sets of formal languages can be equivalently formulated as claiming that every infinite system of word equations is equivalent to a finite subsystem. Ehrenfeucht's conjecture was proved by Albert and Lawrence [1] and independently by Guba [9], and it follows that independent systems cannot be infinite, but no finite upper bounds depending only on the number of variables have been found. Independent systems of size $\Theta(n^4)$ on n variables were constructed by

[1] This work was partially supported by the DFG research project 181615770

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 136; pp. 136:1–136:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Karhumäki and Plandowski [13], and the hidden constant in $\Theta(n^4)$ was improved in [14]. This is the best known lower bound.

The case of three variables is particularly interesting. In this case, it is easy to find systems of size two that are independent and have a nonperiodic solution, or systems of size three that are independent but have no nonperiodic solution, and Culik and Karhumäki conjectured that there are no larger such systems, but no finite upper bounds have been found even in this case. In fact, despite Ehrenfeucht's conjecture, even the existence of a bound is not guaranteed, because in principle it might be possible that there are unboundedly large finite independent systems. This case of three variables is very striking because it is the simplest nontrivial case, but the gap between the almost trivial lower bound and the infinite upper bound has remained huge despite the considerable attention the problem has received. Some results about systems of specific forms are known [10, 5, 6], and some upper bounds that depend on the sizes of the equations have been proved [17, 11, 16]. The best current bound is logarithmic with respect to the size of the smallest equation in the system [16].

In the above, we have considered constant-free word equations. If we add constants, the equations become more complicated. For constant-free equations, the three-variable case is the first nontrivial one, but for equations with constants, already the one-variable case is interesting. One-variable equations have been studied in many articles [8, 7, 15], and the main open question about them is the maximal number of solutions such an equation can have if we exclude equations with infinitely many solutions (if the solution set is infinite, it is known to be of a very specific form). Even finding an example with exactly two solutions is not entirely trivial, but a simple example was given by Laine and Plandowski [15]. An example with exactly three solutions was recently found [16]. No fixed upper bound, or even the existence of an upper bound, has been proved. The best known result is a bound that depends logarithmically on the number of occurrences of the variable in the equation [15]. It can be noted that the solutions of a one-variable equation can be found in linear time in the RAM model, as proved by Jeż [12].

In this article, we solve the open problem about sizes of solution sets of one-variable equations by proving that a one-variable equation has either infinitely many solutions or at most three, which is an optimal result. As a consequence, we prove the first upper bound for the sizes of independent systems of constant-free three-variable equations, thus settling the old open question about the existence of such a bound. More specifically, we prove that if an independent system of constant-free three-variable equations is independent and has a nonperiodic solution, then the system is of size at most 17 (if the system is not required to have a nonperiodic solution, then the size can be at most one larger). This bound is probably not optimal and the conjecture of Culik and Karhumäki remains open, as does the more general question about n-variable equations.

Two previous articles provide crucial tools for our proofs. The first article is [18], where new methods were introduced to solve a certain open problem on word equations. We use and further develop these methods to analyze one-variable equations. The second article is [16], where a surprising connection between the two topics we have discussed above was found: It was proved that a bound for the maximal size of a finite solution set of a one-variable equation implies a (larger) bound for the maximal size of independent systems of constant-free three-variable equations.

Some of the proofs have been omitted from this conference version to save space.

2 Preliminaries

We begin this section by considering constant-free word equations. Let Ξ be an alphabet of variables and Γ an alphabet of constants. A *constant-free word equation* is a pair $(U, V) \in \Xi^* \times \Xi^*$, and the solutions of this equation are the morphisms $h : \Xi^* \to \Gamma^*$ such that $h(U) = h(V)$. A solution h is *periodic* if there exists $p \in \Gamma^*$ such that $h(X) \in p^*$ for all $X \in \Xi$. Otherwise, h is *nonperiodic*. It is well-known that h is periodic if and only if $h(PQ) = h(QP)$ for all words $P, Q \in \Xi^*$.

▶ **Example 1.** Let $\Xi = \{X, Y, Z\}$ and consider the equation (XYZ, ZYX). For all $p, q \in \Gamma^*$ and $i, j, k \geq 0$, the morphism h defined by $h(X) = (pq)^i p$, $h(Y) = (qp)^j q$, $h(Z) = (pq)^k p$ is a solution of this equation because

$$h(XYZ) = (pq)^i p \cdot (qp)^j q \cdot (pq)^k p = (pq)^{i+j+k+1} p = (pq)^k p \cdot (qp)^j q \cdot (pq)^i p = h(ZYX).$$

Every nonperiodic solution of the equation is of this form.

A set of equations is a *system of equations*. A morphism is a solution of a system if it is a solution of every equation in the system. Two equations or systems are *equivalent* if they have exactly the same solutions. A system of equations is *independent* if it is not equivalent to any of its proper subsets.

▶ **Example 2.** Let $\Xi = \{X, Y, Z\}$ and $\Gamma = \{a, b\}$. The system of equations $S = \{(XYZ, ZYX), (XYYZ, ZYYX)\}$ is independent and has a nonperiodic solution h defined by $h(X) = a$, $h(Y) = b$, $h(Z) = a$. To see independence, note that S is not equivalent to (XYZ, ZYX), because the morphism h defined by $h(X) = a$, $h(Y) = b$, $h(Z) = aba$ is a solution of (XYZ, ZYX) but not of S, and S is not equivalent to $(XYYZ, ZYYX)$, because the morphism h defined by $h(X) = a$, $h(Y) = b$, $h(Z) = abba$ is a solution of $(XYYZ, ZYYX)$ but not of S.

The following question is a big open problem on word equations: If a system of constant-free three-variable equations is independent and has a nonperiodic solution, then how large can the system be? The largest known examples are of size two, see Example 2, and it has been conjectured that these examples are optimal. Even the following weaker conjecture is open.

▶ **Conjecture 3.** *There exists a number c such that every independent system of constant-free three-variable equations with a nonperiodic solution is of size c or less.*

Currently, the best known result is the following.

▶ **Theorem 4** ([16])**.** *Every independent system of constant-free three-variable equations is of size $O(\log n)$, where n is the length of the shortest equation.*

Next, we will consider word equations with constants. As before, let Ξ be an alphabet of variables and Γ an alphabet of constants. A *word equation with constants* is a pair $(U, V) \in (\Xi \cup \Gamma)^* \times (\Xi \cup \Gamma)^*$, and the solutions of this equation are the constant-preserving morphisms $h : (\Xi \cup \Gamma)^* \to \Gamma^*$ such that $h(U) = h(V)$. If $U = V$, then the equation is *trivial*.

In this article, we are interested in the one-variable case $\Xi = \{X\}$. We use the notation $[u]$ for the constant-preserving morphism $h : (\{X\} \cup \Gamma)^* \to \Gamma^*$ defined by $h(X) = u$. If S is a set of words, we use the notation $[S] = \{[u] \mid u \in S\}$. If $[u]$ is a solution of a one-variable equation E, then u is called a *solution word* of E. The set of all solutions of E is denoted by $\mathrm{Sol}(E)$.

▶ **Example 5.** Let $\Gamma = \{a, b\}$. The equation (Xab, abX) has infinitely many solutions $[(ab)^i]$, where $i \geq 0$. The equation $(XaXbab, abaXbX)$ has exactly two solutions $[\varepsilon]$ and $[ab]$. The equation $(XXbaaba, aabaXbX)$ has exactly two solutions $[a]$ and $[aaba]$. The equation

$$(XaXbXaabbabaXbabaabbab, abaabbabaXbabaabbXaXbX)$$

has exactly three solutions $[\varepsilon], [ab], [abaabbab]$.

The following is a well-known open problem: If a one-variable equation has only finitely many solutions, then what is the maximal number of solutions it can have? Example 5 shows that the answer is at least three, but no upper bound is known. Currently, the best known result is the following.

▶ **Theorem 6** ([15, Theorems 23, 26, 29]). *If the solution set of a one-variable equation is finite, then it has size at most* $8 \log n + O(1)$*, where* n *is the number of occurrences of the variable.*

If the solution set is infinite and the equation is not trivial, then there are words p, q *such that* pq *is primitive and the solution set is* $[(pq)^* p]$*.*

We will need the following lemma.

▶ **Lemma 7** ([7, Lemma 1]). *Let* E *be a one-variable equation and let* pq *be primitive. The set*

$$\mathrm{Sol}(E) \cap [(pq)^+ p]$$

is either $[(pq)^+ p]$ *or has at most one element.*

A connection between constant-free three-variable equations and one-variable equations with constants was recently found [16]. Here we give the relevant special case of one of the results.

▶ **Theorem 8** ([16]). *If every one-variable word equation has either infinitely many solutions or at most three, then Conjecture 3 is true for* $c = 17$*.*

In this article, we will prove that every one-variable word equation has either infinitely many solutions or at most three, and thus Conjecture 3 is true for $c = 17$.

3 Sums of words

In this section, we will give some definitions and ideas that will be used in our proofs. Most of these were introduced in [18].

We can assume that the alphabet Γ is a subset of $\mathbb{R}$. Then we can define $\Sigma(w)$ to be the sum of the letters of a word $w \in \Gamma^*$, that is, if $w = a_1 \cdots a_n$ and $a_1, \ldots, a_n \in \Gamma$, then $\Sigma(w) = a_1 + \cdots + a_n$. Words w such that $\Sigma(w) = 0$ are called *zero-sum words*. If w is zero-sum, then the morphism $[w]$ can also be called zero-sum. The largest and smallest letters in a word w can be denoted by $\max(w)$ and $\min(w)$, respectively.

The *prefix sum word* of $w = a_1 \cdots a_n$ is the word $\mathrm{psw}(w) = b_1 \cdots b_n$, where $b_i = \Sigma(a_1 \cdots a_i)$ for all i. Of course, $\mathrm{psw}(w)$ is usually not a word over Γ, but over some other alphabet. The mapping psw is injective and length-preserving. We also use the notation $\mathrm{psw}_r(w) = c_1 \cdots c_n$, where $r \in \mathbb{R}$ and $c_i = b_i + r$ for all i.

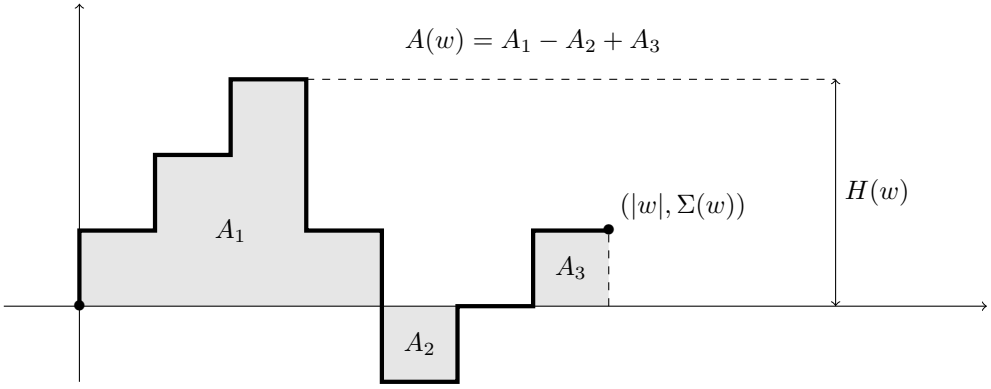


Figure 1 Representation of the word $w = aaabbaa$, where $a = 1$ and $b = -2$. We have $|w| = 7$, $\Sigma(w) = 1$, $H(w) = 3$, and $A(w) = 7$.

▶ **Example 9.** Let $w = bbcaac$, where $a = 1$, $b = 2$, and $c = -3$. We have $|w| = 6$, $\max(w) = 2$, and $\min(w) = -3$. Because $\Sigma(w) = 2 + 2 - 3 + 1 + 1 - 3 = 0$, w is a zero-sum word. The prefix sum word of w is $\mathrm{psw}(w) = 241230$, and $\max(\mathrm{psw}(w)) = 4$ and $\min(\mathrm{psw}(w)) = 0$.

For a word w, we define its *height* $H(w)$ and *area* $A(w)$:

$$H(w) = \max(\mathrm{psw}(w)) = \max\{\Sigma(u) \mid \varepsilon \neq u \sqsubseteq w\},$$
$$A(w) = \Sigma(\mathrm{psw}(w)) = \sum_{u \sqsubseteq w} \Sigma(u),$$

where $u \sqsubseteq w$ means that u is a prefix of w. For the empty word, $H(\varepsilon) = -\infty$ and $A(\varepsilon) = 0$.

These definitions have the following graphical interpretation: A word $w = a_1 \cdots a_n$ can be represented by a polygonal chain by starting at the origin, moving a_1 steps up, one step to the right, a_2 steps up, one step to the right, and so on. The end point of this curve is then $(|w|, \Sigma(w))$. The biggest y-coordinate (after the initial line segment starting at the origin) is $H(w)$. The number $A(w)$ is the area under the curve, defined in the same way as a definite integral, that is, parts below the x-axis count as negative areas. See Figure 1 for an example.

▶ **Lemma 10.** *For words $w_1, \ldots, w_n$, we have*

$$\Sigma(w_1 \cdots w_n) = \Sigma(w_1) + \cdots + \Sigma(w_n),$$
$$\mathrm{psw}(w_1 \cdots w_n) = \prod_{i=1}^{n} \mathrm{psw}_{\Sigma(w_1 \cdots w_{i-1})}(w_i),$$
$$H(w_1 \cdots w_n) = \max\{\Sigma(w_1 \cdots w_{i-1}) + H(w_i) \mid 1 \leq i \leq n\},$$
$$A(w_1 \cdots w_n) = \sum_{i=1}^{n} (A(w_i) + \Sigma(w_1 \cdots w_{i-1})|w_i|).$$

Proof. Follows easily from the definitions. ◀

When studying words from a combinatorial point of view, the choice of the alphabet is arbitrary (except for the size of the alphabet), so we can assign numerical values to the letters in any way we like, as long as no two letters get the same value. The next two lemmas show that, given any word w, the alphabet can be normalized so that w becomes a zero-sum word, and every zero-sum word can be written as a product of minimal zero-sum words in a unique way.

▶ **Lemma 11** ([18, Lemma 3]). *Let $w \in \Gamma^*$. There exists an alphabet Δ and an isomorphism $h : \Gamma^* \to \Delta^*$ such that $h(w)$ is zero-sum.*

▶ **Lemma 12** ([18, Lemma 4]). *The set of zero-sum words over Γ is a free monoid.*

4 Equations in normal form

If a one-variable equation has more occurrences of the variable on the left-hand side than on the right-hand side, or vice versa, then it is easy to see by a length argument that it can have at most one solution. Therefore every one-variable equation with more than one solution can be written in the form

$$(u_0Xu_1 \cdots Xu_n, v_0Xv_1 \cdots Xv_n), \tag{1}$$

where X is the variable, $n \geq 1$, and $u_0, \ldots, u_n, v_0, \ldots, v_n$ are constant words. Clearly, it must be $|u_0 \cdots u_n| = |v_0 \cdots v_n|$. If the equation is nontrivial, x_1, x_2 are solution words, and $|x_1| \leq |x_2|$, then it is quite easy to see that x_1 is a prefix and a suffix of x_2.

We say that the equation (1) is in *normal form* if the following conditions are satisfied:

(N1) It has the empty solution and at least one other zero-sum solution,
(N2) $|u_0 \cdots u_i| < |v_0 \cdots v_i|$ for all $i \in \{0, \ldots, n-1\}$,
(N3) $|u_0 \cdots u_i| \leq |v_0 \cdots v_{i-1}|$ for all $i \in \{0, \ldots, n\}$.

It follows from these conditions that $u_0 = v_n = \varepsilon$. By the next two lemmas, it is usually sufficient to consider equations in normal form.

▶ **Lemma 13.** *Let E be a one-variable equation,* $\mathrm{Sol}(E) = \{[x_0], \ldots, [x_m]\}$, *and $|x_0| \leq |x_i|$ for all i. There exists a one-variable equation E' such that* $\mathrm{Sol}(E') = \{[\varepsilon], [x_0^{-1}x_1], \ldots, [x_0^{-1}x_m]\}$.

Proof. If $m = 0$, the claim is clear. Otherwise, we can assume that E is of the form (1). Let E' be the equation we get from E by replacing X by x_0X:

$$E' : (u_0x_0Xu_1 \cdots x_0Xu_n, v_0x_0Xv_1 \cdots x_0Xv_n).$$

Because E is nontrivial, x_0 is a prefix of every x_i. Clearly, the word $x_0^{-1}x_i$ is a solution word of E'. On the other hand, if x is a solution word of E', then x_0x is a solution word of E. This proves the claim. ◀

Next we will give an example of how to transform an equation that satisfies Condition N1 into an equation in normal form.

▶ **Example 14.** Consider the equation

$$(XabXababXaabaXbX, abXXXababaXaXbab).$$

By a length argument, it is equivalent to the system of equations

$$(Xab, abX), (X, X), (ababX, Xabab), (a, a), (abaXbX, XaXbab).$$

We can drop the trivial equations (X, X) and (a, a), and then switch the left-hand and right-hand sides of the equations $(ababX, Xabab)$ and $(abaXbX, XaXbab)$ to get the system

$$(Xab, abX), (Xabab, ababX), (XaXbab, abaXbX).$$

Then we can combine these equations into the equation

$$(XabXababXaXbab, abXababXabaXbX),$$

which satisfies Conditions N2 and N3. (Actually, this equation is equivalent to the equation $(XaXbab, abaXbX)$.)

▶ **Lemma 15.** *Let E be a nontrivial one-variable equation with the empty solution and at least one other solution. There exists an equation in normal form that is equivalent to E up to a renaming of the letters and not longer than E.*

Proof. Omitted. ◀

5 Sums and heights of solutions

In this section, we prove lemmas about the sums and heights of solution words of one-variable equations in normal form.

▶ **Lemma 16.** *All solutions of an equation in normal form are zero-sum.*

Proof. Let the equation be (1). Let $u'_i = u_0 \cdots u_{i-1}$ and $v'_i = v_0 \cdots v_{i-1}$ for all i. After applying a solution $[x]$ on the left-hand side and taking the area we get

$$\begin{aligned}
&A(u_0 x u_1 \cdots x u_n)\\
=&\sum_{i=0}^{n}(A(u_i) + \Sigma(u_0 x u_1 \cdots u_{i-1}x)|u_i|) + \sum_{i=1}^{n}(A(x) + \Sigma(u_0 x u_1 \cdots x u_{i-1})|x|)\\
=&\sum_{i=0}^{n}(A(u_i) + \Sigma(u'_i)|u_i| + i\Sigma(x)|u_i|) + \sum_{i=1}^{n}(A(x) + \Sigma(u'_i)|x| + (i-1)\Sigma(x)|x|)\\
=&A(u_0 \cdots u_n) + \Sigma(x)\sum_{i=0}^{n} i|u_i| + nA(x) + |x|\sum_{i=1}^{n}\Sigma(u'_i) + \frac{(n-1)n}{2}\cdot\Sigma(x)|x|.
\end{aligned}$$

We get a similar formula for $A(v_0 x v_1 \cdots x v_n)$. Because $u_0 x u_1 \cdots x u_n = v_0 x v_1 \cdots x v_n$, we get

$$\begin{aligned}
0 &= A(u_0 x u_1 \cdots x u_n) - A(v_0 x v_1 \cdots x v_n)\\
&= A(u_0 \cdots u_n) - A(v_0 \cdots v_n) + \Sigma(x)\sum_{i=0}^{n} i(|u_i| - |v_i|) + |x|\sum_{i=1}^{n}(\Sigma(u'_i) - \Sigma(v'_i))\\
&= \Sigma(x)\sum_{i=0}^{n} i(|u_i| - |v_i|) + |x|\sum_{i=1}^{n}(\Sigma(u'_i) - \Sigma(v'_i)). \qquad (2)
\end{aligned}$$

By the definition of normal form, the equation has a nonempty zero-sum solution $[x_1]$. Replacing x by x_1 in (2) gives

$$0 = |x_1|\sum_{i=1}^{n}(\Sigma(u'_i) - \Sigma(v'_i)).$$

Because $|x_1| > 0$, $\sum_{i=1}^{n}(\Sigma(u'_i) - \Sigma(v'_i)) = 0$. Then (2) takes the form

$$0 = \Sigma(x)\sum_{i=0}^{n} i(|u_i| - |v_i|),$$

so either $\Sigma(x) = 0$ or $\sum_{i=0}^{n} i(|u_i| - |v_i|) = 0$. The latter is not possible, because

$$\begin{aligned}
&\sum_{i=0}^{n} i(|u_i| - |v_i|) = \sum_{i=1}^{n}(|u_i \cdots u_n| - |v_i \cdots v_n|)\\
=&\sum_{i=1}^{n}(|u_0 \cdots u_n| - |u'_i| - (|v_0 \cdots v_n| - |v'_i|)) = \sum_{i=1}^{n}(-|u'_i| + |v'_i|) > 0,
\end{aligned}$$

by Condition N2 in the definition of normal form. Thus every solution $[x]$ is zero-sum. ◀

▶ **Lemma 17.** *Consider the nontrivial equation* (1)*. Let* $s_i = \Sigma(u_0 \cdots u_{i-1})$ *and* $t_i = \Sigma(v_0 \cdots v_{i-1})$ *for all* i*. If the equation has at least two zero-sum solutions, then* $(s_1, \ldots, s_n)$ *is a permutation of* $(t_1, \ldots, t_n)$*.*

Proof. Omitted. ◀

▶ **Lemma 18.** *Let* (1) *be an equation in normal form. Let*

$$h = H(u_0 \cdots u_n) - \max\{\Sigma(u_0 \cdots u_i) \mid i \in \{0, \ldots, n-1\}\}. \tag{3}$$

If the equation has at least three nonempty solutions, then every nonempty solution is of height h*. If the equation has two nonempty solutions, then the shorter one is of height* h *and the longer one of height at least* h*.*

Proof. The idea of the proof is to look at the first occurrences of the highest points on the curves of the left-hand side and the right-hand side of the equation; these must match. If the length of the solution changes, these first occurrences often move with respect to each other so that they no longer match; this puts a limit on the number of solutions under certain conditions. A first occurrence can be either inside a constant part or inside a variable. We will see that if the first occurrences are inside constant parts on both sides, then the solution is empty, if they are inside variables on both sides, then the solution is of height at least h and there can be at most one solution of height more than h, and if the first occurrence is inside a constant part on one side and inside a variable on the other side, then the solution is of height h, and if there is a solution of height more than h, then there can be at most one solution of height h.

For any word w, let $\phi(w)$ be its shortest prefix such that $H(\phi(w)) = H(w)$. For any solution $[x]$, we have

$$\phi(u_0 x u_1 \cdots x u_n) = \phi(v_0 x v_1 \cdots x v_n). \tag{4}$$

Let $s_i = \Sigma(u_0 \cdots u_{i-1})$ and $t_i = \Sigma(v_0 \cdots v_{i-1})$ for all i. Let i and j be such that $\phi(u_0 \cdots u_n) = u_0 \cdots u_{i-1}\phi(u_i)$ and $\phi(v_0 \cdots v_n) = v_0 \cdots v_{j-1}\phi(v_j)$. Because $[\varepsilon]$ is a solution, $\phi(u_0 \cdots u_n) = \phi(v_0 \cdots v_n)$ and thus

$$|u_0 \cdots u_{i-1}| + |\phi(u_i)| = |v_0 \cdots v_{j-1}| + |\phi(v_j)|. \tag{5}$$

By (5) and Condition N3 in the definition of normal form, $i > j$.

Because $[\varepsilon]$ is a solution, $H(u_0 \cdots u_n) = H(v_0 \cdots v_n)$, and by Lemma 17,

$$\max\{\Sigma(u_0 \cdots u_i) \mid i \in \{0, \ldots, n-1\}\} = \max\{\Sigma(v_0 \cdots v_i) \mid i \in \{0, \ldots, n-1\}\},$$

so

$$h = H(v_0 \cdots v_n) - \max\{\Sigma(v_0 \cdots v_i) \mid i \in \{0, \ldots, n-1\}\}.$$

Let k and l be the smallest indices such that $s_k = \max\{s_1, \ldots, s_n\}$ and $t_l = \max\{t_1, \ldots, t_n\}$. Then

$$\phi(u_0 x u_1 \cdots x u_n) = \begin{cases} u_0 x u_1 \cdots u_{i-1} x \phi(u_i) & \text{if } H(x) < h \text{ or if } H(x) = h \text{ and } i < k, \\ u_0 x u_1 \cdots x u_{k-1} \phi(x) & \text{if } H(x) > h \text{ or if } H(x) = h \text{ and } i \geq k, \end{cases}$$

$$\phi(v_0 x v_1 \cdots x v_n) = \begin{cases} v_0 x v_1 \cdots v_{j-1} x \phi(v_j) & \text{if } H(x) < h \text{ or if } H(x) = h \text{ and } j < l, \\ v_0 x v_1 \cdots x v_{l-1} \phi(x) & \text{if } H(x) > h \text{ or if } H(x) = h \text{ and } j \geq l, \end{cases}$$

This means that, for a given x, (4) can take one of four possible forms:

(i) If $H(x) < h$ or if $H(x) = h$, $i < k$ and $j < l$, then

$$u_0xu_1\cdots u_{i-1}x\phi(u_i) = v_0xv_1\cdots v_{j-1}x\phi(v_j)$$

and thus

$$|u_0\cdots u_{i-1}| + |\phi(u_i)| + (i-j)|x| = |v_0\cdots v_{j-1}| + |\phi(v_j)|.$$

Because $i > j$, it follows that this equality can hold for at most one $|x|$, so there is only one possible x in this case, namely, the empty word.

(ii) If $H(x) = h$, $i < k$ and $j \geq l$, then

$$u_0xu_1\cdots u_{i-1}x\phi(u_i) = v_0xv_1\cdots xv_{l-1}\phi(x),$$

but

$$\begin{aligned}&|u_0xu_1\cdots u_{i-1}x\phi(u_i)| = |u_0\cdots u_{i-1}| + |\phi(u_i)| + i|x| = |v_0\cdots v_{j-1}| + |\phi(v_j)| + i|x|\\ &> |v_0\cdots v_{l-1}| + l|x| \geq |v_0xv_1\cdots xv_{l-1}\phi(x)|\end{aligned}$$

by (5) and $i > j \geq l$, a contradiction.

(iii) If $H(x) > h$ or if $H(x) = h$, $i \geq k$ and $j \geq l$, then

$$u_0xu_1\cdots xu_{k-1}\phi(x) = v_0xv_1\cdots xv_{l-1}\phi(x)$$

and thus

$$|u_0\cdots u_{k-1}| + (k-l)|x| = |v_0\cdots v_{l-1}|.$$

By Condition N2 in the definition of normal form, $k > l$. It follows that this equality can hold for at most one $|x|$, so there is only one possible x in this case.

(iv) If $H(x) = h$, $i \geq k$ and $j < l$, then

$$u_0xu_1\cdots xu_{k-1}\phi(x) = v_0xv_1\cdots v_{j-1}x\phi(v_j)$$

and thus

$$|u_0\cdots u_{k-1}| + |\phi(x)| + (k-1-j)|x| = |v_0\cdots v_{j-1}| + |\phi(v_j)|. \tag{6}$$

If x and x' are solution words, then one of them is a prefix of the other, so if they have the same height, then $\phi(x) = \phi(x')$. Therefore, (6) can hold for more than one solution word x of height h only if $k-1-j = 0$. In general, this can happen (for example, if the equation has infinitely many solutions). However, if there exists a solution word of height more than h, then it follows from Case (iii) that $k > l$. Then $j < l < k$, so $k-1 > j$ and there is at most one solution word x of height h. ◀

▶ **Example 19.** Consider the equation

$$(XaXbXaabbabaXbabaabbab, abaabbabaXbabaabbXaXbX)$$

that was mentioned in Example 5. Let $a = 1$ and $b = -1$. The equation has exactly three solutions $[\varepsilon], [ab], [abaabbab]$. All of them are zero-sum, and their heights are $-\infty$, 1, 2, respectively. If we use the notation of the proof of Lemma 18, then $i = 3$, $j = 0$, $k = 2$, $l = 1$, and $h = 1$. We have $\phi(u_i) = \phi(aabbaba) = aa$, $\phi(v_j) = \phi(abaabbaba) = abaa$, $\phi(ab) = a$, and $\phi(abaabbab) = abaa$. Then

$$\phi(xaxbxaabbabaxbabaabbab) = \begin{cases} xaxbxaa & \text{if } x = \varepsilon,\\ xa\phi(x) & \text{if } x = abaabbab \text{ or if } x = ab,\end{cases}$$

$$\phi(abaabbabaxbabaabbxaxbx) = \begin{cases} abaa & \text{if } x = \varepsilon \text{ or if } x = ab,\\ abaabbaba\phi(x) & \text{if } x = abaabbab.\end{cases}$$

6 Some Lemmas

In this section, we state many lemmas about one-variable equations that will be used in the proof of the main result. The proofs are omitted.

A subset Z of Γ^* is called a *code* if the elements of Z do not satisfy any nontrivial relations. In other words, Z is a code if and only if for all $x_1, \ldots, x_m, y_1, \ldots, y_n \in Z$, $x_1 \cdots x_m = y_1 \cdots y_n$ implies $m = n$ and $x_i = y_i$ for all $i \in \{1, \ldots, m\}$. If Z is a code, then Z^* is a free monoid, and if Δ is an alphabet of the same size as Z, then the free monoids Z^* and Δ^* are isomorphic. More information about codes can be found in the book of Berstel, Perrin and Reutenauer [2].

The next lemma can be used to compress an equation into a shorter one. We will use it with two codes Z: The set of all minimal zero-sum words (those zero-sum words which cannot be written as a product of two shorter zero-sum words), and the set of words of a specific length.

▶ **Lemma 20.** *Let E be the equation (1) and let Z be a code. If $u_i, v_i \in Z^*$ for all i, then there exists an alphabet Δ and an isomorphism $h : Z^* \to \Delta^*$, and the equation*

$$(h(u_0)Xh(u_1)\cdots Xh(u_n), h(v_0)Xh(v_1)\cdots Xh(v_n)) \tag{7}$$

has the solution set $\{[h(x)] \mid [x] \in \mathrm{Sol}(E),\ x \in Z^*\}$.

Note that the equation E in Lemma 20 can have solution words that are not in Z^*, so (7) can have less solutions than E.

The next lemma can be used to cut off part of an equation so that all solutions are preserved, except possibly the empty solution (and maybe some additional solutions are added).

▶ **Lemma 21.** *Consider the equation (1). Let $k \in \{0, \ldots, n\}$ and let*

$$d = |v_0 \cdots v_{k-1}| - |u_0 \cdots u_k| \geq 0.$$

If all nonempty solutions of the equation are of length at least d, and if y is the common prefix of length d of all nonempty solution words, then each one of the nonempty solutions is a solution of the equation

$$(u_0 X u_1 \cdots X u_k y, v_0 X v_1 \cdots v_{k-1} X). \tag{8}$$

Using Lemma 21 requires the existence of a suitable index k. The next two lemmas can sometimes be used to find such an index. The proof of Lemma 22 is somewhat similar to the proof of Lemma 18, but simpler.

▶ **Lemma 22.** *Let (1) be an equation in normal form. If it has at least three nonempty solutions, and if there exists $k \in \{1, \ldots, n-1\}$ such that*

$$\Sigma(u_0) = \cdots = \Sigma(u_{k-1}) = 0 \neq \Sigma(u_k),$$

then every nonempty solution is of length more than $|v_0 \cdots v_{k-1}| - |u_0 \cdots u_k|$.

▶ **Lemma 23.** *Let the equation (1) have the solution set $[p^*]$ for some primitive word p. Let $u_0 = v_n = \varepsilon$. Let $j \in \{0, \ldots, n\}$ be the largest index such that the lengths of $u_0, \ldots, u_{j-1}$ and $v_0, \ldots, v_{j-1}$ are divisible by $|p|$. Then $j > 0$ and $|v_0 \cdots v_{j-1}| - |u_0 \cdots u_j| \leq |p|$.*

Lemma 21 does not guarantee that the new, shorter equation would have the empty solution. Sometimes the next lemma can be used to get around this problem.

▶ **Lemma 24.** *If the equation* (1) *has a nonempty solution,* $u_n = ua^m$ *for some* $u \in \Gamma^*$, $a \in \Gamma$ *and* $m \geq 0$, *and* $u_0 \cdots u_{n-1}u$ *is a prefix of* $v_0 \cdots v_n$, *then the equation has the empty solution.*

7 Main results

Now we are ready to prove our main results.

▶ **Theorem 25.** *If a one-variable equation has only finitely many solutions, it has at most three solutions.*

Proof. Assume that there is a counterexample. Then there is one with an empty solution by Lemma 13. Of all equations with the empty solution, at least three nonempty solutions, and only finitely many solutions, let E_1 be a shortest one. We are going to prove a contradiction by showing that there exists a shorter equation with these properties. By Lemma 15, we can assume that E_1 is the equation (1) and it is in normal form. By Lemma 16, each one of its solutions is zero-sum.

The idea of the proof is to cut off part of the equation to get a shorter equation E_2 that has at least three nonempty solutions but only finitely many. Unfortunately, E_2 does not necessarily have the empty solution. We map E_2 with a length-preserving mapping to get an equation E_3 that has at least three nonempty solution and also the empty solution. Unfortunately, E_3 might have infinitely many solutions. We analyze E_3 to find another way to cut off part of E_1 to get an equation E_4, which is then modified to an equation E_5. For E_5, we can finally prove that it has the empty solution and at least three but only finitely many nonempty solutions.

If $\Sigma(u_i) = 0$ for all $i < n$, then $\Sigma(v_i) = 0$ for all $i < n$ by Lemma 17, and then also $\Sigma(u_n) = 0$, because $\Sigma(u_0 \cdots u_n) = \Sigma(v_0 \cdots v_n)$ and $v_n = \varepsilon$. Thus all u_i, v_i are zero-sum, and we can use Lemma 20 with Z the set of all minimal zero-sum words to get a shorter equation with the same number of solutions, one of them empty.

For the rest of the proof, we assume that there exists a minimal $k < n$ such that $\Sigma(u_k) \neq 0$. By symmetry, we can assume that $\Sigma(u_k) > 0$. By Lemmas 22 and 21, we get a shorter equation

$$E_2 : (u_0 X u_1 \cdots X u_k y, v_0 X v_1 \cdots v_{k-1} X)$$

that has at least all the same nonempty solutions as E_1. It might have some other solutions as well, but it cannot have infinitely many solutions, because the intersection of an infinite solution set of a nontrivial one-variable equation and a finite solution set of a one-variable equation is of size at most two by Theorem 6 and Lemma 7. If it has also the empty solution, then we are done, but we do not know yet whether this is the case. We can use Lemma 17 for E_2 to see that $(\Sigma(u_0), \ldots, \Sigma(u_0 \cdots u_{k-1}))$ and $(\Sigma(v_0), \ldots, \Sigma(v_0 \cdots v_{k-1}))$ are permutations of each other. We know that $u_0, \ldots, u_{k-1}$ are zero-sum, so also $v_0, \ldots, v_{k-1}$ are zero-sum.

Let $[x_1]$ be the shortest nonempty solution of E_1. Let $\{a, b\}$ be an alphabet and let g be the morphism that maps the letter $\min(\mathrm{psw}(x_1))$ to b and every other letter to a. Let $f = g \circ \mathrm{psw}$. Then f is length-preserving, and if w is zero-sum, then $f(ww') = f(w)f(w')$. If $[x]$ is a nonempty solution of E_1, then $[f(x)]$ is a solution of the equation

$$E_3 : (f(u_0) X f(u_1) \cdots X f(u_k y), f(v_0) X f(v_1) \cdots f(v_{k-1}) X).$$

We have $f(u_k y) = f(u_k)g(\mathrm{psw}_{\Sigma(u_k)}(y))$. Because $\Sigma(u_k) > 0$ and y is a prefix of x_1, $\min(\mathrm{psw}_{\Sigma(u_k)}(y)) > \min(\mathrm{psw}(x_1))$. Thus $g(\mathrm{psw}_{\Sigma(u_k)}(y)) \in a^*$. Because $u_0 \cdots u_k$ is a prefix of $v_0 \cdots v_{k-1}$, also $f(u_0 \cdots u_k) = f(u_0) \cdots f(u_k)$ is a prefix of $f(v_0 \cdots v_{k-1}) = f(v_0) \cdots f(v_{k-1})$. We can use Lemma 24 with $g(\mathrm{psw}_{\Sigma(u_k)}(y))$ as a^m, so E_3 has the empty solution. If it has only finitely many solutions, then we are done. For the rest of the proof, we assume that it has infinitely many solutions. Then its solution set is $[p^*]$ for some primitive word p. Consequently, the length of every solution word of E_1 is divisible by $|p|$. Because the solution word $f(x_1)$ of E_3 contains the letter b, also p must contain b. This means that p cannot be a suffix of $g(\mathrm{psw}_{\Sigma(u_k)}(y)) \in a^*$, so $|p| > |y|$.

We can use Lemma 23 for E_3 to find an index j such that the lengths of $u_0, \ldots, u_{j-1}$ and $v_0, \ldots, v_{j-1}$ are divisible by $|p|$ and, if $j < k$, $|v_0 \cdots v_{j-1}| - |u_0 \cdots u_j| \leq |p|$ (remember that f is length-preserving). By letting $z = y$ if $j = k$, or by using Lemma 21 with j as k for E_1 otherwise, we get an equation

$$E_4 : (u_0 X u_1 \cdots X u_j z, v_0 X v_1 \cdots v_{j-1} X)$$

that has at least all the same nonempty solutions as E_1. In both cases, $|z| \leq |p|$. Like in the case of E_2, we see that E_4 cannot have infinitely many solutions. The lengths of all the constant words in E_4 are divisible by $|p|$, and so are the lengths of at least three nonempty solutions (the solutions of E_1). We can use Lemma 20 with $Z = \Gamma^{|p|}$ for E_4. If h is the morphism of Lemma 20, then we get the equation

$$E_5 : (h(u_0) X h(u_1) \cdots X h(u_j z), h(v_0) X h(v_1) \cdots h(v_{j-1}) X).$$

It has at least three nonempty solutions, but only finitely many. Because $|z| \leq |p|$, $h(u_j z) = h(u)c$, where u is a prefix of u_j and c is a letter. Because $u_0 \cdots u_j$ is a prefix of $v_0 \cdots v_{j-1}$, also $h(u_0 \cdots u_{j-1} u) = h(u_0) \cdots h(u_{j-1}) h(u)$ is a prefix of $h(v_0 \cdots v_{k-1}) = h(v_0) \cdots h(v_{k-1})$. We can use Lemma 24 with c as a and $m = 1$, so E_5 has the empty solution. This contradicts the minimality of E_1. ◀

▶ **Theorem 26.** *If a system of constant-free three-variable equations is independent and has a nonperiodic solution, then it has at most 17 equations.*

Proof. Follows from Theorem 25 and Theorem 8. ◀

8 Conclusion

We have proved that the maximal size of a finite solution set of a one-variable word equation is three, and that the maximal size of an independent system of constant-free three-variable equations with a nonperiodic solution is somewhere between two and 17.

Improving the bound 17 is an obvious open problem. A possible approach would be to improve the results in [16].

Another open problem is proving similar bounds for more than three variables. The result in [16] is based on a characterization of three-generator subsemigroups of a free semigroup by Budkina and Markov [3], or alternatively a similar result by Spehner [19, 20]. This means that it is very specific to the three-variable case, and analyzing the general case would require an entirely different approach.

Finally, characterizing possible solution sets of one-variable equations would be interesting. The possible infinite solution sets are given by Theorem 6, and every singleton set is possible, but for sets of size two or three the question is open.

References

1 M. H. Albert and J. Lawrence. A proof of Ehrenfeucht's conjecture. *Theoret. Comput. Sci.*, 41(1):121–123, 1985. doi:10.1016/0304-3975(85)90066-0.

2 Jean Berstel, Dominique Perrin, and Christophe Reutenauer. *Codes and Automata*. Cambridge University Press, 2010.

3 L. G. Budkina and Al. A. Markov. F-semigroups with three generators. *Mat. Zametki*, 14:267–277, 1973.

4 Karel Culik, II and Juhani Karhumäki. Systems of equations over a free monoid and Ehrenfeucht's conjecture. *Discrete Math.*, 43(2–3):139–153, 1983. doi:10.1016/0012-365X(83)90152-8.

5 Elena Czeizler and Juhani Karhumäki. On non-periodic solutions of independent systems of word equations over three unknowns. *Internat. J. Found. Comput. Sci.*, 18(4):873–897, 2007. doi:10.1142/S0129054107005030.

6 Elena Czeizler and Wojciech Plandowski. On systems of word equations over three unknowns with at most six occurrences of one of the unknowns. *Theoret. Comput. Sci.*, 410(30–32):2889–2909, 2009. doi:10.1016/j.tcs.2009.01.023.

7 Robert Dąbrowski and Wojciech Plandowski. On word equations in one variable. *Algorithmica*, 60(4):819–828, 2011. doi:10.1007/s00453-009-9375-3.

8 S. Eyono Obono, P. Goralčík, and M. Maksimenko. Efficient solving of the word equations in one variable. In *Proceedings of the 19th MFCS*, volume 841 of *LNCS*, pages 336–341. Springer, 1994. doi:10.1007/3-540-58338-6_80.

9 V. S. Guba. Equivalence of infinite systems of equations in free groups and semigroups to finite subsystems. *Mat. Zametki*, 40(3):321–324, 1986. doi:10.1007/BF01142470.

10 Tero Harju and Dirk Nowotka. On the independence of equations in three variables. *Theoret. Comput. Sci.*, 307(1):139–172, 2003. doi:10.1016/S0304-3975(03)00098-7.

11 Štěpán Holub and Jan Žemlička. Algebraic properties of word equations. *J. Algebra*, 434:283–301, 2015. doi:10.1016/j.jalgebra.2015.03.021.

12 Artur Jeż. One-variable word equations in linear time. *Algorithmica*, 74(1):1–48, 2016. doi:10.1007/s00453-014-9931-3.

13 Juhani Karhumäki and Wojciech Plandowski. On the defect effect of many identities in free semigroups. In Gheorghe Paun, editor, *Mathematical aspects of natural and formal languages*, pages 225–232. World Scientific, 1994.

14 Juhani Karhumäki and Aleksi Saarela. On maximal chains of systems of word equations. *Proc. Steklov Inst. Math.*, 274:116–123, 2011. doi:10.1134/S0081543811060083.

15 Markku Laine and Wojciech Plandowski. Word equations with one unknown. *Internat. J. Found. Comput. Sci.*, 22(2):345–375, 2011. doi:10.1142/S0129054111008088.

16 Dirk Nowotka and Aleksi Saarela. One-variable word equations and three-variable constant-free word equations. *Internat. J. Found. Comput. Sci.*, To appear.

17 Aleksi Saarela. Systems of word equations, polynomials and linear algebra: A new approach. *European J. Combin.*, 47:1–14, 2015. doi:10.1016/j.ejc.2015.01.005.

18 Aleksi Saarela. Word equations where a power equals a product of powers. In *Proceedings of the 34th STACS*, volume 66 of *LIPIcs*, pages 55:1–55:9. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.55.

19 Jean-Claude Spehner. *Quelques problémes d'extension, de conjugaison et de présentation des sous-monoïdes d'un monoïde libre*. PhD thesis, Univ. Paris, 1976.

20 Jean-Claude Spehner. Les systemes entiers d'équations sur un alphabet de 3 variables. In *Semigroups*, pages 342–357, 1986.

Separating Without Any Ambiguity

Thomas Place
LaBRI, University of Bordeaux and IUF, France

Marc Zeitoun
LaBRI, University of Bordeaux, France

Abstract

We investigate a standard operator on classes of languages: unambiguous polynomial closure. We show that if $\mathcal{C}$ is a class of regular languages having some mild properties, the membership problem for its unambiguous polynomial closure $UPol(\mathcal{C})$ reduces to the same problem for $\mathcal{C}$. We give a new, self-contained and elementary proof of this result. We also show that unambiguous polynomial closure coincides with alternating left and right deterministic closure. Finally, if additionally $\mathcal{C}$ is finite, we show that the separation and covering problems are decidable for $UPol(\mathcal{C})$.

2012 ACM Subject Classification Theory of computation → Regular languages

Keywords and phrases Regular languages, separation problem, decidable characterizations

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.137

Related Version A full version of the paper is available at `https://hal.archives-ouvertes.fr/hal-01798847`.

Funding Both authors acknowledge support from the DeLTA project (ANR-16-CE40-0007).

1 Introduction

Most of the interesting classes of regular languages are built using a restricted set of operators. From a class $\mathcal{C}$, one may consider its Boolean closure $Bool(\mathcal{C})$, its polynomial closure $Pol(\mathcal{C})$, and deterministic variants thereof, which yield usually a more elaborate class than $\mathcal{C}$. It is therefore desirable to investigate the operators themselves rather than individual classes.

The *polynomial closure* $Pol(\mathcal{C})$ of a class $\mathcal{C}$ is its closure under union and marked concatenation (a *marked concatenation* of K and L is a language of the form KaL for a letter a). Together with the Boolean closure, it is used to define concatenation hierarchies: starting from a given class (*level 0* in the hierarchy), level $n + \frac{1}{2}$ is the polynomial closure of level n, and level $n+1$ is the Boolean closure of level $n+\frac{1}{2}$. The importance of these hierarchies stems from the fact that they are the combinatorial counterpart of quantifier alternation hierarchies in logic, which count the number of $\forall/\exists$ alternations needed to define a language [29, 23].

The main question when investigating a class of languages is the *membership* problem: can we decide whether an input language belongs to the class? Despite decades of research on concatenation hierarchies, one knows little about it. The state of the art is that when level 0 is finite and has some mild properties, membership is decidable for levels $\frac{1}{2}$, 1, $\frac{3}{2}$, and $\frac{5}{2}$ [21, 19, 16, 17, 23]. These results encompass those that were obtained previously [3, 2, 26, 14, 15] and even go beyond by investigating the *separation problem*, a generalization of membership. This problem for a class $\mathcal{C}$ takes *two arbitrary regular* languages as input (unlike membership, which takes a single one). It asks whether there exists a third language from $\mathcal{C}$, containing the first and disjoint from the second. Membership is the special case of separation when the input consists of a language and its complement. Although more difficult than membership, separation is also more rewarding. This is witnessed by a transfer

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 137; pp. 137:1–137:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

theorem [19, 23]: membership for $Pol(\mathcal{C})$ reduces to separation for $\mathcal{C}$. The above results on membership come from this theorem and the fact that separation is decidable for $Pol(\mathcal{C})$, $BPol(\mathcal{C}) \stackrel{\text{def}}{=} Bool(Pol(\mathcal{C}))$ and $Pol(BPol(\mathcal{C}))$ when $\mathcal{C}$ is finite with some mild properties. See [12, 23] for detailed surveys on concatenation hierarchies.

Unambiguous closure. Deterministic variants of the polynomial closure are also important. The most classical example is the unambiguous closure, where marked concatenations are required to be unambiguous. A marked concatenation KaL is *unambiguous* if every word w of KaL has a *unique* factorization $w = w'aw''$ with $w' \in K$ and $w'' \in L$. The *unambiguous closure* $UPol(\mathcal{C})$ of $\mathcal{C}$ is the closure of $\mathcal{C}$ under *disjoint* union and *unambiguous* concatenation. Note that it is not clear on the definition whether $UPol(\mathcal{C})$ is a Boolean algebra, even when $\mathcal{C}$ is.

A prominent example of a class built using unambiguous concatenation is that of *unambiguous languages* [25]. It is the unambiguous polynomial closure of the Boolean algebra generated by languages of the form A^*aA^*, where A is the working alphabet. Its robustness makes it one of the most investigated classes: it enjoys a number of equivalent definitions [27, 5, 7, 6, 28].

State of the art. The class $UPol(\mathcal{C})$ was described in algebraic terms in [13], following earlier work on deterministic products by Pin [9]. Note however that [9] starts from an alternate definition that assumes closure under Boolean operations already. Both papers use elaborate mathematical tools (categories, bilateral kernel, relational morphisms) as well as black boxes (results by Schützenberger [25] in [9] and a result of Rhodes [24] in [13]). Unambiguous polynomial closure also appears in concatenation hierarchies as *intermediate levels*: Pin and Weil [14, 15] have proved that $UPol(\mathcal{C}) = Pol(\mathcal{C}) \cap co\text{-}Pol(\mathcal{C})$, where $co\text{-}Pol(\mathcal{C})$ is the class consisting of all complements of languages in $Pol(\mathcal{C})$. Finally, a reduction from $UPol(\mathcal{C})$-membership to $\mathcal{C}$-membership was obtained in [1]. This proof is indirect: it is based on the nontrivial equality $UPol(\mathcal{C}) = Pol(\mathcal{C}) \cap co\text{-}Pol(\mathcal{C})$, which itself depends on the algebraic characterizations of $UPol(\mathcal{C})$ and $Pol(\mathcal{C})$ obtained in [13, 14, 15, 4].

Contributions. Unambiguous polynomial closure was not yet investigated with respect to separation, aside from the particular case of unambiguous languages [18]. This is the starting point of this paper: we look for a generic separation result applying to $UPol(\mathcal{C})$, similar to the ones obtained for $Pol(\mathcal{C})$ and $BPol(\mathcal{C})$ in [21]. This paper presents such a result: our main theorem states that when $\mathcal{C}$ is finite and satisfies some mild hypotheses, separation for $UPol(\mathcal{C})$ is decidable. However, as it is usual with separation, we also obtain several extra results as a byproduct of our work, improving our understanding of the $UPol$ operator:

- We had to rethink the way membership is classically handled for $UPol(\mathcal{C})$ in order to lift the techniques to separation. This yields a completely new, self-contained and elementary proof that under some natural hypothesis on $\mathcal{C}$, membership for $UPol(\mathcal{C})$ reduces to membership for $\mathcal{C}$. This proof also precisely pinpoints why this result holds for $UPol(\mathcal{C})$ but not $Pol(\mathcal{C})$. More precisely, we show that the languages from $\mathcal{C}$ needed to construct an $UPol(\mathcal{C})$ expression for a language L are all recognized by any recognizer of L.
- We obtain a new proof that $UPol(\mathcal{C})$ is a quotienting Boolean algebra when $\mathcal{C}$ is one.
- We obtain a new proof that $UPol(\mathcal{C}) = Pol(\mathcal{C}) \cap co\text{-}Pol(\mathcal{C})$ using our results on $Pol(\mathcal{C})$ [23].
- We obtain a previously unknown characterization of $UPol(\mathcal{C})$ in terms of alternating left and right deterministic concatenations, which are restricted forms of unambiguous concatenation. A marked concatenation KaL is *left (resp. right) deterministic* when $KaA^* \cap K = \emptyset$ (resp. $A^*aL \cap L = \emptyset$). We prove that $UPol(\mathcal{C})$ coincides with $ADet(\mathcal{C})$, the closure of $\mathcal{C}$ under disjoint union and left and right deterministic concatenation.

Organization of the paper. Section 2 sets up the notation and the terminology. Section 3 presents a solution of the membership problem for $UPol(\mathcal{C})$ when $\mathcal{C}$ has mild closure properties. This result also yields the above byproducts. Finally, in Section 4, we present an algorithm for solving separation for $UPol(\mathcal{C})$ when $\mathcal{C}$ is additionally finite.

2 Preliminaries

Words and languages. For the whole paper, we fix an arbitrary finite alphabet A. We denote by A^* the set of all finite words over A, and by $\varepsilon \in A^*$ the empty word. Given two words $u, v \in A^*$, we write uv for their concatenation. A *language (over A)* is a subset of A^*. Abusing terminology, we denote by u the singleton language $\{u\}$. It is standard to extend concatenation to languages: given $K, L \subseteq A^*$, we write $KL = \{uv \mid u \in K \text{ and } v \in L\}$. Moreover, we also consider marked concatenation, which is less standard. Given $K, L \subseteq A^*$, *a marked concatenation* of K with L is a language of the form KaL, for some $a \in A$.

A class of languages $\mathcal{C}$ is a set of languages. We say that $\mathcal{C}$ is a *lattice* when $\emptyset \in \mathcal{C}$, $A^* \in \mathcal{C}$ and $\mathcal{C}$ is closed under finite union and finite intersection: for any $K, L \in \mathcal{C}$, we have $K \cup L \in \mathcal{C}$ and $K \cap L \in \mathcal{C}$. Moreover, a *Boolean algebra* is a lattice $\mathcal{C}$ which is additionally closed under complement: for any $L \in \mathcal{C}$, we have $A^* \setminus L \in \mathcal{C}$. Finally, a class $\mathcal{C}$ is *quotienting* if it is closed under quotients. That is, for any $L \in \mathcal{C}$ and any word $u \in A^*$, the following properties hold:

$$u^{-1}L \stackrel{\text{def}}{=} \{w \in A^* \mid uw \in L\} \quad \text{and} \quad Lu^{-1} \stackrel{\text{def}}{=} \{w \in A^* \mid wu \in L\} \quad \text{both belong to } \mathcal{C}.$$

All classes that we consider are quotienting Boolean algebras of regular languages.

Regular languages. These are the languages that can be equivalently defined by nondeterministic finite automata, finite monoids or monadic second-order logic. In the paper, we work with the definition by monoids, which we recall now.

A *monoid* is a set M endowed with an associative multiplication $(s, t) \mapsto s \cdot t$ (also denoted by st) having a neutral element 1_M, *i.e.*, such that $1_M \cdot s = s \cdot 1_M = s$ for every $s \in M$. An *idempotent* of a monoid M is an element $e \in M$ such that $ee = e$. It is folklore that for any *finite* monoid M, there exists a natural number $\omega(M)$ (denoted by ω when M is understood) such that for any $s \in M$, the element s^ω is an idempotent.

Our proofs make use of the Green relations [8], which are defined on monoids (we use them as induction parameters). We briefly recall them. Given a monoid M and $s, t \in M$,

$$\begin{array}{ll} s \leqslant_{\mathcal{J}} t & \text{when there exist } x, y \in M \text{ such that } s = xty, \\ s \leqslant_{\mathcal{L}} t & \text{when there exists } x \in M \text{ such that } s = xt, \\ s \leqslant_{\mathcal{R}} t & \text{when there exists } y \in M \text{ such that } s = ty. \end{array}$$

Clearly, $\leqslant_{\mathcal{J}}$, $\leqslant_{\mathcal{L}}$ and $\leqslant_{\mathcal{R}}$ are preorders (*i.e.*, they are reflexive and transitive). We write $<_{\mathcal{J}}$, $<_{\mathcal{L}}$ and $<_{\mathcal{R}}$ for their strict variants (for example, $s <_{\mathcal{J}} t$ when $s \leqslant_{\mathcal{J}} t$ but $t \not\leqslant_{\mathcal{J}} s$). Finally, we write $\mathcal{J}$, $\mathcal{L}$ and $\mathcal{R}$ for the corresponding equivalence relations (for example, $s \mathcal{J} t$ when $s \leqslant_{\mathcal{J}} t$ and $t \leqslant_{\mathcal{J}} s$). There are many technical results about Green relations. We shall only need the following simple lemma which applies to *finite* monoids (see [11]).

▶ **Lemma 1.** *Consider a finite monoid M and $s, t \in M$ such that $s \mathcal{J} t$. Then, $s \leqslant_{\mathcal{R}} t$ implies $s \mathcal{R} t$. Symmetrically, $s \leqslant_{\mathcal{L}} t$ implies $s \mathcal{L} t$.*

Observe that A^* is a monoid whose multiplication is concatenation (the neutral element is ε). Thus, we may consider monoid morphisms $\alpha : A^* \to M$ where M is an arbitrary monoid. Given such a morphism and some language $L \subseteq A^*$, we say that L is *recognized* by α when there exists a set $F \subseteq M$ such that $L = \alpha^{-1}(F)$.

Given any language L, there exists a canonical morphism which recognizes it. Let us briefly recall its definition. One may associate to L an equivalence $\equiv_L$ over A^*: the *syntactic congruence of* L. Given $u, v \in A^*$, $u \equiv_L v$ if and only if $xuy \in L \Leftrightarrow xvy \in L$ for any $x, y \in A^*$. It is known and simple to verify that "$\equiv_L$" is a congruence on A^*. Thus, the set of equivalence classes $M_L = A^*/\equiv_L$ is a monoid and the map $\alpha_L : A^* \to M_L$ sending any word to its equivalence class is a morphism recognizing L, called the *syntactic morphism of* L. Finally, it is known that L is regular if and only if M_L is finite (*i.e.*, $\equiv_L$ has finite index): this is Myhill-Nerode theorem. In that case, one may compute the syntactic morphism $\alpha_L : A^* \to M_L$ from any representation of L (such as a finite automaton).

Decision problems. The two problems that we consider in the paper are both parametrized by an arbitrary class of languages $\mathcal{C}$: they serve as mathematical tools for analyzing $\mathcal{C}$. The *$\mathcal{C}$-membership problem* is the simplest one. It takes as input a single regular language L and asks whether $L \in \mathcal{C}$. The second one, *$\mathcal{C}$-separation*, is more general: it takes **two** regular languages L_1, L_2 as input and asks whether L_1 is $\mathcal{C}$-separable from L_2, that is, whether there exists $K \in \mathcal{C}$ such that $L_1 \subseteq K$ and $L_2 \cap K = \emptyset$. The language K is called a *separator* of L_1 and L_2. Note that $\mathcal{C}$-membership is easily reduced to $\mathcal{C}$-separation: given any regular language L, we have $L \in \mathcal{C}$ if and only if L is $\mathcal{C}$-separable from $A^* \setminus L$ (which is also regular).

3 Unambiguous polynomial closure

In this section, we define the unambiguous polynomial closure operation, which is the main focus of the paper. Furthermore, we investigate the associated membership problem.

3.1 Definition

Given two languages $H, L \subseteq A^*$, we say that their concatenation HL is *unambiguous* when any word $w \in HL$ admits a *unique* decomposition witnessing this membership: for any $u, u' \in H$ and $v, v' \in L$, if $w = uv = u'v'$, then $u = u'$ and $v = v'$. More generally, we say that a product of n languages $L_1 \cdots L_n$ is *unambiguous* when any word $w \in L_1 \cdots L_n$ admits a *unique* decomposition witnessing this membership. Note that unambiguous *marked* concatenations are well-defined: HaL is a product of three languages, namely H, $\{a\}$ and L.

▶ Remark. Clearly, not all products are unambiguous. For example, A^*aA^* is ambiguous: $aa \in A^*aA^*$ admits two decompositions witnessing this membership (εaa and $aa\varepsilon$).

▶ Remark. Being unambiguous is a *semantic* property: whether HL is unambiguous may not be apparent on the definitions of H and L. Moreover, this depends on the *product* HL and not only on the resulting language $K = HL$. It may happen that two products represent the same language but one is unambiguous while the other is not. For example, A^*aA^* is ambiguous while $(A \setminus \{a\})^*aA^*$ (which represents the same language) is unambiguous.

In the paper, we shall only need two special kinds of unambiguous products, which we now present. Let $K, L \subseteq A^*$ and $a \in A$. We say that the marked concatenation KaL,

- is *left deterministic* when $K \cap KaA^* = \emptyset$,
- is *right deterministic* when $L \cap A^*aL = \emptyset$.

▶ **Fact 2.** *Any left or right deterministic marked concatenation is unambiguous.*

We use these definitions to introduce three standard operations on classes of languages. Consider an arbitrary class $\mathcal{C}$.

- The *polynomial closure* of $\mathcal{C}$, denoted by $Pol(\mathcal{C})$, is the smallest class containing $\mathcal{C}$ and closed under marked concatenation and union: for any $H, L \in Pol(\mathcal{C})$ and $a \in A$, we have $HaL \in Pol(\mathcal{C})$ and $H \cup L \in Pol(\mathcal{C})$. Furthermore, we denote by *co-Pol*$(\mathcal{C})$ the class containing all complements of languages in $Pol(\mathcal{C})$: $L \in$ *co-Pol*$(\mathcal{C})$ when $A^* \setminus L \in Pol(\mathcal{C})$.
- The *unambiguous polynomial closure* of $\mathcal{C}$, denoted by $UPol(\mathcal{C})$, is the smallest class containing $\mathcal{C}$ and closed under *unambiguous* marked concatenation and *disjoint* union. That is, for any $H, L \in UPol(\mathcal{C})$ and $a \in A$, if HaL is unambiguous, then $HaL \in UPol(\mathcal{C})$ and if $H \cap L = \emptyset$, then $H \uplus L \in UPol(\mathcal{C})$. Here, we denote union by "$\uplus$" to underline the fact that H and L are disjoint (we use this convention in the whole paper).
- The *alternating deterministic closure* of $\mathcal{C}$, denoted by $ADet(\mathcal{C})$, is the smallest class containing $\mathcal{C}$ and closed under *deterministic* marked concatenation and *disjoint* union. That is, for any $H, L \in ADet(\mathcal{C})$ and $a \in A$, if HaL is either left or right deterministic, then $HaL \in ADet(\mathcal{C})$ and if $H \cap L = \emptyset$, then $H \uplus L \in ADet(\mathcal{C})$.

It is immediate by definition and Fact 2 that we have $\mathcal{C} \subseteq ADet(\mathcal{C}) \subseteq UPol(\mathcal{C}) \subseteq Pol(\mathcal{C})$. In general the inclusion $UPol(\mathcal{C}) \subseteq Pol(\mathcal{C})$ is strict. On the other hand, we shall prove that when $\mathcal{C}$ is a quotienting Boolean algebra, $ADet(\mathcal{C}) = UPol(\mathcal{C})$.

It is not immediate that $Pol(\mathcal{C})$, $UPol(\mathcal{C})$ and $ADet(\mathcal{C})$ have robust closure properties beyond those explicitly stated in the definitions. However, it turns out that when $\mathcal{C}$ satisfies robust properties itself, this is the case for these three classes as well. It was shown by Arfi [3] that when $\mathcal{C}$ is a quotienting Boolean algebra of regular languages, then $Pol(\mathcal{C})$ is a quotienting lattice. Pin [10] extended the result for the case when $\mathcal{C}$ is a quotienting lattice. Here, we are mostly interested in $UPol(\mathcal{C})$. We prove the following theorem which combines and extends several results by Pin, Straubing, Thérien and Weil [13, 15].

▶ **Theorem 3.** *Let $\mathcal{C}$ be a quotienting Boolean algebra of regular languages. Then, $UPol(\mathcal{C})$ is a quotienting Boolean algebra as well. Moreover, $UPol(\mathcal{C}) = ADet(\mathcal{C}) = Pol(\mathcal{C}) \cap$ co-$Pol(\mathcal{C})$.*

That $UPol(\mathcal{C})$ is a quotienting Boolean algebra of regular languages is due to Pin, Straubing and Thérien [13]. The correspondence between $UPol(\mathcal{C})$ and $Pol(\mathcal{C}) \cap$ *co-Pol*$(\mathcal{C})$ is due to Pin and Weil [15]. The correspondence between $UPol(\mathcal{C})$ and $ADet(\mathcal{C})$ is a new result, to the best of our knowledge. Let us point out that the original proofs of these results require a stronger hypothesis on $\mathcal{C}$, which needs additionally to be closed under inverse morphic image. Moreover, these proofs require to introduce and manipulate a lot of algebraic machinery. This is because they are based on a generic algebraic characterization of $UPol(\mathcal{C})$.

While we use a similar approach (*i.e.*, we prove a generic algebraic characterization of $UPol(\mathcal{C})$), our argument is much more elementary. The only algebraic notion that we need is the syntactic morphism of a regular language.

3.2 Algebraic characterization

We now present a generic algebraic characterization of $UPol(\mathcal{C})$. It holds provided that $\mathcal{C}$ is a quotienting Boolean algebra of regular languages. It implies Theorem 3, but also that $UPol(\mathcal{C})$-membership reduces to $\mathcal{C}$-membership.

The characterization is parameterized by two relations that we define now. Let $\mathcal{C}$ be some class of languages. Consider a finite monoid M and a *surjective* morphism $\alpha : A^* \to M$ (such as the syntactic morphism of some language). Given a pair $(s, t) \in M \times M$,

- (s, t) is a *$\mathcal{C}$-pair* (for α) when **no** language of $\mathcal{C}$ can separate $\alpha^{-1}(s)$ from $\alpha^{-1}(t)$.
- (s, t) is a *weak $\mathcal{C}$-pair* (for α) when **no** language of $\mathcal{C}$ **recognized by α** can separate $\alpha^{-1}(s)$ from $\alpha^{-1}(t)$.

Note that any $\mathcal{C}$-pair is also a weak $\mathcal{C}$-pair (the converse is not true in general). By definition, we are able to compute all $\mathcal{C}$-pairs as soon as we have an algorithm for $\mathcal{C}$-separation. On the other hand, computing all weak $\mathcal{C}$-pairs boils down to deciding $\mathcal{C}$-membership, as it suffices to check which languages recognized by α (the potential separators) belong to $\mathcal{C}$.

▶ Remark. An equivalent definition of the weak $\mathcal{C}$-pairs is to introduce them as the transitive closure of the $\mathcal{C}$-pairs. We prove this in the full version. In fact, when $\mathcal{C}$ is a quotienting Boolean algebra, the weak $\mathcal{C}$-pair relation is a congruence whose equivalence classes correspond exactly to the languages recognized by α and belonging to $\mathcal{C}$.

We may now state the following characterization of $UPol(\mathcal{C})$.

▶ **Theorem 4.** *Let $\mathcal{C}$ be a quotienting Boolean algebra of regular languages. Consider a regular language L and let $\alpha : A^* \to M$ be its syntactic morphism. The following are equivalent:*

1. $L \in UPol(\mathcal{C})$.
2. $L \in ADet(\mathcal{C})$.
3. $L \in Pol(\mathcal{C}) \cap co\text{-}Pol(\mathcal{C})$.
4. *For all $\mathcal{C}$-pairs $(s,t) \in M^2$, we have $s^{\omega+1} = s^\omega t s^\omega$.*
5. *For all weak $\mathcal{C}$-pairs $(s,t) \in M^2$, we have $s^{\omega+1} = s^\omega t s^\omega$.*

Theorem 3 is a simple corollary of Theorem 4 (it is standard that any class satisfying a property such as Item (4) in the theorem is a quotienting Boolean algebra). Another consequence is that if $\mathcal{C}$ is a quotienting Boolean algebra of regular languages, $UPol(\mathcal{C})$-membership reduces to the same problem for $\mathcal{C}$. Indeed, given as input a regular language L, one may compute its syntactic morphism α. By Theorem 4, deciding whether $L \in UPol(\mathcal{C})$ amounts to checking whether α satisfies Item (5). This is possible provided that we have all weak $\mathcal{C}$-pairs for α in hand. In turn, an algorithm for $\mathcal{C}$-membership immediately yields an algorithm for computing them all. Altogether, we obtain the following corollary.

▶ **Corollary 5.** *Let $\mathcal{C}$ be a quotienting Boolean algebra of regular languages and assume that $\mathcal{C}$-membership is decidable. Then $UPol(\mathcal{C})$-membership is decidable as well.*

We now focus on proving Theorem 4. A first point is that we do not show the equivalence (3) $\Leftrightarrow$ (4): it is a simple corollary of the generic characterization of $Pol(\mathcal{C})$ which is not our main focus in the paper (a full proof is available in [23]). Here, we concentrate on proving the implications (1) $\Rightarrow$ (4) $\Rightarrow$ (5) $\Rightarrow$ (2) $\Rightarrow$ (1). The implication (2) $\Rightarrow$ (1) ($ADet(\mathcal{C}) \subseteq UPol(\mathcal{C})$) is immediate. Even though the presentation is different, the equivalence (4) $\Leftrightarrow$ (5) is a result of [1] (which investigates $Pol(\mathcal{C}) \cap co\text{-}Pol(\mathcal{C})$) and is based on algebraic manipulations. We prove this equivalence as well as the implication (1) $\Rightarrow$ (4) in the full version, to focus on (5) $\Rightarrow$ (2), which is the most interesting implication: when a language satisfies (5), we show that it belongs to $ADet(\mathcal{C})$.

We fix a quotienting Boolean algebra of regular languages $\mathcal{C}$ for the proof. Consider an arbitrary surjective morphism $\alpha : A^* \to M$ satisfying Item (5) in Theorem 4. We show that any language recognized by α belongs to $ADet(\mathcal{C})$. We start with a preliminary lemma.

▶ **Lemma 6.** *There exists a finite monoid N and a surjective morphism $\beta : M \to N$ which satisfies the following properties:*

- *For any $s,t \in M$, (s,t) is a weak $\mathcal{C}$-pair if and only if $\beta(s) = \beta(t)$.*
- *Any language recognized by the composition $\gamma = \beta \circ \alpha : A^* \to N$ belongs to $\mathcal{C}$.*

Lemma 6 is obtained by proving that the weak $\mathcal{C}$-pair relation is a congruence on M and that for any equivalence class $F \subseteq M$, we have $\alpha^{-1}(F) \in \mathcal{C}$. It then suffices to define N as the quotient of M by this congruence. The proof is presented in the full version of the paper.

Let us come back to the main proof. Let $\beta : M \to N$ and the composition $\gamma = \beta \circ \alpha$ be defined as in Lemma 6. Given any $r_1, r_2, s \in M$ and any $x \in N$, we define:

$$L_s^x[r_1, r_2] = \{w \in \gamma^{-1}(x) \mid r_1\alpha(w)r_2 = s\}.$$

The purpose of introducing $L_s^x[r_1, r_2]$ is that it provides induction parameters s, r_1, r_2 and it coincides with $\alpha^{-1}(s)$ when $x = \beta(s)$, $r_1 = r_2 = 1_M$. Our goal is to show that it is in $ADet(\mathcal{C})$.

▶ **Proposition 7.** *Let $r_1, r_2, s \in M$ and $x \in N$. Then, $L_s^x[r_1, r_2] \in ADet(\mathcal{C})$.*

Before proving this proposition, let us use it to finish the main proof. By definition, a language recognized by α is a disjoint union of sets $\alpha^{-1}(s)$ for $s \in M$. Therefore, it suffices to prove that $\alpha^{-1}(s) \in ADet(\mathcal{C})$ for any $s \in M$. Let $x = \beta(s)$. Clearly, $L_s^{\beta(s)}[1_M, 1_M] = \alpha^{-1}(s)$. Thus, Proposition 7 yields that $\alpha^{-1}(s) \in ADet(\mathcal{C})$, finishing the proof.

It remains to prove Proposition 7. We let $r_1, r_2, s \in M$ and $x \in N$. Our objective is to show that $L_s^x[r_1, r_2] \in ADet(\mathcal{C})$. Observe that we may assume without loss of generality that $\beta(s) = \beta(r_1)x\beta(r_2)$. Otherwise, $L_s^x[r_1, r_2] = \emptyset \in ADet(\mathcal{C})$ by definition and the result is immediate. The proof is an induction on the three following parameters listed by order of importance (the three of them depend on Green's relations in both M and N):

1. The *rank of* $\beta(s)$ which is the number of elements $y \in N$ such that $\beta(s) \leqslant_{\mathcal{J}} y$.
2. The *right index of* r_1 which is the number of elements $t \in M$ such that $t \leqslant_{\mathcal{R}} r_1$.
3. The *left index of* r_2 which is the number of elements $t \in M$ such that $t \leqslant_{\mathcal{L}} r_2$.

We consider three cases depending on the following properties of s, r_1, r_2 and x.

- We say that x is *smooth* when $x \mathrel{\mathcal{J}} \beta(s)$.
- We say that r_1 is *right stable* when there exists $t \in M$ such that $\beta(t) \mathrel{\mathcal{R}} x$ and $r_1 t \mathrel{\mathcal{R}} r_1$.
- We say that r_2 is *left stable* when there exists $t \in M$ such that $\beta(t) \mathrel{\mathcal{L}} x$ and $tr_2 \mathrel{\mathcal{L}} r_2$.

In the base case, we assume that all three properties hold. Otherwise, we consider two inductive cases. First, we assume that x is not smooth. Then, we assume that either r_1 is not right stable or r_2 is not left stable.

Base case. Assume that x is smooth and that r_1, r_2 are respectively right and left stable. We use this hypothesis to prove the following lemma.

▶ **Lemma 8.** *For any $u, v \in \gamma^{-1}(x)$, we have $r_1\alpha(u)r_2 = r_1\alpha(v)r_2$.*

Observe that Lemma 8 concludes the proof. Indeed, by definition of $L_s^x[r_1, r_2]$, it implies that either $L_s^x[r_1, r_2] = \gamma^{-1}(x)$ (when $r_1\alpha(w)r_2 = s$ for all $w \in \gamma^{-1}(x)$) or $L_s^x[r_1, r_2] = \emptyset$ (when $r_1\alpha(w)r_2 \neq s$ for all $w \in \gamma^{-1}(x)$). Since both of these languages belong to $\mathcal{C} \subseteq ADet(\mathcal{C})$ by Lemma 6, Proposition 7 follows. It remains to prove Lemma 8 to conclude the base case. The argument relies on the following fact (this is where we use our hypothesis on r_1 and r_2).

▶ **Fact 9.** *When Item (5) in Theorem 4 holds, the two following properties hold as well:*

- *For all $t \in M$ such that $\beta(t) \mathrel{\mathcal{R}} x$, we have $r_1 t \mathrel{\mathcal{R}} r_1$.*
- *For all $t \in M$ such that $\beta(t) \mathrel{\mathcal{L}} x$, we have $tr_2 \mathrel{\mathcal{L}} r_2$.*

Let us first use the fact to prove Lemma 8 and finish the base case. Consider $u, v \in \gamma^{-1}(x)$, *i.e.*, $\beta(\alpha(u)) = \beta(\alpha(v)) = x$. We show that $r_1\alpha(u)r_2 = r_1\alpha(v)r_2$.

By hypothesis, we have $\beta(s) = \beta(r_1)x\beta(r_2)$. Moreover, $\beta(s) \mathrel{\mathcal{J}} x$ since x is smooth by hypothesis. Thus, $x\beta(r_2) \mathrel{\mathcal{J}} x$ and $\beta(r_1)x \mathrel{\mathcal{J}} x$. Hence, since $x\beta(r_2) \leqslant_{\mathcal{R}} x$ and $\beta(r_1)x \leqslant_{\mathcal{L}} x$, Lemma 1 implies $x\beta(r_2) \mathrel{\mathcal{R}} x$ and $\beta(r_1)x \mathrel{\mathcal{L}} x$. Since $\beta(\alpha(u)) = x$, this yields $\beta(\alpha(u)r_2) \mathrel{\mathcal{R}} x$ and $\beta(r_1\alpha(u)) \mathrel{\mathcal{L}} x$. Applying Fact 9 with $t = \alpha(u)r_2$, one gets $r_1\alpha(u)r_2 \mathrel{\mathcal{R}} r_1$ and $r_1\alpha(u)r_2 \mathrel{\mathcal{L}} r_2$.

We get $p, q \in M$ such that $r_1 = r_1\alpha(u)r_2p$ and $r_2 = qr_1\alpha(u)r_2$. Let $t = qr_1\alpha(u)r_2p = r_2p = qr_1$. We combine our two equalities for r_1 and r_2 to obtain,

$$r_1 = r_1\alpha(u)t = r_1(\alpha(u)t)^\omega \quad \text{and} \quad r_2 = t\alpha(u)r_2 = (t\alpha(u))^{\omega+1}r_2. \tag{1}$$

Since $\beta(\alpha(u)) = \beta(\alpha(v))$, we know that $\beta(\alpha(u)t) = \beta(\alpha(v)t)$. Therefore, $(\alpha(u)t, \alpha(v)t)$ is a weak $\mathcal{C}$-pair by Lemma 6, and Item (5) yields $(\alpha(u)t)^{\omega+1} = (\alpha(u)t)^\omega\alpha(v)t(\alpha(u)t)^\omega$. We may now multiply by r_1 on the left and by $\alpha(u)r_2$ on the right to get,

$$r_1(\alpha(u)t)^\omega\alpha(u)(t\alpha(u))^{\omega+1}r_2 = r_1(\alpha(u)t)^\omega\alpha(v)(t\alpha(u))^{\omega+1}r_2.$$

Since we already know from (1) that $r_1 = r_1(\alpha(u)t)^\omega$ and $r_2 = (t\alpha(u))^{\omega+1}r_2$, we get as desired that $r_1\alpha(u)r_2 = r_1\alpha(v)r_2$, finishing the proof of Lemma 8. It remains to prove Fact 9.

Proof of Fact 9. By symmetry, we focus on the first property and leave the second to the reader. Let $t \in M$ such that $\beta(t) \mathrel{\mathcal{R}} x$. We show that $r_1t \mathrel{\mathcal{R}} r_1$. By hypothesis, r_1 is right stable which yields $t' \in M$ such that $\beta(t') \mathrel{\mathcal{R}} x \mathrel{\mathcal{R}} \beta(t)$ and $r_1t' \mathrel{\mathcal{R}} r_1$. Since $\beta(t') \mathrel{\mathcal{R}} \beta(t)$, we have $y \in N$ such that $\beta(t') = \beta(t)y$. Let $p \in M$ such that $\beta(p) = y$: we have $\beta(t') = \beta(tp)$. Since $r_1t' \mathrel{\mathcal{R}} r_1$, we have $q \in M$ such that $r_1 = r_1t'q$ which yields $r_1 = r_1(t'q)^\omega = r_1(t'q)^{\omega+1}$. We have $\beta(t'q) = \beta(tpq)$ which means that $(t'q, tpq)$ is a weak $\mathcal{C}$-pair by Lemma 6. Therefore, Equation (5) yields that $(t'q)^{\omega+1} = (t'q)^\omega tpq(t'q)^\omega$. Finally, we obtain,

$$r_1 = r_1(t'q)^{\omega+1} = r_1(t'q)^\omega tpq(t'q)^\omega = r_1tpq(t'q)^\omega.$$

This implies that $r_1 \leqslant_{\mathcal{R}} r_1t$. Since it is immediate that $r_1t \leqslant_{\mathcal{R}} r_1$, we get $r_1t \mathrel{\mathcal{R}} r_1$. ◀

First inductive case. We now assume that x is not smooth: x and $\beta(s)$ are not $\mathcal{J}$-equivalent. We use induction on our first parameter (the rank of $\beta(s)$). Recall that we assumed $\beta(s) = \beta(r_1)x\beta(r_2)$, which yields $\beta(s) \leqslant_{\mathcal{J}} x$. Thus, we have $\beta(s) <_{\mathcal{J}} x$ by hypothesis.

By definition, $L_s^x[r_1, r_2]$ is the disjoint union of all languages $\alpha^{-1}(t)$ where $t \in M$ satisfies $\beta(t) = x$ and $r_1tr_2 = s$. Therefore, it suffices to show that for any $t \in M$ such that $\beta(t) = x$, we have $\alpha^{-1}(t) \in ADet(\mathcal{C})$. This is immediate by induction. Indeed, since $\beta(t) = x$, we have $\alpha^{-1}(t) = L_t^x[1_M, 1_M]$. Moreover, since $\beta(s) <_{\mathcal{J}} x$, we have $\beta(s) <_{\mathcal{J}} \beta(t)$. It follows that the rank of $\beta(t)$ is strictly smaller than the one of $\beta(s)$. Hence, we may apply induction on our first and most important parameter to get $L_t^x[1_M, 1_M] \in ADet(\mathcal{C})$.

Second inductive case. We assume that either r_1 is not *right stable* or r_2 is not *left stable*. By symmetry, we treat the case when r_1 is not right stable and leave the other to the reader.

▶ Remark. We only apply induction on our two first parameters. Moreover, we show that $L_s^x[r_1, r_2]$ is built from languages in $ADet(\mathcal{C})$ (obtained from induction) using only disjoint union and left deterministic marked concatenations. Induction on our third parameter and right deterministic marked concatenations are used in the case when r_2 is not left stable.

Observe that we have $x <_{\mathcal{J}} 1_N$ (x is not maximal for $\leqslant_{\mathcal{J}}$). Indeed, otherwise, we would have $x \mathrel{\mathcal{R}} 1_N$ by Lemma 1 and r_1 would be left stable: $1_M \in M$ would satisfy $\beta(1_M) = 1_N \mathrel{\mathcal{R}} x$ and $r_11_M = r_1 \mathrel{\mathcal{R}} r_1$. Therefore, there are elements $y \in N$ such that $x <_{\mathcal{J}} y$.

We use this observation to define T as the set of all triples $(y, a, z) \in N \times A \times N$ such that $x = y\gamma(a)z$, $x <_{\mathcal{J}} y$ and $x \mathrel{\mathcal{J}} y\gamma(a)$. Using the definition of T and the fact that $x <_{\mathcal{J}} 1_N$, one may decompose $L_s^x[r_1, r_2]$ as follows (this lemma is proved in the full version).

▶ **Lemma 10.** *The language $L_s^x[r_1, r_2]$ is equal to the following disjoint union,*

$$L_s^x[r_1, r_2] = \biguplus_{(y,a,z)\in T} \left(\biguplus_{t\in\beta^{-1}(y)} \alpha^{-1}(t) \cdot a \cdot L_s^z[r_1 t\alpha(a), r_2] \right).$$

We now use Lemma 10 to show as desired that $L_s^x[r_1, r_2] \in ADet(\mathcal{C})$. Since $ADet(\mathcal{C})$ is closed under disjoint union by definition, it suffices to show that for any $(y, a, z) \in T$ and any $t \in \beta^{-1}(y)$, we have,

$$\alpha^{-1}(t) \cdot a \cdot L_s^z[r_1 t\alpha(a), r_2] \in ADet(\mathcal{C}).$$

We prove that this is a left deterministic marked concatenation of two languages in $ADet(\mathcal{C})$ which concludes the proof.

We start with $\alpha^{-1}(t) \in ADet(\mathcal{C})$. Since $y = \beta(t)$, we have $\alpha^{-1}(t) = L_t^y[1_M, 1_M]$. Moreover, since $\beta(s) = \beta(r_1)x\beta(r_2)$, we have $\beta(s) \leqslant_{\mathcal{J}} x$. Finally, by definition of T we have $x <_{\mathcal{J}} y = \beta(t)$. Altogether, we get $\beta(s) <_{\mathcal{J}} \beta(t)$: the rank of $\beta(t)$ is strictly smaller than the one of $\beta(s)$ and induction on our first parameter yields $\alpha^{-1}(t) = L_t^y[1_M, 1_M] \in ADet(\mathcal{C})$.

We turn to $L_s^z[r_1 t\alpha(a), r_2] \in ADet(\mathcal{C})$. By definition of T, we have $x \mathcal{J} y\gamma(a)$ and $x = y\gamma(a)z$ which yields that $x \mathcal{R} y\gamma(a)$ by Lemma 1. Moreover, since $y = \beta(t)$, it follows that $x \mathcal{R} \beta(t\alpha(a))$. Therefore, since we know that r_1 is **not** right stable (this is our hypothesis), it follows that r_1 and $r_1 t\alpha(a)$ are not $\mathcal{R}$-equivalent. Since it is clear that $r_1 t\alpha(a) \leqslant_{\mathcal{R}} r_1$, it follows that $r_1 t\alpha(a) <_{\mathcal{R}} r_1$: the right index of $r_1 t\alpha(a)$ is strictly smaller than the one of r_1. By induction on our second parameter, we then get that $L_s^z[r_1 t\alpha(a), r_2] \in ADet(\mathcal{C})$.

It remains to show that $\alpha^{-1}(t) \cdot a \cdot L_s^z[r_1 t\alpha(a), r_2]$ is a left deterministic marked concatenation, *i.e.*, that $\alpha^{-1}(t) \cap \alpha^{-1}(t)aA^* = \emptyset$. Since $\beta(t) = y$, we have $\alpha^{-1}(t) \subseteq \gamma^{-1}(y)$ and it suffices to show that $\gamma^{-1}(y) \cap \gamma^{-1}(y)aA^* = \emptyset$. Let $w \in \gamma^{-1}(y)$ and $w' \in \gamma^{-1}(y)aA^*$, we show that $w \neq w'$. Since $(x, a, z) \in T$, we have $x <_{\mathcal{J}} y$ and $x \mathcal{J} y\gamma(a)$. It follows that $y\gamma(a) <_{\mathcal{J}} y$. Finally, we have $\gamma(w) = y$ and $\gamma(w') = y\gamma(a)y'$ for some $y' \in N$. This implies that $\gamma(w') \leqslant_{\mathcal{J}} y\gamma(a) <_{\mathcal{J}} y = \gamma(w)$. Therefore $\gamma(w) \neq \gamma(w')$ which implies that $w \neq w'$.

4 Separation

We now turn to separation for $UPol(\mathcal{C})$ and show that the problem is decidable for any finite quotienting Boolean algebra $\mathcal{C}$. For the sake of avoiding clutter, we fix $\mathcal{C}$ for the section.

▶ Remark. This result may seem weak: our solution for $UPol(\mathcal{C})$-separation requires $\mathcal{C}$ to be finite while $UPol(\mathcal{C})$-membership reduces to $\mathcal{C}$-membership. This intuition is wrong: the result on separation is the strongest. The proof of Theorem 4 shows that when $L \in UPol(\mathcal{C})$, the basic languages in $\mathcal{C}$ needed to build L are all recognized by the syntactic morphism of L. Hence, $L \in UPol(\mathcal{C})$ if and only if $L \in UPol(\mathcal{D})$ where $\mathcal{D} \subseteq \mathcal{C}$ is a *finite class* obtained from the syntactic morphism of L. We lose this when moving to separation: the languages in $\mathcal{C}$ needed to build a potential separator in $UPol(\mathcal{C})$ may not be encoded in our two inputs.

Our algorithm is based on a general framework designed to handle separation problems and to present solutions in an elegant way. It was introduced in [20, 22]. We first summarize what we need in this framework to present our solution for $UPol(\mathcal{C})$-separation.

▶ Remark. The framework of [20, 22] is actually designed to handle a more general decision problem: covering, which generalizes separation to arbitrarily many input languages. Thus, our solution actually yields an algorithm for $UPol(\mathcal{C})$-covering as well. While we do not detail this point due to lack of space, this follows from the definitions of [20, 22].

4.1 Methodology

We briefly recall the framework of [20, 22]. We refer the reader to [22] for details. The approach is based on "rating maps", a notion designed to measure how well a language separates others.

The definition of rating maps relies on semirings. A *semiring* is a set R equipped with two binary operations $+$ and $\cdot$, called addition and multiplication, satisfying the following axioms:

- $(R, +)$ is a commutative monoid whose neutral element is denoted by 0_R.
- $(R, \cdot)$ is a monoid whose neutral element is denoted by 1_R.
- The multiplication distributes over addition: $r \cdot (s + t) = rs + rt$ and $(s + t) \cdot r = sr + tr$.
- The element 0_R is a zero for multiplication: for any $r \in R$, $0_R \cdot r = r \cdot 0_R = 0_R$.

Moreover, we say that a semiring R is *idempotent* when any element $r \in R$ is idempotent for addition: $r + r = r$. Any idempotent semiring R can be equipped with a canonical order "$\leq$": given $s, r \in R$, we have $s \leq r$ when $s + r = r$. It can be verified that this is indeed an order which is compatible with addition and multiplication (R being idempotent is required).

▶ **Example 11.** The set 2^{A^*} of all languages over A is an idempotent semiring: the addition is union and the multiplication is language concatenation. In this case, the canonical order is inclusion ($H \subseteq L$ if and only if $H \cup L = L$). Another important example is the powerset 2^M of any monoid M. Again the addition is union (therefore, the order is inclusion). The multiplication is obtained from the one of M: given $S, T \in 2^M$, $S \cdot T = \{st \mid s \in S \text{ and } t \in T\}$.

Rating maps. A rating map[1] is a *semiring morphism*, $\rho : 2^{A^*} \to R$ where R is a *finite idempotent semiring*. It can be verified that any rating map is compatible with the canonical order ($K \subseteq L \Rightarrow \rho(K) \leq \rho(L)$). For the sake of improved readability, when applying a rating map ρ to a singleton language $\{w\}$, we shall simply write $\rho(w)$ for $\rho(\{w\})$. The connection with separation only requires to consider special rating maps called "*nice*". A rating map $\rho : 2^{A^*} \to R$ is *nice* if for any language $K \subseteq A^*$, we have $\rho(K) = \sum_{w \in K} \rho(w)$ (note that this sum boils down to a finite one, as R is a finite idempotent commutative monoid for addition).

▶ Remark. Any nice rating map $\rho : 2^{A^*} \to R$ is finitely representable: it is determined by the images $\rho(a)$ of letters $a \in A$. We may speak of algorithms whose inputs are nice rating maps.

Solving $UPol(\mathcal{C})$-separation requires to consider a special class of rating maps: the *$\mathcal{C}$-compatible ones* (our algorithm is restricted to them). The definition is based on a canonical equivalence $\sim_{\mathcal{C}}$ on A^* associated to $\mathcal{C}$. Given $u, v \in A^*$, we write $u \sim_{\mathcal{C}} v$ if and only if $u \in L \Leftrightarrow v \in L$ for all $L \in \mathcal{C}$. Clearly, $\sim_{\mathcal{C}}$ is an equivalence relation. For any word $w \in A^*$, we write $[w]_{\mathcal{C}} \subseteq A^*$ for the $\sim_{\mathcal{C}}$-class of w. Moreover, since $\mathcal{C}$ is a finite quotienting Boolean algebra, we have the following classical properties.

▶ **Lemma 12.** *The equivalence $\sim_{\mathcal{C}}$ is a congruence of finite index for word concatenation. Moreover, for any language $L \subseteq A^*$, we have $L \in \mathcal{C}$ if and only if L is a union of $\sim_{\mathcal{C}}$-classes.*

Lemma 12 implies that the set $A^*/\sim_{\mathcal{C}}$ of $\sim_{\mathcal{C}}$-classes is a finite monoid and the map $w \mapsto [w]_{\mathcal{C}}$ is a morphism. For the sake of avoiding confusion with language concatenation, we shall write "$\bullet$" for the monoid multiplication of $A^*/\sim_{\mathcal{C}}$. In general, if $C, D \subseteq A^*$ are $\sim_{\mathcal{C}}$-classes, then $C \bullet D \neq CD$ (indeed, CD is not even a $\sim_{\mathcal{C}}$-class in general).

[1] What we call rating map here is called **multiplicative** rating map in [22] (the "true" rating maps are weaker and do not require a multiplication). We abuse terminology for the sake of improved readability.

We may now define $\mathcal{C}$-compatibility. We say that a rating map $\rho : 2^{A^*} \to R$ is *$\mathcal{C}$-compatible* when there exists a map $r \mapsto [\![r]\!]_{\mathcal{C}}$ from R to $2^{A^*/\sim_{\mathcal{C}}}$ such that (1) for every $K \subseteq A^*$, we have $[\![\rho(K)]\!]_{\mathcal{C}} = \{[w]_{\mathcal{C}} \mid w \in K\}$ and (2) for all $r, r' \in R$, such that $r \leq r'$, we have $[\![r]\!]_{\mathcal{C}} \subseteq [\![r']\!]_{\mathcal{C}}$.

▶ Remark. Intuitively, Condition (1) in the definition of $\mathcal{C}$-compatibility states that the image $\rho(K)$ of any language K records the $\sim_{\mathcal{C}}$-classes of words of K. The purpose of Condition (2) is to constrain the definition of the map $[\![\ ]\!]_{\mathcal{C}}$ on elements that have no preimage under ρ.

Optimal covers. We use rating maps to define objects called "optimal universal $\mathcal{D}$-covers", which encode separation-related information. We fix an arbitrary Boolean algebra $\mathcal{D}$ for which one wants a $\mathcal{D}$-separation algorithm (we are interested in the case $\mathcal{D} = \mathit{UPol}(\mathcal{C})$).

A *cover of some language* L is a *finite* set of languages $\mathbf{K}$ such that $L \subseteq \bigcup_{K \in \mathbf{K}} K$. When $L = A^*$, we speak of *universal cover*. Moreover, we say that $\mathbf{K}$ is a $\mathcal{D}$-cover when all $K \in \mathbf{K}$ belong to $\mathcal{D}$. A fixed rating map $\rho : 2^{A^*} \to R$ is used to define a "quality measure" for $\mathcal{D}$-covers which yields a notion of "best" universal $\mathcal{D}$-cover. Given a finite set of languages $\mathbf{K}$ (such as a universal $\mathcal{D}$-cover), the *ρ-imprint* $\mathcal{I}[\rho](\mathbf{K})$ of $\mathbf{K}$ is the following subset of R:

$$\mathcal{I}[\rho](\mathbf{K}) = \{r \in R \mid r \leq \rho(K) \text{ for some } K \in \mathbf{K}\}.$$

We now define the optimal universal $\mathcal{D}$-covers as those with the smallest possible ρ-imprint (with respect to inclusion). A *universal $\mathcal{D}$-cover $\mathbf{K}$ is optimal for ρ* when $\mathcal{I}[\rho](\mathbf{K}) \subseteq \mathcal{I}[\rho](\mathbf{K}')$ for any universal $\mathcal{D}$-cover $\mathbf{K}'$. In general, there can be infinitely many optimal universal $\mathcal{D}$-covers for a given rating map ρ. The crucial point is that there always exists a least one. This is simple and proved in [22]. The key idea is that there are finitely many possible ρ-imprints (since R is finite) and given two universal $\mathcal{D}$-covers, one may always build a third one which has a smaller ρ-imprint than the first two, by simple use of language intersections.

Finally, a key observation is that by definition, all optimal universal $\mathcal{D}$-covers for ρ share the same ρ-imprint. This unique ρ-imprint is a *canonical* object for $\mathcal{D}$ and ρ called the *$\mathcal{D}$-optimal universal ρ-imprint* and we denote it by $\mathcal{I}_{\mathcal{D}}[\rho]$. That is, $\mathcal{I}_{\mathcal{D}}[\rho] = \mathcal{I}[\rho](\mathbf{K})$ for any optimal universal $\mathcal{D}$-cover $\mathbf{K}$ for ρ.

The connection with separation. We may now explain how these notions are used to handle separation. This is summarized by the following lemma.

▶ **Lemma 13.** *Let $\mathcal{D}$ be a Boolean algebra. If there exists an algorithm that takes as input a nice $\mathcal{C}$-compatible rating map $\rho : 2^{A^*} \to R$ and outputs $\mathcal{I}_{\mathcal{D}}[\rho]$, then $\mathcal{D}$-separation is decidable.*

Let us sketch how to go from computing $\mathcal{D}$-optimal ρ-imprints to $\mathcal{D}$-separation (see [20, 22] for a full proof of Lemma 13). Consider two regular languages L_1 and L_2: we wish to know whether L_1 is $\mathcal{D}$-separable from L_2. Since $\mathcal{C}$ is finite, one can build a monoid morphism $\alpha : A^* \to M$, with M finite, recognizing both L_1 and L_2 as well as all languages in $\mathcal{C}$. Furthermore, one may lift α as a map $\rho : 2^{A^*} \to 2^M$ by defining $\rho(K) = \{\alpha(w) \mid w \in K\}$ for any language $K \subseteq A^*$. It is simple to verify that this map ρ is a nice $\mathcal{C}$-compatible rating map. Moreover, the two following properties (which we prove in the full version) hold:

- L_1 is $\mathcal{D}$-separable from L_2 iff for any $s_1 \in \alpha(L_1)$ and $s_2 \in \alpha(L_2)$, we have $\{s_1, s_2\} \notin \mathcal{I}_{\mathcal{D}}[\rho]$.
- When the first item holds, one may build a separator in $\mathcal{D}$ from any optimal universal $\mathcal{D}$-cover $\mathbf{K}$ for ρ: this separator is the union of all languages intersecting L_1 in $\mathbf{K}$.

By the first item, having an algorithm that computes $\mathcal{I}_{\mathcal{D}}[\rho] \subseteq 2^M$ suffices to decide whether L_1 is $\mathcal{D}$-separable from L_2. Moreover, by the second item, having an algorithm that computes an optimal universal $\mathcal{D}$-cover $\mathbf{K}$ for ρ is enough to build a separator (when it exists).

▶ Remark. Here, we only use the sets of size two in $\mathcal{I}_{\mathcal{D}}[\rho] \subseteq 2^M$. However, $\mathcal{I}_{\mathcal{D}}[\rho]$ contains more information corresponding to the more general $\mathcal{D}$-covering problem considered in [20, 22].

4.2 Computing $UPol(\mathcal{C})$-optimal universal imprints

We use the framework defined above to present an algorithm for $UPol(\mathcal{C})$-separation. We give a characterization $UPol(\mathcal{C})$-optimal imprints. It yields a procedure for computing them.

Consider a rating map $\rho : 2^{A^*} \to R$. For any subset $S \subseteq R$, we say that S is $UPol(\mathcal{C})$-*saturated* (for ρ) if it contains the set $\mathcal{I}_{triv}[\rho] = \{r \in R \mid r \leq \rho(w) \text{ for some } w \in A^*\}$ and is closed under the following operations:

1. *Downset*: for any $s \in S$, if $r \in R$ satisfies $r \leq s$, then we have $r \in S$.
2. *Multiplication*: For any $s, t \in S$, we have $st \in S$.
3. $UPol(\mathcal{C})$-*closure*: Given two $\sim_{\mathcal{C}}$-classes C, D and $s, t \in S$ such that $[\![s]\!]_{\mathcal{C}} = \{C \bullet D\}$ and $[\![t]\!]_{\mathcal{C}} = \{D \bullet C\}$, we have $s^\omega \cdot \rho(C) \cdot t^\omega \in S$.

We are ready to state the main theorem of this section: when ρ is $\mathcal{C}$-compatible, $UPol(\mathcal{C})$-saturation characterizes the $UPol(\mathcal{C})$-optimal universal ρ-imprint.

▶ **Theorem 14.** *Let $\rho : 2^{A^*} \to R$ be a $\mathcal{C}$-compatible rating map. Then, $\mathcal{I}_{UPol(\mathcal{C})}[\rho]$ is the smallest $UPol(\mathcal{C})$-saturated subset of R (with respect to inclusion).*

Clearly, given a nice $\mathcal{C}$-compatible rating map $\rho : 2^{A^*} \to R$ as input, one may compute the smallest $UPol(\mathcal{C})$-saturated subset of R with a least fixpoint algorithm. One starts from $\mathcal{I}_{triv}[\rho]$ (which is clearly computable) and saturates this set with the three above operations. Thus, we get a procedure for computing $\mathcal{I}_{UPol(\mathcal{C})}[\rho]$ from any input nice $\mathcal{C}$-compatible rating map. By Lemma 13, this yields the desired corollary: $UPol(\mathcal{C})$-separation is decidable.

▶ **Corollary 15.** *For any finite quotienting Boolean algebra $\mathcal{C}$, $UPol(\mathcal{C})$-separation is decidable.*

The proof of Theorem 14 is a difficult generalization of the argument we used to show the algebraic characterization of $UPol(\mathcal{C})$ (*i.e.*, Theorem 4). We postpone it to the full version. An interesting byproduct of this proof is an algorithm which computes optimal universal $UPol(\mathcal{C})$-covers (and therefore $UPol(\mathcal{C})$-separators when they exist, as we explained above).

5 Conclusion

We presented a new, self-contained proof that for any quotienting Boolean algebra regular languages $\mathcal{C}$, membership for $UPol(\mathcal{C})$ reduces to membership for $\mathcal{C}$. An interesting byproduct of this proof is that $UPol(\mathcal{C})$ corresponds exactly to the class $ADet(\mathcal{C})$, which is obtained by restricting the unambiguous marked concatenations to left or right deterministic ones. Moreover, we showed that when $\mathcal{C}$ is a finite quotienting Boolean algebra, $UPol(\mathcal{C})$-separation is decidable. This completes similar results of [21] for $Pol(\mathcal{C})$ and $Bool(Pol(\mathcal{C}))$ and of [16, 17] for $Pol(Bool(Pol(\mathcal{C})))$. These results raise several natural questions.

Historically, $UPol(\mathcal{C})$ was investigated together with two weaker operations: left and right deterministic closures. The left (resp. right) deterministic closure of $\mathcal{C}$, is the smallest class containing $\mathcal{C}$ closed under disjoint union and left (resp. right) deterministic marked concatenation. Our results can be adapted to these two weaker operations. In both cases, membership reduces to $\mathcal{C}$-membership when $\mathcal{C}$ is a quotienting Boolean algebra of regular languages and separation is decidable when $\mathcal{C}$ is a finite quotienting Boolean algebra. In fact, these operations are simpler to handle than $UPol(\mathcal{C})$. We leave this for further work.

Another question is whether our results can be pushed to classes built by combining unambiguous polynomial closure with other operations. A natural example is as follows. It is known [17] that $Pol(Bool(Pol(\mathcal{C})))$-separation is decidable when $\mathcal{C}$ is a finite quotienting Boolean algebra. Is this true as well for $UPol(Bool(Pol(\mathcal{C})))$? This seems difficult: the proof of [17] crucially exploits the fact that $Pol(Bool(Pol(\mathcal{C})))$ is closed under concatenation (which is not the case for $UPol(Bool(Pol(\mathcal{C}))))$ to handle the first polynomial closure.

References

1. Jorge Almeida, Jana Bartonová, Ondrej Klíma, and Michal Kunc. On decidability of intermediate levels of concatenation hierarchies. In *Proceedings of the 19th International Conference on Developments in Language Theory, DLT'15*, volume 9168 of *Lecture Notes in Computer Science*, pages 58–70. Springer, 2015.
2. Mustapha Arfi. Polynomial operations on rational languages. In *Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science, STACS'87*, volume 247 of *Lecture Notes in Computer Science*, pages 198–206. Springer, 1987.
3. Mustapha Arfi. Opérations polynomiales et hiérarchies de concaténation. *Theoretical Computer Science*, 91(1):71–84, 1991.
4. Mário Branco and Jean-Éric Pin. Equations defining the polynomial closure of a lattice of regular languages. In *Proceedings of the 36th International Colloquium on Automata, Languages, and Programming, ICALP'09*, volume 5556 of *Lecture Notes in Computer Science*, pages 115–126. Springer, 2009.
5. Volker Diekert, Paul Gastin, and Manfred Kufleitner. A survey on small fragments of first-order logic over finite words. *International Journal of Foundations of Computer Science*, 19(3):513–548, 2008.
6. Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science, LICS'97*, pages 228–235. IEEE Computer Society, 1997.
7. Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Information and Computation*, 179(2):279–295, 2002.
8. James Alexander Green. On the structure of semigroups. *Annals of Mathematics*, 54(1):163–172, 1951.
9. Jean-Éric Pin. Propriétés syntactiques du produit non ambigu. In *Proceedings of the 7th International Colloquium on Automata, Languages and Programming, ICALP'80*, volume 85 of *Lecture Notes in Computer Science*, pages 483–499. Springer, 1980.
10. Jean-Éric Pin. An explicit formula for the intersection of two polynomials of regular languages. In *Proceedings of the 17th International Conference on Developments in Language Theory, DLT'13*, volume 7907 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2013.
11. Jean-Éric Pin. Mathematical foundations of automata theory. In preparation, 2016. URL: `http://www.irif.fr/~jep/MPRI/MPRI.html`.
12. Jean-Éric Pin. *The dot-depth hierarchy, 45 years later*, chapter 8, pages 177–202. World Scientific, 2017.
13. Jean-Éric Pin, Howard Straubing, and Denis Thérien. Locally trivial categories and unambiguous concatenation. *Journal of Pure and Applied Algebra*, 52(3):297–311, 1988.
14. Jean-Éric Pin and Pascal Weil. Polynomial closure and unambiguous product. In *Proceedings of the 22nd International Colloquium on Automata, Languages and Programming, ICALP'95*, volume 944 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 1995.
15. Jean-Éric Pin and Pascal Weil. Polynomial closure and unambiguous product. *Theory of Computing Systems*, 30(4):383–422, 1997.
16. Thomas Place. Separating regular languages with two quantifiers alternations. In *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'15*, pages 202–213. IEEE Computer Society, 2015.
17. Thomas Place. Separating regular languages with two quantifiers alternations. Unpublished, a preliminary version can be found at `https://arxiv.org/abs/1707.03295`, 2018.
18. Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Proceedings of the 38th International*

Symposium on Mathematical Foundations of Computer Science, MFCS'13, volume 8087 of *Lecture Notes in Computer Science*, pages 729–740. Springer, 2013.

19 Thomas Place and Marc Zeitoun. Going higher in the first-order quantifier alternation hierarchy on words. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming, ICALP'14*, volume 8573 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 2014.

20 Thomas Place and Marc Zeitoun. The covering problem: A unified approach for investigating the expressive power of logics. In *Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science, MFCS'16*, volume 58 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 77:1–77:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

21 Thomas Place and Marc Zeitoun. Separation for dot-depth two. In *Proceedings of the 32th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'17*, pages 202–213. IEEE Computer Society, 2017.

22 Thomas Place and Marc Zeitoun. The covering problem. Unpublished, a preliminary version can be found at `https://arxiv.org/abs/1707.03370`, 2018.

23 Thomas Place and Marc Zeitoun. Generic results for concatenation hierarchies. *Theory of Computing Systems*, 2018. Selected papers from CSR'17.

24 John L. Rhodes. A homomorphism theorem for finite semigroups. *Mathematical Systems Theory*, 1:289–304, 1967.

25 Marcel-Paul Schützenberger. Sur le produit de concaténation non ambigu. *Semigroup Forum*, 13:47–75, 1976.

26 Imre Simon. Piecewise testable events. In *Proceedings of the 2nd GI Conference on Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 214–222. Springer, 1975.

27 Pascal Tesson and Denis Thérien. Diamonds are forever: The variety DA. In *Semigroups, Algorithms, Automata and Languages*, pages 475–500. World Scientific, 2002.

28 Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing, STOC'98*, pages 234–240. ACM, 1998.

29 Wolfgang Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25(3):360–376, 1982.

A Superpolynomial Lower Bound for the Size of Non-Deterministic Complement of an Unambiguous Automaton

Mikhail Raskin[1]
LaBRI, University of Bordeaux, 351, cours de la Libération F-33405 Talence cedex, France
raskin@mccme.ru
https://orcid.org/0000-0002-6660-5673

Abstract

Unambiguous non-deterministic finite automata (UFA) are non-deterministic automata (over finite words) such that there is at most one accepting run over each input. Such automata are known to be potentially exponentially more succinct than deterministic automata, and non-deterministic automata can be exponentially more succinct than them.

In this paper we establish a superpolynomial lower bound for the state complexity of the translation of an UFA to a non-deterministic automaton for the complement language. This disproves the formerly conjectured polynomial upper bound for this translation. This lower bound only involves a one letter alphabet, and makes use of the random graph methods.

The same proof also shows that the translation of sweeping automata to non-deterministic automata is superpolynomial.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases unambiguous automata, language complement, lower bound

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.138

Acknowledgements I would like to thank Gabriele Puppis for numerous useful discussions. I would like to thank anonymous reviewers, Thomas Colcombet and Bruno Courcelle for their advice regarding presentation.

1 Introduction

In many areas of computer science, the relationship between deterministic and non-deterministic devices is a subject of significant interest. An intermediate notion between deterministic and non-deterministic computation devices is the notion of unambiguous device. Such a device can make non-deterministic choices, but it is guaranteed that for every input there is at most one accepting execution trace.

For finite automata it is known that non-deterministic automata can be exponentially more succinct than deterministic automata [10]. It is also known that unambiguous automata can be exponentially more succinct than deterministic automata and in other situations they can be exponentially less succinct than non-deterministic automata [8]. The paper establishing exponential separation also defines several automata classes of limited ambiguity and provides exponential separation between some of them.

Other notions of unambiguity have been considered. Some of them (for example, structural unambiguity [9]: for all input words u and all states p, there is at most one run of the

[1] This work was supported by the French National Research Agency (ANR project GraphEn / ANR-15-CE40-0009)

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 138; pp. 138:1–138:11


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

automaton over u starting in an initial state and ending in p) describe a wider class of automata than unambiguity. Some are more restrictive than simple unambiguity (for example, strong unambiguity [14]: there is a set of result states, for every input there is exactly one way to reach a result state, and the result states can be accepting or rejecting). We do not consider these notions in the present paper.

We study the problem of representing a complement of a language specified by a finite automaton. It is easy to see that replacing the set of accepting states with its complement allows to recognize the complement of a language specified by a deterministic finite automaton without increasing the number of states. Complementing a language specified by a non-deterministic finite automaton may require an exponential number of states [1].

It has been conjectured (see for instance [3]) that every unambiguous non-deterministic one-way finite automaton (*UFA*) recognizing some language L can be converted into an *UFA* recognizing the complement of the original language L with polynomial increase in the number of states. The best known lower bound was quadratic [13], while the upper bounds were exponential [6]. The quadratic lower bound holds even for the single-letter alphabet. One of the arguments in favour of the conjecture was the fact that universality and even containment of the languages recognized by unambiguous finite automata can be decided in polynomial time [15].

The case of the single-letter alphabet has a better upper bound for the state complexity of recognizing the complement of the language of a non-deterministic finite automaton. A one-way non-deterministic finite automaton (*NFA*) with n states can be converted to a one-way deterministic finite automaton (*DFA*) with $e^{\Theta(\sqrt{n \log n})}$ states accepting the same language [11]. As a *DFA* can be converted into a *DFA* for the complement of the language without any increase in the number of states, this conversion provides an upper bound on the state complexity of recognizing the complement of the language recognized by an *NFA*. This upper bound is tight [12].

Recognizing the complement of the language of a two-way non-deterministic automaton (*2NFA*) with n states over the single-letter alphabet can be done using an *2NFA* with at most $O(n^8)$ states [4]. The same paper also shows that recognising the complement of the language of a *2DFA* with n states can be done by a $4n$-states *2DFA* for arbitrary alphabet. For the single-letter alphabet the complement of the language of a *2DFA* can be recognized by a *2DFA* with $2n + 3$ states [7].

In the present paper we show a superpolynomial lower bound for the state complexity of recognizing the complement of a language of an unambiguous finite automaton by a non-deterministic finite automaton.

▶ **Theorem 1.** *There exists a sequence of unary UFA $(A_d)_{d\in\mathbb{N}}$ such that every NFA recognising the complement of the language of A_d has size at least $|A_d|^d$.*

The size of A_d is $2^{2^{2^{d^{\Theta(1)}}}}$.

▶ **Corollary 2.** *Worst case, complementing an UFA of size z by an NFA may require more than $z^{(\log\log\log z)^{\Theta(1)}}$ states.*

In other words, complementing an *UFA* requires more than polynomial increase in size regardless of the size of the alphabet, and the bound holds even if we allow the complement to be represented by *NFA*.

We also note that the same languages (and their complements) can be recognized by sweeping deterministic finite automata with a small increase in the state complexity compared to the case of *UFA*.

The proof revolves around a connection between *UFA* and tournaments (orientations of complete graphs), and an observation about existence of tournaments with special properties described in Section 4.

The rest of the paper is structured as follows. In the next section we give the standard definitions. Then we present in Section 3 our construction of the unambiguous automata A_d. It involves the use of tournaments with special properties, and the choice of many relatively close primes. We prove the existence of the suitable tournaments in Section 4, and explain how to chose the prime numbers in Section 5. The Section 6 finishes the proof of Theorem 1. We briefly study the case of sweeping automata in Section 7. In the final section we summarize the results and outline some possible future directions.

2 Definitions

In this section we will remind the definitions of deterministic, unambiguous and non-deterministic finite automata, and their normal forms.

▶ **Definition 3.** An *non-deterministic finite automaton* (*NFA*) is defined by an *alphabet* Σ, a set of *states* Q, a subset of *initial states* $I \subseteq Q$, a subset of *accepting states* $F \subseteq Q$ and the *transition relation* $T \subseteq Q \times \Sigma \times Q$. The *size* of an *NFA* A is the number of its states, and is denoted by $|A|$. A *run* of an *NFA* over a word $u = a_1 \dots a_n$ is a sequence of states $q_0, \dots, q_n$ such that (q_{i-1}, a_i, q_i) belongs to T for all $i = 1 \dots n$ and $q_0 \in I$. The run is *accepting* if its last state is accepting, i.e. $q_n \in F$. A *language* L over alphabet Σ is an subset $L \subseteq \Sigma^*$. The *language recognized by* an automaton A is the set $L(A)$ of all words w such that there exists an accepting run of A on w. An automaton over the single-letter alphabet is called *unary*.

A *deterministic finite automaton* (*DFA*) is an *NFA* such that I is a singleton and for all states q and all letters a there is at most one transition of the form $(q, a, q') \in T$.

An *unambiguous non-deterministic finite automaton* (*UFA*) is an *NFA* such that for every word there is at most one accepting run.

A unary non-deterministic finite automaton is in the *Chrobak normal form* [2] if it consists of a path of states followed by a single nondeterministic choice to a set of disjoint cycles.

An automaton is in *simple Chrobak normal form* if it consists of a disjoint union of cycles, each of them containing exactly one initial state.

The following theorem shows that every *UFA* can be transformed into one in Chrobak normal form without increase in size, and as a consequence we sought the construction that would have this shape.

▶ **Theorem 4** ([5]). *For all regular unary languages, there exists an unambiguous automaton recognizing the language which is minimal in size and is furthermore in Chrobak normal form.*

3 The construction

We present in this section the construction of the automaton A_d involved in the proof of Theorem 1. We also establish the unambiguity of A_d in Lemma 5 and compute its size in Lemma 6.

Parameters

The construction of A_d involves several parts, and the parameters have to be adjusted carefully for the lower bound. In this section, we use many parameters, to be specified in the final proof, in Section 6.

These parameters are the following:

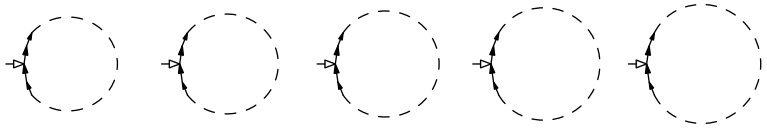
- $n \in \mathbb{N}$ is the number of cycles of the automaton A_d in simple Chrobak normal form.
- R is a tournament of size n: a tournament is an orientation of the edges of the complete undirected graph, see Section 4 for more details. The tournament R will eventually be required to have a special property, established in Lemma 7.
- $b \in \mathbb{N}$ is used as a basis for numbering, and we set $N = b^n$.
- $P = \{p_i \mid i = 0 \dots N-1\}$ is a set of N distinct primes. These will eventually be chosen sufficiently close one from each other thanks to Lemma 10.

The construction

We now construct the automaton A_d as follows.

- It consists of n disjoint cycles $C_1, \dots, C_n$, the cycle C_i having as length m_i which is the product of the primes p_j's such that the ith digit of j in base b is 0 (the digits are numbered from 1 to n). We write that p_j *belongs to* m_i if $p_j | m_i$.
- The 0th state of the cycle C_i is *initial*.
- The rth state of a cycle C_i is *accepting* if it satisfies three conditions:
  1. r is non null,
  2. r modulo p belongs to $\{0, i\}$ for all p belonging to m_i,
  3. if iRj for some j, then there exists a prime p belonging to both m_i and m_j such that $r \mod p = i$.

  And in this case, we call r an *accepting remainder for* m_i.



Let us look more precisely at the structure of this automaton.

We first note that the empty word is not accepted by this automaton, thanks to Item 1 of the definition. One can also note that each cycle is the product of b^{n-1} distinct prime numbers. Furthermore, if one computes the gcd of ℓ different m_i's, the result is the product of $b^{n-\ell}$ prime numbers. Hence there are many primes dividing a cycle, there are many primes dividing simultaneously two cycles, and so on.

Of course, the subtlety in this construction lies in the choice of the accepting remainders for each m_i. This has to respect several constraints. The remainders are chosen to be sufficiently complicated for allowing the lower bound proof, and there should be not too many of them in order to guarantee the unambiguity for A_d. In particular if Condition 3 was omitted, it would be easy find accepting remainders for two distinct m_i's that would yield ambiguity[2]. The Condition 3 is used to resolve these conflictual situations, and when an input would be accepted by two cycles, the tournament is used to "declare the winner".

Concretely, we prove:

▶ **Lemma 5.** *The automaton A_d is unambiguous.*

Proof. Assume that the automaton A_d would be ambiguous. This would mean that there exists a word, of length ℓ, such that it is accepted by two distinct cycles. Let us say by C_i and C_j. This means that $r = \ell \mod m_i$ is an accepting remainder for m_i, and $r' = \ell$

[2] Indeed, let p being a prime of m_i and p' a prime of m_j, consider, by the Chinese remainder theorem an integer ℓ that is equal to i modulo p, equal to j modulo p', and null modulo all other primes. In the absence of Assumption 3, the word of length ℓ would be accepted by both C_i and C_j.

mod m_j is an accepting remainder for m_j. Let us assume without loss of generality that iRj. This implies by Item 3 that there is a prime number p that belongs to both m_i and m_j such that $r \mod p = i$. Hence $\ell \mod p = i$ since p belongs to m_i. Hence $r' \mod p = i$ since p also belongs to m_j. However, we know that r' is an accepting remainder for m_j, therefore Item 2 requires that $r' \mod p \in \{0, j\}$. A contradiction. ◀

We conclude this section by computing the size of this automaton.

▶ **Lemma 6.** *The automaton A_d has between $n(\min P)^{n^{b-1}}$ and $n(\max P)^{n^{b-1}} = n(\max P)^{\frac{N}{b}}$ states.*

Proof. Indeed, the automaton is a union of n cycles and the length of each cycle is a product of $b^{n-1} = \frac{N}{b}$ primes from P. ◀

The rest of the proof is now devoted to showing that there are no small non-deterministic automata for the complement of the language accepted by A_d.

4 Tournaments

A *tournament graph* (or simply a *tournament*) of size n is an orientation of the complete graph. In our case, we see it as a relation over $\{1, \dots, n\}$ such that for all distinct $i, j = 1 \dots n$, either iRj and not jRi, or jRi and not iRj. By convention, iRi is assumed to never hold.

As we have seen in the previous section, a tournament is used as a parameter in the construction of the automaton A_d. For our lower bound proof to go through, we use the fact that this tournament has a special technical property, that is shown possible according to the following lemma.

▶ **Lemma 7.** *For all positive integers k, there exists a tournament R such that the following property holds: for all $E \subseteq R$, if for all vertices x there exists a vertex y such that xEy, then E contains at least k distinct edges that do not share an extremity.*

It is possible to chose a tournament with this property of size $n = 12k^2 2^{2k}$.

The rest of this section is devoted to the proof of Lemma 7. Note that this proof involves a probabilistic argument.

The core notion used in the proof, and therefore the notion at the core of the entire proof of Theorem 1, is the notion of inbound-covering sets.

▶ **Definition 8.** A set S is an *inbound-covering set* for a tournament R if for all vertices x outside S, we have xRy for some $y \in S$.

▶ **Lemma 9.** *For every positive integer h there exist a large enough integer n and a tournament of size n such that the smallest inbound-covering set has size larger than h.*

It is enough to take $n = 3h^2 2^h$.

Proof. Consider a uniformly random tournament of size n, i.e., the vertices are fixed as $1, \dots, n$, and for all $i < j$, one tosses a fair coin in order to chose whether iRj or jRi. Consider an arbitrary set $S \subset V(G)$ of size h. The probability (over the choice of a tournament) that a given vertex $v \in V(G) \setminus S$ has at least one edge from v to S is $1 - 2^{-h}$. For a given set S and $v_1, v_2, \dots \in V(G) \setminus S$, the existence of an outgoing edge from v_i towards S is independent for the different vertices (indeed for all $i \neq j$, the set of edges from v_i to S and the set of edges from v_j to S are disjoint and thus their orientations are chosen independently). Therefore the probability for a given set S to be inbound-covering is equal to $(1 - 2^{-h})^{(n-h)}$. Note that since $\log(1 - 2^{-h}) < (-2^{-h})$, this quantity is bounded from above by $e^{-2^{-h}(n-h)}$ $(\star)$.

Let us provide now an upper bound on the probability α that a tournament has an inbound covering set of size h. Since there are (less than) n^h sets of size n and using $(\star)$, we immediately get that

$$\alpha \leqslant n^h \exp\left(-\frac{n-h}{2^h}\right) = \exp\left(h\log n - \frac{n-h}{2^h}\right) .$$

We shall prove now that for $h \geqslant 8$ and $n = 3h^2 2^h$, this quantity α is smaller than one, which concludes the proof. According to the above inequality, it is sufficient for proving $\alpha < 1$ to establish that $h\log n < \frac{n-h}{2^h}$, which is equivalent to $h2^h \log n < n - h$. We establish this inequality as follows:

$$\begin{aligned} h2^h \log n &= h2^h(h\log 2 + 2\log h + \log 3) \\ &< h2^h \times (2h) \\ &= 2h^2 2^h \\ &< 3h^2 2^h - h \\ &= n - h \end{aligned}$$

◀

▶ Remark. Note that, as it is customary with probabilistic constructions, our choice of n is in fact enough to ensure that *most* tournaments have no inbound-covering sets of sizes up to h.

Now we can prove Lemma 7.

Proof. By Lemma 9 we can pick a tournament with orientation R that has no inbound-covering sets of size up to $h = 2k$. We can choose such a tournament of size $n = 3h^2 2^h = 12k^2 2^{2k}$.

Assume we have already constructed 2ℓ distinct vertices $x_1, y_1, \ldots, x_\ell, y_\ell$ forming edges $(x_1, y_1), \ldots, (x_\ell, y_\ell)$. Since $S = \{x_1, y_1, \ldots, x_\ell, y_\ell\}$ has cardinality $2\ell < 2h$, it is not an inbound-covering set. Hence, one can find a vertex $x_{\ell+1}$ such that there is an edge from all vertices of S to it. We know that E must contain some edge $(x_{\ell+1}, y_{\ell+1})$ from $x_{\ell+1}$, and this edge cannot lead to S, so the edge $(x_{\ell+1}, y_{\ell+1})$ doesn't share an extremity with any previously chosen edge. Applying this argument by induction on ℓ for $\ell = 0$ to k, we have proved Lemma 7. ◀

5 Choice of primes

▶ **Lemma 10.** *For all large enough N it is possible to select N primes no larger than $4N^2 \log N$ within a factor of $1 + \frac{1}{N}$ of each other.*

Proof. We will take the interval of length $3N\log N$ between $3N^2\log N$ and $4N^2\log N$ that contains the most primes. By the Prime number theorem there are

$$\frac{3N^2\log N}{2\log N + \log\log N + \log 3} + o(N^2) = \frac{3}{2}N^2 + o(N^2)$$

primes no larger than $3N^2\log N$ and

$$\frac{4N^2\log N}{2\log N + \log\log N + \log 4} + o(N^2) = 2N^2 + o(N^2)$$

primes no larger than $4N^2\log N$. Therefore, there are $\frac{1}{2}N^2 + o(1)$ primes between $3N^2\log N$ and $4N^2\log N$. If we divide this interval into subintervals of length $3N\log N$, the average

subinterval will contain

$$\frac{1}{2}N^2\frac{3N\log N}{N^2\log N}(1+o(1)) = \frac{3}{2}N + o(N)$$

primes, which is enough. ◀

6 The lower bound

In this section we present the main combinatorial argument of the proof, and complete the proof of Theorem 1.

▶ **Lemma 11.** *A non-deterministic automaton that accepts the complement of the language of A_d has to have at least $(\min P)^{N(1-\exp(-\frac{k}{b^2}))}$ states.*

Let us fix ourselves a non-deterministic automaton C_d that accepts the complement of the language of A_d.

The principle of the proof of Lemma 11 is the following: as we have already noticed, the word of length $\prod_{p\in P} p$, since it is congruent to 0 modulo all the m_i's, is not accepted by A_d (this follows from Condition 1 in the definition of accepting remainders). Thus it has to be accepted by C_d. Since this word is very long (the length is much larger than the bound we want to prove), the run of C_d that accepts it has to visit twice some state and perform a cycle in the mean time. We shall look at what are the words obtained by pumping this cycle, that are all accepted by C_d, and obtain from this analysis that this cycle in C_d has to be rather long.

The core combinatorial result justifying this intuition is the following.

▶ **Lemma 12.** *Let A_d be constructed from a tournament of size $n = 12k^2 2^{2k}$ satisfying the conclusion of Lemma 7.*

Let x and y be integers such that

(a) *$(\prod_{p\in P} p) = x + y$, and;*

(b) *$(xs+y) \mod m_i$ is not an accepting remainder modulo m_i for all i and all $s \geqslant 0$,*

then x has to be divisible by at least $N(1-(1-\frac{1}{b}^2)^k)$ distinct primes from P.

Proof. Consider the set $E \subseteq R$ defined as

$$E = \{(i,j) \in R \mid \gcd(m_i, m_j) | x\} .$$

The proof then goes in two steps. We shall show in step 1 that the assumption for E in Lemma 7 are fulfilled. Then we will apply Lemma 7 and conclude in step 2.

Step 1: We assume that E does not fulfill the assumptions of Lemma 7, and head toward a contradiction. This means that we assume that there exists an $i = 1 \dots n$ such that whenever iRj then $\gcd(m_i, m_j)$ does not divide x.

According to the Chinese remainder theorem and existence of inverse in $\mathbb{Z}/p\mathbb{Z}$ there exists $s > 0$ such that $(xs+y) \mod p = i$ for all primes $p \in P$ that do not divide x. Note that for a prime p that divides x, since furthermore p divides $x+y$ (assumption (a) of the lemma), we obtain $p|y$, and thus $(xs+y) \mod p = 0$. Overall, for $r = xs + y$, we have that for all primes $p \in P$:

$$r \mod p = \begin{cases} 0 & \text{if } p \text{ divides } x, \text{ and} \\ i & \text{otherwise.} \end{cases}$$

Let us show that this r is an accepting remainder:

1 Let j be such that iRj. By assumption, $\gcd(m_i, m_j) \nmid x$. Hence, there exists $p \in P$ that divides m_i but not x. For this p we know that $r \mod p = i$, therefore $r \mod m_i \neq 0$.
2 We have seen above that $r \mod p \in \{0, i\}$.
3 Let j be such that iRj. According to the assumption, $\gcd(m_i, m_j)$ does not divide x. Hence there exists a prime p that divides both m_i and m_j but not x. For this prime, we have seen that $r \mod p = i$.

However, we knew by assumption (b) of the lemma that a number of the form $xs + y$ such as r cannot be an accepting remainder. This is contradiction, and thus terminates the proof of the step 1.

Step 2: Let us now apply Lemma 7. According to the lemma, there are k distinct E-edges $(i_1, j_1), \ldots, (i_k, j_k)$ that do not share an extremity. Let us count the number of primes that divide both m_{i_t} and m_{j_t} for some $t = 1 \ldots k$ (and thus divide x). By construction of the m_i's, it contains all the primes p_v such that both the i_tth and the j_tth digits (in the base-b notation) of v are null for some $t = 1 \ldots k$.

We will first count the primes in P **not** dividing x. These are the primes p_v with v having a nonzero digit in at least one of the two positions i_t and j_t for every t. There are k pairs of positions and there are $b^2 - 1$ combinations of digits that are not $(0, 0)$. There are also $n - 2k$ positions with no such constraints. The total number of possible combinations is $(b^2-1)^k b^{n-2k} = (b^2(1-\frac{1}{b^2}))^k b^{n-2k} = b^n(1-\frac{1}{b^2})^k = N(1-\frac{1}{b^2})^k$.

The primes in P dividing x are all the other primes, and there are $N - N(1-\frac{1}{b^2})^k = N(1-(1-\frac{1}{b^2})^k)$ of them. ◀

Let us prove Lemma 11.

Proof. Let us fix a tournament of size n according to Lemma 7.

Let use fix a set of primes according to Lemma 10.

An *NFA* recognizing the complement of the language has to have a cycle, because the complement is infinite. Consider the word of length $\prod_{p \in P} p$. This length is obviously greater than $|A_d|^d$. If the *NFA* has an accepting run over the word with no cycles, it has to be very large. Otherwise, let C be a cycle of length x occurring in this run, and y be the remaining part of the run length, i.e. $\prod_{p \in P} p = x + y$ (a). The product of all the primes $\prod_{p \in P} p$ has remainder zero modulo every modulus m_i in the construction. By iterating $s \geqslant 0$ times the cycle C, we obtain that the word of length $xs + y$ has to be accepted by C_d. Thus it is not accepted by A_d, and hence $(xs + y) \mod m_i$ is not an accepting remainder for all $s \geqslant 0$ and $i = 1 \ldots n$ (b).

Hence the assumptions (a) and (b) of Lemma 12 are fulfilled. It follows that the cycle C has a length x divisible by $N(1-(1-\frac{1}{b}^2)^k)$ distinct primes from P.

This ensures that the cycle C has a length at least $(\min P)^{N(1-(1-\frac{1}{b}^2)^k)}$. Since furthermore $(1-\frac{1}{b^2})^k < \exp(-\frac{1}{b^2})^k = \exp -\frac{k}{b^2}$, this is at least $(\min P)^{N(1-\exp(-\frac{k}{b^2}))}$.

The size of the *NFA* cannot be less than that. ◀

We can finally conclude the proof of the main theorem of this paper, Theorem 1.

Proof of Theorem 1. Let us fix d. Let $b = 2d, k = b^2, n = 12k^2 2^{2k}, N = b^n$.

By Lemma 6 the size of the automaton A_d is at most $n(\max P)^{\frac{N}{b}}$ states. This automaton is unambiguous by Lemma 5.

By Lemma 11 each *NFA* recognizing the complement of the language of A_d must have at least $(\min P)^{N(1-\exp(-\frac{k}{b^2}))}$ states. As $\frac{k}{b^2} = 1$, the size of the *NFA* cannot be less than $(\min P)^{0.6N}$.

We now only need to verify that $(O(n)(\max P)^{\frac{N}{b}})^d < (\min P)^{0.6N}$. But indeed, for large enough d we have $\min P > N \gg n \gg d$ and

$$(O(n)(\max P)^{\frac{N}{b}})^d < ((O(n)(1+2\frac{1}{N})(\min P))^{\frac{N}{2d}})^d$$
$$< O(n)^{\frac{N}{2}} \exp(\frac{d}{N})(\min P)^{\frac{N}{2}} < (\min P)^{0.6N}$$

In case of d not large enough, we can replace the automaton with the automaton for the smallest large enough d.

Let us estimate the size of A_d. We know that b is linear in d, k is quadratic in d, n is $2^{\Theta(d^2)}$, N is $b^n = b^{2^{\Theta(d^2)}} = 2^{(\log b)2^{\Theta(d^2)}} = 2^{2^{\Theta(d^2)}}$. The primes in P are all $\Theta(N^2 \log N)$. Then the size of the automaton A_d is $\Theta(n(\min P)^{\frac{N}{b}}) = (\min P)^{\Theta(\frac{N}{b})} = (N^2 \log N)^{\Theta(\frac{N}{b})} = 2^{2^{\Theta(d^2)}2^{\Theta(d^2)}} = 2^{2^{2^{\Theta(d^2)}}} = 2^{2^{2^{\Theta(1)}}}$ ◀

7 Sweeping automata

We will now make some additional remarks about the application of the main construction to two-way and sweeping automata.

First we remind the definitions of two-way and sweeping automata.

▶ **Definition 13.** A two-way non-deterministic finite automaton (*2NFA*) is defined by an alphabet Σ, a set of *states* $Q \sqcup \{\top, \bot\}$, a subset of *initial states* I, and the *transition relation* $T \subseteq Q \times (\Sigma \sqcup \{\vdash, \dashv\}) \times (Q \sqcup \{\top, \bot\}) \times \{+1, -1\}$. We call $\vdash$ and $\dashv$ *endpoint markers*.

A *run* of an *2NFA* on an input word $u_1 \dots u_k$ is a list of pairs of positions and states, $(x_0 = 1, q_0 \in I), (x_1, q_1), \dots, (x_n, q_n)$ such that all transitions are allowed and the run ends with one of the special states $\top, \bot$. The exact conditions are as follows:

1. x_0 is 1;
2. q_0 is in I;
3. all x_i are between 0 and $k+1$;
4. $(q_{i-1}, w_{x_{i-1}}, q_i, x_i - x_{i-1}) \in T$ for all $i = 1 \dots n$, in which we assume that $u_0 = \vdash$ and $u_{k+1} = \dashv$;
5. the last state q_n is either $\top$ or $\bot$;
6. $x_i \neq x_{i-1}$ for all $i = 1 \dots n$.

A run is *accepting* if the last state is $\top$.

A *two-way non-deterministic finite automaton* (*2DFA*) is a *2NFA* such that for every state q and every letter a there is at most one transition of the form $(q, s, q', j) \in T$.

A *sweeping two-way deterministic finite automaton* (*swNFA*) is a *2NFA* with exactly one initial state such that for each state q all the transitions of the form (q, s, q', j) where s is in Σ have the same j.

A *swDFA* is an *swNFA* that is also a *2DFA*.

▶ **Lemma 14.** *The languages $L(A_d)$ and $\overline{L(A_d)}$ constructed in the proof of Theorem 1 can also be recognized. by a swDFA of size $|A_d|$.*

Proof. A *swDFA* can go through the word n times calculating the remainder modulo the next modulus each time. This construction requires the same number of states as the *UFA* constructed in the proof of Theorem 1. Such a *swDFA* can be constructed to recognize either the language or its complement. ◀

▶ **Theorem 15.** *Converting a unary sweeping two-way deterministic automaton to a non-deterministic finite automaton for the same language may require a superpolynomial size.*

Proof. Consider the automata constructed in Lemma 14. ◀

8 Conclusion and further directions

We have constructed a counterexample to the conjecture that the complement of a language recognized by an *UFA* can be recognized by an *UFA* with polynomial increase in the number of states. Moreover, in our example the language and its complement are easy to recognize by a *swDFA* with approximately the same number of states, but the complement requires superpolynomial number of states in the recognizing *NFA* even without the requirement of unambiguity. The example only uses the single-letter alphabet.

The construction provides a relatively weak kind of superpolynomial growth. It would be interesting to improve the lower bound. It seems likely that the number of primes used in the construction could be reduced, making the growth faster.

The question about exponential separation in the case of a general alphabet remains open. We hope that our counterexample to the conjectured polynomial upper bound for complementing *UFA* will inspire new results in this area.

References

1 Jean-Camille Birget. Partial orders on words, minimal elements of regular languages and state complexity. *Theor. Comput. Sci.*, 119(2):267–291, 1993. doi:10.1016/0304-3975(93)90160-U.

2 Marek Chrobak. Finite automata and unary languages. *Theor. Comput. Sci.*, 47(3):149–158, 1986. doi:10.1016/0304-3975(86)90142-8.

3 Thomas Colcombet. Unambiguity in automata theory. In Jeffrey Shallit and Alexander Okhotin, editors, *Descriptional Complexity of Formal Systems - 17th International Workshop, DCFS 2015, Waterloo, ON, Canada, June 25-27, 2015. Proceedings*, volume 9118 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015. doi:10.1007/978-3-319-19225-3_1.

4 Viliam Geffert, Carlo Mereghetti, and Giovanni Pighizzini. Complementing two-way finite automata. *Inf. Comput.*, 205(8):1173–1187, 2007. doi:10.1016/j.ic.2007.01.008.

5 Tao Jiang, Edward McDowell, and Bala Ravikumar. The structure and complexity of minimal nfa's over a unary alphabet. *Int. J. Found. Comput. Sci.*, 2(2):163–182, 1991. doi:10.1142/S012905419100011X.

6 Jozef Jirásek Jr., Galina Jirásková, and Juraj Sebej. Operations on unambiguous finite automata. In Srecko Brlek and Christophe Reutenauer, editors, *Developments in Language Theory - 20th International Conference, DLT 2016, Montréal, Canada, July 25-28, 2016, Proceedings*, volume 9840 of *Lecture Notes in Computer Science*, pages 243–255. Springer, 2016. doi:10.1007/978-3-662-53132-7_20.

7 Michal Kunc and Alexander Okhotin. State complexity of operations on two-way finite automata over a unary alphabet. *Theor. Comput. Sci.*, 449:106–118, 2012. doi:10.1016/j.tcs.2012.04.010.

8 Hing Leung. Descriptional complexity of nfa of different ambiguity. *Int. J. Found. Comput. Sci.*, 16(5):975–984, 2005. doi:10.1142/S0129054105003418.

9 Hing Leung. Structurally unambiguous finite automata. In Oscar H. Ibarra and Hsu-Chun Yen, editors, *Implementation and Application of Automata, 11th International Conference, CIAA 2006, Taipei, Taiwan, August 21-23, 2006, Proceedings*, volume 4094 of *Lecture Notes in Computer Science*, pages 198–207. Springer, 2006. doi:10.1007/11812128_19.

10 Oleg B Lupanov. A comparison of two types of finite automata. *Problemy Kibernetiki*, 9:321–326, 1963.

11 J.I. Lyubich. Estimates of the number of states that arise in the determinization of a nondeterministic autonomous automaton. *Sov. Math., Dokl.*, 5:345–348, 1964.

12 Filippo Mera and Giovanni Pighizzini. Complementing unary nondeterministic automata. *Theor. Comput. Sci.*, 330(2):349–360, 2005. doi:10.1016/j.tcs.2004.04.015.

13 Alexander Okhotin. Unambiguous finite automata over a unary alphabet. *Inf. Comput.*, 212:15–36, 2012. doi:10.1016/j.ic.2012.01.003.

14 Seppo Sippu and Eljas Soisalon-Soininen. *Parsing Theory. Vol. 1: Languages and Parsing.* Springer-Verlag New York, Inc., New York, NY, USA, 1988.

15 Richard Edwin Stearns and Harry B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM J. Comput.*, 14(3):598–611, 1985. doi:10.1137/0214044.

The Isomorphism Problem for Finite Extensions of Free Groups Is In PSPACE

Géraud Sénizergues
LABRI, Bordeaux, France
geraud.senizergues@u-bordeaux.fr

Armin Weiß
Universität Stuttgart, FMI, Germany
armin.weiss@fmi.uni-stuttgart.de

Abstract

We present an algorithm for the following problem: given a context-free grammar for the word problem of a virtually free group G, compute a finite graph of groups $\mathcal{G}$ with finite vertex groups and fundamental group G. Our algorithm is non-deterministic and runs in doubly exponential time. It follows that the isomorphism problem of context-free groups can be solved in doubly exponential space. Moreover, if, instead of a grammar, a finite extension of a free group is given as input, the construction of the graph of groups is in NP and, consequently, the isomorphism problem in PSPACE.

2012 ACM Subject Classification Mathematics of computing → Graph theory, Theory of computation → Grammars and context-free languages, Theory of computation → Computational complexity and cryptography

Keywords and phrases virtually free groups, context-free groups, isomorphism problem, structure tree, graph of groups

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.139

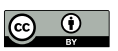
Related Version A full version of the paper is available at [23], `https://arxiv.org/abs/1802.07085`.

Acknowledgements G.S. thanks the FMI for hosting him from October to the end of the year 2017. Both authors acknowledge the financial support by the DFG project DI 435/7-1 "Algorithmic problems in group theory" for this work.

1 Introduction

The study of algorithmic problems in group theory was initiated by Dehn [6] when he introduced the word and the isomorphism problem. The word problem asks whether a given word over a (finite) set of generators represents the identity of the group. It also can be viewed as a formal language, namely $\varphi^{-1}(1) \subseteq \Sigma^*$ for some surjective monoid homomorphism $\varphi : \Sigma^* \to G$. The isomorphism problem receives two finite presentations as input, the question is whether the groups they define are isomorphic. Although both these problems are undecidable in general [18, 3], there are many classes of groups where at least the word problem can be decided efficiently.

One of these classes are the finitely generated *virtually free* groups (groups with a free subgroup of finite index). It is easy to see that the word problem of a finitely generated virtually free group can be solved in linear time. Indeed, it forms a deterministic context-free language. A seminal paper by Muller and Schupp [16] shows the converse: every group



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 139; pp. 139:1–139:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

with a context-free word problem is virtually free. Since then, also a wide range of other characterizations of virtually free groups have emerged – for a survey we refer to [1, 9].

The isomorphism problem of virtually free groups is also decidable as Krstić showed in [15] (indeed, later Dahmani and Guirardel showed that even the isomorphism problem for the larger class of hyperbolic groups is decidable [5]). Here the input consists of two arbitrary finite presentations with the promise that both define virtually free groups. Unfortunately, the approach in [15] does not give any bound on the complexity. For the special case where the input is given as finite extensions of free groups or as context-free grammars for the word problems, Sénizergues [21, 22] showed that the isomorphism problem is primitive recursive. Krstić's and Sénizergues' approaches both compute so-called graphs of groups, which encode groups acting on trees, and then check these graph of groups for "isomorphism". By the work of Karrass, Pietrowski and Solitar [14], a finitely generated group is virtually free if and only if it is the fundamental group of a finite graph of groups with finite vertex groups.

Contribution. We improve the complexity for the isomorphism problem by showing:

- **(A)** Given a context-free grammar for the word problem of a context-free group G, a graph of groups for G with finite vertex groups can be computed in $\mathsf{NTIME}(2^{2^{\mathcal{O}(n^2)}})$ (Theorem 33).
- **(B)** Given a virtually free presentation for G, a graph of groups for G with finite vertex groups can be computed in NP (Theorem 34).
- **(C)** The isomorphism problem for context-free groups given as grammars is in $\mathsf{DSPACE}(2^{2^{\mathcal{O}(n^2)}})$ (Theorem 37).
- **(D)** The isomorphism problem for virtually free groups given as virtually free presentations is in PSPACE (Theorem 38).

Here, a *virtually free presentation* for G consists of a free group F plus a set of representatives S for the quotient $F\backslash G$ together with relations describing pairwise multiplications of elements from F and S. Typical examples of virtually free presentations are finite extensions of free groups (i. e., where the free sugroup F is normal in G). For non-deterministic function problems we use the convention, that every accepting computation must yield a correct result; but the results of different accepting computations might differ[1].

The results C and D can be seen be to follow from A and B rather easily. Indeed, we conclude from Forester's work on deformation spaces [10] that two graphs of groups with finite vertex groups and isomorphic fundamental groups can be transformed one into each other by a sequence of slide moves (Proposition 35).

Our approach for proving A and B is as follows: in both cases the algorithm simply guesses a graph of groups together with a map and afterwards it verifies deterministically whether the map is indeed an isomorphism. The latter can be done using standard results from formal language theory. The difficult part is to show the existence of a "small" graph of groups and isomorphism (within the bounds of A and B).

For this, we introduce the *structure tree* theory by Dicks and Dunwoody [7] following a slightly different approach by Diekert and Weiß [8] based on the *optimal cuts* of the Cayley graph (Section 2.3). The optimal cuts can be seen as the edge set of some tree on which the group G acts. By Bass-Serre theory, this yields the graph of groups we are aiming

[1] Thus, B means that the graph of groups can be computed in NPMV in the sense of [20]. More precisely, it can be rephrased as follows: the multi-valued function mapping a virtually free presentation for G into a pair $(\mathcal{G}, \varphi)$, where $\mathcal{G}$ is a graph of groups and $\varphi : \pi_1(\mathcal{G}) \to G$ is an isomorphism of polynomial size, is everywhere defined and belongs to the class FNP as defined in [19].

for. Vertices in the graph of groups are defined in terms of equivalence classes of optimal cuts. The key in the proof is to bound the size of the equivalence classes. Using Muller and Schupp's [16] notion of k-triangulability, Sénizergues [22] proved bounds on the size of finite subgroups and on the number of edges in a reduced graph of groups for a context-free group, from which we derive our bounds.

Outline. After fixing our notation, we recall basic facts from Bass-Serre theory and the results from [22] and give a short review on structure trees based on [8]. Section 3, develops bounds on the size of the vertices (= equivalence classes of cuts) of the structure tree. After that, we introduce virtually free presentations formally and we derive stronger bounds for this case in Section 4. Section 5 completes the proofs of A and B. Finally, in Section 6 we derive C and D and we conclude with some open questions. Due to space constraints most of the proofs are omitted; they can be found in the full version on arXiv [23].

2 Preliminaries

Complexity. We use the following convention for non-deterministic function problems: each accepting computation path must yield a correct answer – though different accepting paths can compute different correct answers. We use this convention to define the classes NP (non-deterministic polynomial time) and NTIME($f(n)$) (non-deterministic time bounded by $f(n)$). Otherwise, we use standard complexity classes P (deterministic polynomial time), PSPACE (polynomial space) and DSPACE($f(n)$) (deterministic space bounded by $f(n)$) for both decision and function problems.

Words. An *alphabet* is a (finite) set Σ; an element $a \in \Sigma$ is called a *letter*. The set Σ^n forms the set of *words* of length n. The length of $w \in \Sigma^n$ is denoted by $|w|$. The set of all words is denoted by Σ^*. It is the free monoid over Σ – its neutral element is the empty word 1.

Context-free grammars. We use standard notation for context-free grammars: a context-free grammar is a tuple $\mathbb{G} = (V, \Sigma, P, S)$ with variables V, terminals Σ, a finite set of productions rules $P \subseteq V \times (V \cup \Sigma)^*$, and a start symbol S. We denote its size by $\|\mathbb{G}\| = |V| + |\Sigma| + \sum_{A\to\alpha\in P} |\alpha|$. It is in *Chomsky normal form* if all productions are of the form $S \to 1$, $A \to a$ or $A \to BC$ with $A, B, C \in V$, $a \in \Sigma$. For further definitions, we refer to [12].

Groups. Let Σ be an alphabet and $R \subseteq \Sigma^* \times \Sigma^*$. The monoid (or group) G presented by (Σ, R) is defined as $G = \Sigma^*/R = \Sigma^*/{=_G}$ where $=_G$ is the smallest monoid congruence over Σ^* containing R. There is a canonical projection $\pi : \Sigma^* \to G$. The *word problem* of the group G is the formal language $\mathrm{WP}(G) = \{w \in \Sigma^* \mid w =_G 1\} = \pi^{-1}(1)$. A *symmetric alphabet* is an alphabet Σ endowed with an involution $a \mapsto \overline{a}$ without fixed points (i. e., $\overline{\overline{a}} = a$ and $\overline{a} \neq a$ for all $a \in \Sigma$). If Σ is a symmetric alphabet for G (i. e., $G = \Sigma^*/{=_G}$), we always assume that $a\overline{a} =_G \overline{a}a =_G 1$ for all $a \in \Sigma$ (without writing these relations explicitly).

Let Σ be a symmetric alphabet and $w \in \Sigma^*$. We say that w is *freely reduced* if w has no factor $a\overline{a}$ for any letter $a \in \Sigma$. Given an arbitrary set X, the *free group* over X is denoted by $F(X)$. It is defined as the group presented by $(X \cup \overline{X}, \{x\overline{x} \mid x \in X \cup \overline{X}\})$.

Graphs. A *graph* $\Gamma = (V, E, s, t, \bar{\cdot})$ is given by the following data: a set of vertices $V = V(\Gamma)$, a set of edges $E = E(\Gamma)$ together with two *incidence* maps $s : E \to V$ and $t : E \to V$ and an involution $E \to E$, $e \mapsto \overline{e}$ without fixed points such that $s(e) = t(\overline{e})$. The *degree* of a vertex u is the number of incident edges. An *undirected edge* is a set $\{e, \overline{e}\}$. For the cardinality of sets of edges we usually count the number of undirected edges.

A (finite) *path* from v_0 to v_n is a pair of sequences $((v_0,\dots,v_n),(e_1,\dots,e_n))$ such that $s(e_i)=v_{i-1}$ and $t(e_i)=v_i$ for all $1\le i\le n$. Similarly, a *bi-infinite path* is a pair of sequences $((v_i)_{i\in\mathbb{Z}},(e_i)_{i\in\mathbb{Z}})$ such that $s(e_i)=v_{i-1}$ and $t(e_i)=v_i$ for all $i\in\mathbb{Z}$. A path is *simple* if the vertices are pairwise distinct. It is *closed* if $v_0=v_n$. Depending on the situation we also denote paths simply by the sequence of edges or the sequence of vertices. The *distance* $d(u,v)$ between vertices u and v is defined as the length (i. e., the number of edges) of a shortest path connecting u and v. For $A,B\subseteq V(\Gamma)$ with $A,B\neq\emptyset$, the distance is defined as $d(A,B)=\min\{\,d(u,v)\mid u\in A,v\in B\,\}$.

For $S\subseteq V(\Gamma)$ we define $\Gamma-S$ to be the induced subgraph of Γ with vertices $V(\Gamma)\setminus S$. For $C\subseteq V(\Gamma)$, we write $\overline{C}$ for the complement of C, i. e., $\overline{C}=V(\Gamma)\setminus C$. We call C connected, if the induced subgraph is connected. A group G acts on a graph Γ, if it acts on both $V(\Gamma)$ and $E(\Gamma)$ and the actions preserve the incidences.

Cayley graphs. Let G be a group with a symmetric alphabet Σ. The *Cayley graph* $\Gamma=\Gamma_\Sigma(G)$ of G (with respect to Σ) is defined by $V(\Gamma)=G$ and $E(\Gamma)=G\times\Sigma$, with the incidence functions $s(g,a)=g$, $t(g,a)=ga$, and involution $\overline{(g,a)}=(ga,\overline{a})$. For $r\in\mathbb{N}$ let $B(r)=\{\,u\in V(\Gamma)\mid d(u,1)\le r\,\}$ denote the ball with radius r around the identity.

Cuts. For a subset $C\subseteq V(\Gamma)$ we define the *edge* and *vertex boundaries* of C as follows:

$$\vec{\delta}C=\{\,e\in E(\Gamma)\mid s(e)\in C,t(e)\in\overline{C}\,\}=\text{directed edge boundary},$$
$$\partial C=\{\,s(e)\mid e\in\vec{\delta}C\,\}=\text{inner vertex boundary},$$
$$\beta C=\{\,s(e)\mid e\in\vec{\delta}C\text{ or }\overline{e}\in\vec{\delta}C\,\}=\partial C\cup\partial\overline{C}=\text{vertex boundary}.$$

▶ **Definition 1.** A *cut* is a subset $C\subseteq V(\Gamma)$ such that C and $\overline{C}$ are both non-empty and connected and $\vec{\delta}C$ is finite. The *weight* of a cut is $|\vec{\delta}C|$. If $|\vec{\delta}C|\le K$, we call C a *K-cut*.

2.1 Bass-Serre theory

We give a brief summary of the basic definitions and results of Bass-Serre theory [24].

▶ **Definition 2** (Graph of Groups)**.** Let $Y=(V(Y),E(Y),s,t,\bar{\cdot})$ be a connected graph. A *graph of groups $\mathcal{G}$ over Y* is given by the following data:

- **(i)** For each vertex $P\in V(Y)$ there is a *vertex group* G_P.
- **(ii)** For each edge $y\in E(Y)$ there is an *edge group* G_y such that $G_y=G_{\overline{y}}$.
- **(iii)** For each edge $y\in E(Y)$ there is an injective homomorphism from G_y to $G_{s(y)}$, which is denoted by $a\mapsto a^y$. The image of G_y in $G_{s(y)}$ is denoted by G_y^y.

Since we have $G_y=G_{\overline{y}}$, there is also a homomorphism $G_y\to G_{t(y)}$ defined by $a\mapsto a^{\overline{y}}$. The image of G_y in $G_{t(y)}$ is denoted by $G_y^{\overline{y}}$. A graph of groups is called *reduced* if $G_y^y\neq G_{s(y)}$ whenever $s(y)\neq t(y)$ for $y\in E(Y)$. Throughout we assume that all graphs of groups are connected and finite (i. e., Y is a connected, finite graph).

Fundamental group of a graph of groups. We begin with the group $F(\mathcal{G})$. It is defined as the free product of the free group $F(E(Y))$ and the groups G_P for $P\in V(Y)$ modulo the set of defining relations $\{\,\overline{y}a^yy=a^{\overline{y}}\mid a\in G_y,\,y\in E(Y)\,\}$. As an alphabet we fix the disjoint union $\Delta=\biguplus_{P\in V(Y)}(G_P\setminus\{1\})\cup E(Y)$ throughout. Now, we have

$$F(\mathcal{G})=F(\Delta)/\{\,gh=[gh],\,\overline{y}a^yy=a^{\overline{y}}\mid P\in V(Y),\,g,h\in G_P;\,y\in E(Y),\,a\in G_y\,\},$$

where $[gh]$ denotes the element obtained by multiplying g and h in G_P.

For $P \in V(Y)$ we define a subgroup $\pi_1(\mathcal{G}, P)$ of $F(\mathcal{G})$ by the elements $g_0 y_1 \cdots g_{n-1} y_n g_n \in F(\mathcal{G})$, such that $y_1 \cdots y_n$ is a closed path from P to P and $g_i \in G_{s(y_{i+1})}$ for $0 \leq i < n$ and $g_n \in G_P$. The group $\pi_1(\mathcal{G}, P)$ is called the *fundamental group* of $\mathcal{G}$ with respect to the base point P. Since we assumed Y to be connected, there exists a spanning tree $T = (V(Y), E(T))$ of Y. The *fundamental group* of $\mathcal{G}$ with respect to T is defined as

$$\pi_1(\mathcal{G}, T) = F(\mathcal{G}) / \{ y = 1 \mid y \in T \}.$$

▶ **Proposition 3** ([24]). *The canonical homomorphism ψ from the subgroup $\pi_1(\mathcal{G}, P)$ of $F(\mathcal{G})$ to the quotient group $\pi_1(\mathcal{G}, T)$ is an isomorphism. In particular, the two definitions of the fundamental group are independent of the choice of the base point or the spanning tree.*

A word $w \in \Delta^*$ is called *reduced* if it does not contain a factor gh with $g, h \in G_P$ for some P or a factor $\overline{y} a^y y$ with $y \in E(Y)$, $a \in G_y$.

▶ **Lemma 4** ([24, Thm. I.11]). *A reduced word in $\pi_1(\mathcal{G}, P)$ represents the trivial element if and only if it is the empty word.*

The quotient of a G-tree. Graphs of groups arise in a natural way in situations where a group G acts (from the left) on some connected tree $Z = (V, E)$ without edge inversion, i. e., $\overline{e} \neq ge$ for all $e \in E$, $g \in G$. We let $Y = G \backslash Z$ be the quotient graph with vertex set $V(Y) = \{ Gv \mid v \in V \}$ and edge set $E(Y) = \{ Ge \mid e \in E \}$ and incidences and involution induced by Z. By choosing representatives we find embeddings $\iota : V(Y) \hookrightarrow V$ and $\iota : E(Y) \hookrightarrow E$ and we can assume that $\iota(V(Y))$ induces a connected subgraph of Z and that $\iota(\overline{y}) = \overline{\iota(y)}$ for all $y \in E(Y)$. For $P \in V(Y)$, $y \in E(Y)$, we define vertex and edge groups as the stabilizers of the respective representatives: $G_P = \mathrm{Stab}(\iota P) = \{ g \in G \mid g\iota P = \iota P \}$ and $G_y = \mathrm{Stab}(\iota y) = \{ g \in G \mid g\iota y = \iota y \}$. Note that as abstract groups the vertex and edge groups are independent of the choice of representatives since stabilizers in the same orbit are conjugate. Moreover, for each $y \in E(Y)$ there are $g_y, h_y \in G$ such that $s(\iota y) = g_y \iota P$ and $t(\iota y) = h_y \iota Q$ for $P = s(y)$ and $Q = t(y)$. Note that g_y and h_y are not unique; still the left cosets $g_y G_P$ resp. $h_y G_Q$ are uniquely determined. Clearly, we can choose them such that $g_y = h_{\overline{y}}$ and $h_y = g_{\overline{y}}$. This yields two embeddings:

$$G_y \to G_P, \quad a \mapsto a^y = \overline{g}_y a g_y, \qquad \text{and} \qquad G_y \to G_Q, \quad a \mapsto a^{\overline{y}} = \overline{h}_y a h_y. \tag{1}$$

Hence, we have obtained a well-defined graph of groups $\mathcal{G}$ over Y. Notice that the G_y^y and $G_y^{\overline{y}}$ depend on the choice of g_y and h_y (and change via conjugation when changing them).

We define a homomorphism $\varphi : \Delta^* \to G$ by $\varphi(g) = g$ for $g \in G_P$, $P \in V(Y)$. For $y \in E(Y)$, we set $\varphi(y) = \overline{g}_y h_y$. That means $\varphi(y)$ maps some edge in the preimage of y and terminating in $\iota t(y)$ to an edge in the preimage of y with source in $\iota s(y)$. By our assumption, we have $\overline{\varphi(y)} = \overline{h}_y g_y = \varphi(\overline{y})$. Since $\varphi(\overline{y} a^y y) = \varphi(\overline{y})\varphi(a^y)\varphi(y) = \overline{h}_y g_y a^y \overline{g}_y h_y = a^{\overline{y}} = \varphi(a^{\overline{y}})$, we obtain a well-defined homomorphism $\varphi : F(\mathcal{G}) \to G$.

▶ **Theorem 5** ([24]). *The restriction $\varphi : \pi_1(\mathcal{G}, P) \to G$ is an isomorphism.*

2.2 Context-free groups and graphs

▶ **Definition 6.** A group is called *context-free*, if its word problem is a context free language.

Notice that the word problem of a context-free group is decidable in polynomial time – even if the grammar is part of the input – by applying the CYK algorithm (see e. g. [12]).

▶ **Definition 7** (k-triangulable)**.** Let Γ be a graph. Let $k \in \mathbb{N}$ and let $\gamma = v_0, v_1, \ldots, v_n = v_0$ be a sequence of vertices Γ such that $d(v_{i-1}, v_i) \leq k$ for all $i \in \{1, \ldots, n\}$ (e. g. γ can be a closed path). Let P a convex polygon in the plane whose vertices are labeled by the vertices of γ (i.e. we consider γ as a simple closed curve in the plane). A *k-triangulation* of γ is a triangulation of P which does not introduce any additional vertices (thus only consists of "diagonal" edges) and such that vertices joined by a diagonal edge are at distance at most k. If $n < 3$, we consider γ as triangulated. If every closed path γ has a k-triangulation, then Γ is called k-triangulable and we call k the triangulation constant of Γ.

▶ **Lemma 8** ([16, Thm. I])**.** *Let (V, Σ, P, S) be a context-free grammar in Chomsky normal form for the word problem of a group G where Σ is a symmetric alphabet. Then the Cayley graph Γ can be k-triangulated for $k = 2^{|P|}$.*

Note that in [16] only the existence of some k is shown; however, an easy induction shows the bound of Lemma 8. Moreover, a slightly worse bound applies if Σ is not symmetric [23].

▶ **Lemma 9** ([17, p.65])**.** *Let Γ be k-triangulable and let $r \in \mathbb{N}$. If C is a connected component of $\Gamma - B(r)$, then* $\operatorname{diam}(\partial C) \leq 3k$.

▶ **Lemma 10.** *Let Γ be connected and k-triangulable and let $C \subseteq V(\Gamma)$ be a cut. Then* $\operatorname{diam}(\beta C) \leq \frac{3k}{2}|\vec{\delta} C|$.

This lemma is asserted (without proof) in [21, Lemma 6] with a slightly worse bound on $\operatorname{diam}(\beta C)$. For a proof see [23]. The following upper-bounds will be crucial.

▶ **Proposition 11** ([22, Prop. 1.2])**.** *Let Γ be the Cayley graph of a group G on a symmetric alphabet Σ and let Γ be k-triangulable. Then $|H| \leq |\Sigma|^{12k+10}$ for every finite subgroup $H \leq G$.*

▶ **Theorem 12** ([22, Thm. 1.4])**.** *Let Γ be the Cayley graph of a group G on a symmetric alphabet Σ and let Γ be k-triangulable. Then every reduced graph of groups $\mathcal{G}$ admitting G as fundamental group has at most $|\Sigma|^{12k+11}$ undirected edges.*

2.3 Optimal cuts and structure trees

We briefly present the construction of optimal cuts and the associated structure tree from [8, 9]. While in [8], the proof was for arbitrary accessible, co-compact, locally finite graphs, here we assume that Γ is the Cayley graph of a context-free group. We are interested in bi-infinite simple paths which can be split into two infinite pieces by some cut. For a bi-infinite simple path α denote:

$$\mathcal{C}(\alpha) = \left\{ C \subseteq V(\Gamma) \;\middle|\; C \text{ is a cut and } |\alpha \cap C| = \infty = \left|\alpha \cap \overline{C}\right| \right\},$$
$$\mathcal{C}_{\min}(\alpha) = \{ C \in \mathcal{C}(\alpha) \mid |\delta C| \text{ is minimal in } \mathcal{C}(\alpha) \},$$

where we identify α with its set of vertices. If $C \in \mathcal{C}(\alpha)$, we say that C *splits* α. We define the set of *minimal cuts* $\mathcal{C}_{\min}$ as the union of the $\mathcal{C}_{\min}(\alpha)$ over all bi-infinite simple paths α. Since every bi-infinite simple path α with $\mathcal{C}(\alpha) \neq \emptyset$ can be split by a cut which is a connected component of $\Gamma - B(m)$ for some $m \in \mathbb{N}$, the next lemma follows from Lemma 9 and 10.

▶ **Lemma 13.** *Let Γ be k-triangulable and let d denote the degree of Γ. Then for every $C \in \mathcal{C}_{\min}$ we have $|\vec{\delta} C| \leq d^{3k+2}$ and* $\operatorname{diam}(\beta C) \leq \frac{3k}{2} d^{3k+2}$.

Two cuts C and D are called *nested*, if one of the four inclusions $C \subseteq D$, $C \subseteq \overline{D}$, $\overline{C} \subseteq D$ or $\overline{C} \subseteq \overline{D}$ holds. By Lemma 13, with $K = d^{3k+3}$ for every bi-infinite simple path α with

$\mathcal{C}(\alpha) \neq \emptyset$ there exists some cut $C \in \mathcal{C}(\alpha)$ with $|\vec{\delta}C| \leq K$. We fix this number K. For a cut C let $m(C)$ denote the number of K-cuts that are not nested with C. It follows from [25] that $m(C)$ is always finite, see also [8, Lem. 3.4].

▶ **Definition 14.** The set of *optimal cuts* is defined as

$$\mathcal{C}_{\text{opt}}(\alpha) = \{\, C \in \mathcal{C}_{\min}(\alpha) \mid m(C) \leq m(D) \text{ for all } D \in \mathcal{C}_{\min}(\alpha) \,\},$$
$$\mathcal{C}_{\text{opt}} = \bigcup \{\, \mathcal{C}_{\text{opt}}(\alpha) \mid \alpha \text{ is a bi-infinite simple path} \,\}.$$

▶ **Definition 15.** A set $\mathcal{C} \subseteq \mathcal{C}(\Gamma)$ of cuts is called a *tree set*, if $\mathcal{C}$ is pairwise nested, closed under complementation and for all $C, D \in \mathcal{C}$ the set $\{\, E \in \mathcal{C} \mid C \subseteq E \subseteq D \,\}$ is finite.

▶ **Proposition 16** ([8])**.** *$\mathcal{C}_{\text{opt}}$ is a tree set.*

▶ **Definition 17.** Let $\mathcal{C}$ be a tree set. We can now define the following relation:

$$C \sim D :\Longleftrightarrow C = D \text{ or } (\overline{C} \subsetneqq D \text{ and } E \in \mathcal{C}, \overline{C} \subsetneqq E \subseteq D \Longrightarrow E = D).$$

Indeed, $\sim$ is an equivalence relation – see e. g. [7]. The intuition behind this definition is: We consider $\mathcal{C}$ as the edge set of a graph, and define two edges to be incident to the same vertex, if no other edge lies "between" them.

▶ **Definition 18.** Let $\mathcal{C}$ be a tree set and let $T(\mathcal{C})$ denote the graph defined by

$$V(T(\mathcal{C})) = \{\, [C] \mid C \in \mathcal{C} \,\}, \qquad E(T(\mathcal{C})) = \mathcal{C}.$$

The incidence maps are defined by $s(C) = [C]$ and $t(C) = [\overline{C}]$. The involution $\overline{C}$ is defined by the complementation $\overline{C} = V(\Gamma) \setminus C$; hence, we do not need to change notation.

The directed edges are in canonical bijection with the pairs $([C], [\overline{C}])$. Indeed, let $C \sim D$ and $\overline{C} \sim \overline{D}$. It follows $C = D$ because otherwise $C \subsetneqq \overline{D} \subsetneqq C$. Thus, $T(\mathcal{C})$ is an undirected graph without self-loops and multi-edges. Indeed, $T(\mathcal{C})$ is a tree [7].

▶ **Theorem 19** ([8, Thm. 5.9])**.** *Let Γ be a connected, k-triangulable, locally finite graph. Let a group G act on Γ such that $G\backslash\Gamma$ is finite and each node stabilizer G_v is finite. Then G acts on the tree $T(\mathcal{C}_{\text{opt}})$ such that all vertex and edge stabilizers are finite and $G\backslash T(\mathcal{C}_{\text{opt}})$ is finite.*

Complete cut sets. By Proposition 16 and Theorem 19, $\mathcal{C}_{\text{opt}}$ is a tree set on which G acts with finitely many orbits such that the vertex stabilizers $G_{[C]} = \{\, g \in G \mid gC \sim C \,\}$ of the structure tree are finite. We shall call a set of cuts with these properties a *complete cut set*.

3 Bounds on the structure tree

In order to prove our main result, we have to show that there exists a "small" graph of groups together with a "small" isomorphism. We start with the structure tree and bound the size of the equivalence classes and the diameter of the boundaries of the cuts in one equivalence class. As before Γ is the Cayley graph of a context-free group G.

Avoiding edge inversion. Let $\mathcal{C}$ be a tree set (e. g. $\mathcal{C} = \mathcal{C}_{\text{opt}}$). We aim to construct a graph of groups as described in Section 2.1 from the structure tree $T(\mathcal{C})$. However, if the action of G on $T(\mathcal{C})$ is with edge inversion, the construction cannot be applied directly. Instead, we switch to a subdivision $\widetilde{T}(\mathcal{C})$ of $T(\mathcal{C})$ by putting a new vertex in the middle of every edge which is inverted (in particular, $V(T(\mathcal{C})) \subseteq V(\widetilde{T}(\mathcal{C}))$). Formally, $\widetilde{T}(\mathcal{C})$ is defined as follows: for every edge C of $T(\mathcal{C})$ with $gC = \overline{C}$ for some $g \in G$ we remove C and $\overline{C}$ and instead add a new vertex $v_{\{C,\overline{C}\}}$ together with edges $C_1, \overline{C}_1, C_2, \overline{C}_2$ with $gC_1 = \overline{C}_2$, $gC_2 = \overline{C}_1$ and $s(C_1) = [C]$, $t(C_1) = v_{\{C,\overline{C}\}}$, $s(C_2) = v_{\{C,\overline{C}\}}$, and $t(C_2) = [\overline{C}]$. We extend this in a G-equivariant way to the whole tree. From now on we work with the tree $\widetilde{T}(\mathcal{C})$.

Reduced cut sets. Given a tree set $\mathcal{C}$ with a finite quotient graph $G\backslash\widetilde{T}(\mathcal{C})$, we obtain a graph of groups as in in Section 2.1. We aim to apply Theorem 12, to bound the number of edges in this graph of groups. However, the graph of groups might not be reduced. In terms of the set of cuts $\mathcal{C}$ this means that $G_{[C]} = G_C$ for some $C \in \mathcal{C}$ and either $[C]$ and $[\overline{C}]$ are not in the same G-orbit or there is some $g \in G$ with $gC = \overline{C}$. (Note that the latter case implies that the action on $T(\mathcal{C})$ is with edge inversion. Thus, the new vertex $v_{\{C,\overline{C}\}}$ is introduced and the condition of being reduced is violated for the vertex $G \cdot [C]$ of the corresponding graph of groups.) Nevertheless, in this case we can switch to a subset $\mathcal{C}' \subseteq \mathcal{C}$ such that the corresponding graph of groups is reduced: if there is some cut $C \in \mathcal{C}$ with $G_{[C]} = G_C$ and either $[C] \notin G \cdot \{[\overline{C}]\}$ or $\overline{C} \in G \cdot \{C\}$, then we can replace $\mathcal{C}$ by $\mathcal{C} \smallsetminus G \cdot \{C, \overline{C}\}$ (in terms of the structure tree this means we collapse the respective edges). If no such $C \in \mathcal{C}$ remains, we have obtained a *reduced* set of cuts $\mathcal{C}'$. Since the number of G-orbits of cuts is finite, this procedure terminates. The following lemmas are straightforward to verify.

▶ **Lemma 20.** *Let $\mathcal{C}$ be a complete cut set and let $\mathcal{C}'$ be the reduced cut set obtained by the above procedure. Then $\mathcal{C}'$ is also complete (i. e., all vertex stabilizers are still finite).*

▶ **Lemma 21.** *Let $\mathcal{C}$ be a reduced cut set and let $\widetilde{T}(\mathcal{C})$ be the associated subdivision of the structure tree without edge inversion. Then the graph of groups built on $G\backslash\widetilde{T}(\mathcal{C})$ is reduced.*

Let Ξ be an upper bound on the order of finite subgroups of G and let Θ be a bound on the number of undirected edges in a reduced graph of groups for G. Notice that by Proposition 11, we have $\Xi \leq d^{12k+10}$ and by Theorem 12 we have $\Theta \leq d^{12k+11}$ where k is the triangulation constant and d the degree of Γ. The following lemma is straightforward to prove using the fact that every orbit of cuts in $\mathcal{C}$ yields an edge in the graph of groups.

▶ **Lemma 22.** *Let $\mathcal{C}$ be a reduced complete set of cuts and let $C \sim D \in \mathcal{C}$. Then we have $|\{\, g \in G \mid gD \sim C \,\}| \leq \Xi$ and $|[C]| \leq 2 \cdot \Theta \cdot \Xi$.*

▶ **Lemma 23.** *Let $\mathcal{C}$ be a tree set of cuts and let G act on $\mathcal{C}$. Let $C \in \mathcal{C}$ and $C \in P \subseteq [C]$. Then $P \neq [C]$ if and only if there is some $E \in [C] \smallsetminus P$ with $d(\partial E, \bigcup_{D \in P} \partial D) \leq 1$.*

Proof. The if-part is clear. Thus, let $P \neq [C]$. Then there is some $E \in [C] \smallsetminus P$. Since $\overline{E} \subsetneq D$ for all $D \in P$, we have $\emptyset \neq \overline{E} \subseteq \bigcap_{D \in P} D$. Now, if $\partial\overline{E} \subseteq \bigcup_{D \in P} \partial D$, we are done. Otherwise, there is some vertex $u \in \overline{E} \subseteq \bigcap_{D \in P} D$ with $d(u, \bigcup_{D \in P} \partial D) \geq 1$. Since $\partial\big(\bigcap_{D \in P} D\big) \subseteq \bigcup_{D \in P} \partial D$, we also find a vertex $v \in \bigcap_{D \in P} D$ with $d(v, \bigcup_{D \in P} \partial D) = 1$ by following a path from u to $\partial\big(\bigcap_{D \in P} D\big)$ inside $\bigcap_{D \in P} D$. Notice that, in particular, we have

$$v \notin \beta D \cup \overline{D} \qquad \text{for all } D \in P. \tag{2}$$

Now since Γ is vertex-transitive, we can find some cut $\widetilde{E} \in \mathcal{C}$ such that $v \in \beta\widetilde{E}$. After possibly exchanging $\widetilde{E}$ with its complement, we can assume that $\widetilde{E} \subsetneq C$ or $\widetilde{E} \subseteq \overline{C}$. The

latter would imply $v \in \beta\widetilde{E} \subseteq \beta C \cup \overline{C}$ contradicting (2). Moreover, for any other $D \in P$, we have $\widetilde{E} \subseteq D$ because all other possibilities for $\widetilde{E}$ and D being nested lead to a contradiction:

- if $\overline{D} \subsetneq \widetilde{E}$, then $\overline{D} \subsetneq \widetilde{E} \subsetneq C$ contradicting $D \sim C$,
- if $D \subsetneq \widetilde{E}$, then $D \subseteq \widetilde{E} \subseteq C$ and $\overline{D} \subseteq C$ contradicting $\overline{C} \neq \emptyset$,
- if $\widetilde{E} \subseteq \overline{D}$, then $v \in \beta\widetilde{E} \subseteq \beta D \cup \overline{D}$ contradicting (2).

Thus, $\widetilde{E} \subseteq \bigcap_{D\in P} D$. Let $E \in \mathcal{C}$ be minimal with respect to inclusion such that $\widetilde{E} \subseteq \overline{E} \subsetneq C$. Then $E \sim C$, but $E \notin P$ because $v \in \beta\widetilde{E} \subseteq \beta E \cup \overline{E}$.

It remains to verify that $d(\partial E, \bigcup_{D\in P} \partial D) \leq 1$. Let $w \in \partial D$ for some $D \in P$ a vertex with $d(w, v) = 1$. Then, we have $w \in \beta D \cup \overline{D} \subseteq \beta E \cup E$. Consider the two cases: $v \in \overline{E}$ and $v \in \partial E$. If $v \in \overline{E}$, then $w \in \beta E \cap \partial D$ and hence $d(\partial E, \bigcup_{D\in P} \partial D) \leq 1$. If $v \in \partial E$, then $d(\partial E, \bigcup_{D\in P} \partial D) \leq d(v, w) = 1$. ◀

Now, an easy inductive argument shows the next lemma.

▶ **Lemma 24.** *Let $\mathcal{C}$ be a complete set of cuts and $R \in \mathbb{N}$ such that* $\mathrm{diam}\beta C \leq R$ *for all $C \in \mathcal{C}$. Let $C \in \mathcal{C}$, then* $\mathrm{diam}\Big(\bigcup_{C\sim D} \beta C\Big) \leq (R+1) \cdot |[C]|$.

▶ **Lemma 25.** *Let Γ be the Cayley graph of G. Moreover, assume that*

- *R bounds the diameter of the boundary of minimal cuts,*
- *Θ bounds the number of undirected edges of a reduced graph of groups for G,*
- *Ξ is an upper bound on the size of finite subgroups of G.*

Then there exists a graph of groups $\mathcal{G}$ over Y and an isomorphism $\varphi : \pi_1(\mathcal{G}, T) \to G$ with

(i) $|V(Y)| \leq \Theta + 1$,
(ii) $|G_P| \leq \Xi$ *for all* $P \in V(Y)$,
(iii) $|\varphi(a)| \leq 4(R+1) \cdot (\Theta+1)^2 \cdot \Xi$ *for every* $a \in \bigcup_{P\in V(Y)} G_P \cup E(Y)$.

Points i and ii of Lemma 25 are immediate. The proof iii starts with the set of optimal cuts. As described at the beginning of this section, one can switch to a reduced, complete subset $\mathcal{C}$ yielding a reduced graph of groups over $G\backslash\widetilde{T}(\mathcal{C})$ by Lemmas 20 and 21. Following the construction of the graph of groups in Section 2.1, we can choose representatives for $V(Y) \subseteq V(\widetilde{T}(\mathcal{C}))$ for $G\backslash V(\widetilde{T}(\mathcal{C}))$ such that $\partial C \subseteq B(\Lambda)$ for any $C \in P$ and $P \in V(Y)$ with $\Lambda = 2(R+1) \cdot (\Theta+1) \cdot \Theta \cdot \Xi + \Theta$. This bound follows from Lemmas 24 and 22. Thus, we have a graph of groups and it remains to bound the size of the isomorphism φ. Consider the action of G on its Cayley graph Γ: every $g \in G_P$ for $P \in V(Y)$ maps a vertex from $B(\Lambda)$ to another vertex in $B(\Lambda)$ (namely all vertices in $\bigcup_{D\in P} \partial D$). Likewise the image of an edge D maps $\partial\overline{D} \subseteq B(\Lambda+1)$ into $B(\Lambda)$. Since the action is free, iii follows. For details, see [23].

4 Stronger bounds for virtually free presentations

Let us start with a virtually free group G given as a free subgroup $F(X)$ of finite index and a system of representatives S of $F\backslash G$. That means every group element can be written in a unique way as xs with $x \in F(X)$ and $s \in S$. Moreover, this normal form can be computed in linear time from an arbitrary word by successively applying "commutation rules" of letters from S and $X \cup \overline{X} \cup S$ to the word. This gives a virtually free presentation. For this special case, we can derive stronger bounds on the triangulation constant k and other parameters.

Formally, a *virtually free presentation* $\mathcal{V}$ for G is given by the following data:

- finite sets $X, \overline{X}, S$, where $X \cup \overline{X}$ is a symmetric alphabet and $(X \cup \overline{X}) \cap S = \emptyset$,
- for all $y \in X \cup \overline{X}$, $r, t \in S$, there are words $x_{r,y}, x_{r,t} \in (X \cup \overline{X})^*$, $s_{r,y}, s_{r,t} \in S \cup \{1\}$ with

$$ry =_G x_{r,y}s_{r,y} \qquad\qquad rt =_G x_{r,t}s_{r,t} \tag{3}$$

fulfilling two properties:

(i) for all $r \in S$ there is some $r' \in S$ such that $s_{r',r} = 1$ (i. e., G is a group),

(ii) the equations (3), oriented from left to right, together with the free reductions $x\overline{x} = 1$ for $x \in X \cup \overline{X}$ form a *confluent* rewriting system (for a definition, see e. g. [2, 13]).

We write S^1 for $S \cup \{1\}$. Clearly the associated rewriting system is terminating (noetherian), $F(X)$ is a subgroup of G, $G = F(X) \cdot S^1$, and $F(X) \cap S = \emptyset$ (hence S^1 is a system of right-representatives for $F(X)$). Note that properties i and ii can be checked in polynomial time. Using this confluent rewriting system, every $g \in G$ can be uniquely written in its *normal form* $g = xs$ where $x \in (X \cup \overline{X})^*$ is a freely reduced word and $s \in S^1$. Given any word in $(X \cup \overline{X} \cup S)^*$, the normal form can be computed in linear time from left to right by applying the identities (3) and reducing freely. This is the computation of a deterministic pushdown automaton for the word problem of G:

▶ **Lemma 26.** *Let G be the group defined by a virtually free presentation $\mathcal{V}$. Then a deterministic pushdown automaton for* $\mathrm{WP}(G)$ *can be computed in polynomial time.*

Notice that a finite extension of a free group is a special case of a virtually free presentation where $F(X)$ is a normal subgroup of G (i. e., $s_{r,y} = r$ for all $r \in S$, $y \in X \cup \overline{X}$). We assume that $\mathcal{V}$ is written down in a naive way as input for algorithms: there is a table which for all $a \in X \cup \overline{X} \cup S$ and $r \in S$ contains a word $x_{r,a}$ and some $s_{r,a} \in S$. The size (number of letters) of this table is $|S| \cdot (2\,|X| + |S|) \cdot \max\left\{\, |x_{r,a}| + 1 \;\middle|\; a \in X \cup \overline{X} \cup S,\, r \in S \,\right\}$. Up to logarithmic factors, this is the number of bits required to write down $\mathcal{V}$ this way.

We can always add a disjoint copy of formal inverses $\overline{S}$ of S representing S^{-1} in G. Note that for $\overline{s} \in \overline{S}$ this yields the rule $r\overline{s} = x_{r,\overline{s}}, s_{r,\overline{s}}$ for some $s_{r,\overline{s}} \in S$ where $x_{r,\overline{s}} = x_{s_{r,\overline{s}},s}^{-1}$. In particular, $|x_{r,\overline{s}}| \leq \max\left\{\, |x_{r,a}| \;\middle|\; a \in X \cup \overline{X} \cup S,\, r \in S \,\right\}$. We define the size of $\mathcal{V}$ as $\|\mathcal{V}\| = \left|S^1\right| \cdot (2\,|X| + 2\,|S|) \cdot \max\left\{\, |x_{r,a}| + 1 \;\middle|\; a \in X \cup \overline{X} \cup S,\, r \in S \,\right\}$.

Whenever we talk about a group G given as a virtually free presentation, we denote $\Sigma = X \cup \overline{X} \cup S \cup \overline{S}$. The Cayley graph $\Gamma = \Gamma_\Sigma(G)$ is defined with respect to this alphabet. In particular, its degree is bounded by $\|\mathcal{V}\|$. The following lemma is easy to prove by considering the sequence of normal forms $x_0 s_0, \ldots, x_n s_n$ of a closed path: then $x_0, \ldots, x_n$ is $2\,\|\mathcal{V}\|$-triangulable in the Cayley graph of $F(X)$.

▶ **Lemma 27.** *Let G be the group defined by a virtually free presentation $\mathcal{V}$ and let Γ be its Cayley graph. Then Γ is k-triangulable for $k = 2\,\|\mathcal{V}\| + 2$.*

▶ **Lemma 28.** *Let G be the group defined by a virtually free presentation $\mathcal{V}$. Then for every finite subgroup $H \leq G$, we have $|H| \leq \left|S^1\right|$. Hence, in particular, $|H| \leq \|\mathcal{V}\|$.*

▶ **Lemma 29.** *Let G be the group defined by a virtually free presentation $\mathcal{V}$. Then the number of edges of a reduced graph of groups for G with finite vertex groups is at most $\|\mathcal{V}\|$.*

The proof of Lemma 29 is almost a verbatim repetition of the proof of [22, Thm. 1.4].

▶ **Lemma 30.** *Let G be the group defined by a virtually free presentation $\mathcal{V}$ and let Γ be its Cayley graph. Then every minimal cut in Γ is a K-cut for $K = \|\mathcal{V}\|^2$.*

The proof of Lemma 30 is based on the fact that all cuts of the form $C_x = \{\, ys \mid s \in S,\, x \text{ is a prefix of } y \,\}$ for $x \in (X \cup \overline{X})^*$ satisfy $|\vec{\delta} C_x| \leq \|\mathcal{V}\|^2$ and that all bi-infinite simple paths which are split by some cut also are split by some cut of the form C_x.

5 Main results: computing graphs of groups

▶ **Lemma 31.** *The uniform rational subset membership problem for virtually free groups given as virtually free presentation or as context-free grammar for the word problem can be decided in polynomial time. More precisely, the input is given as*

- *either a virtually free presentation $\mathcal{V}$ or a context-free grammar $\mathbb{G} = (V, \Sigma, P, S)$ for the word problem of a group G,*
- *a rational subset of G given as non-deterministic finite automaton or regular expression over Σ (in the case of a virtually free presentation Σ is defined as in Section 4),*
- *a word $w \in \Sigma^*$.*

The question is whether w is contained in the rational subset of G.

The proof of Lemma 31 is straightforward (for details see [23]): let $p : \Sigma^* \to G$ denote the canonical projection. Then $p(w) \in p(L) \iff 1 \in p(w^{-1}L) \iff \mathrm{WP}(G) \cap w^{-1}L \neq \emptyset$. The latter can be tested in polynomial time by standard facts from formal language theory.

▶ **Proposition 32.** *The following problem is in* P*: Given a virtually free group G either as virtually free presentation $\mathcal{V}$ or as context-free grammar $\mathbb{G}$ for its word problem and a graph of groups $\mathcal{G}$ over the graph Y (with vertex groups as multiplication tables, i. e., for all $g, h \in G_P$ the product gh is written down explicitly) together with a homomorphism $\varphi : \Delta^* \to \Sigma^*$ (where $\Delta = \bigcup_{P \in V(Y)} G_P \cup E(Y)$ and Σ is the alphabet for G defined by $\mathcal{V}$ (resp. $\mathbb{G}$)), decide whether φ induces an isomorphism $\pi_1(\mathcal{G}, T) \to G$.*

Proof. We verify that φ induces a homomorphism $\tilde{\varphi} : \pi_1(\mathcal{G}, T) \to G$ and that $\tilde{\varphi}$ is injective and surjective.

Testing that φ really induces a *homomorphism* reduces to the word problem for the group G, which can be solved in polynomial time: for every relation $a_1 \cdots a_m = 1$ of $\pi_1(\mathcal{G}, T)$ test whether $\varphi(a_1) \cdots \varphi(a_m) = 1$ in G. Testing that $\tilde{\varphi}$ is *surjective* reduces to polynomially many membership-problems for rational subsets of G: for all $a \in \Sigma$ test whether a is contained in the rational subset $\{ \tilde{\varphi}(g) \mid g \in \Delta \}^*$. By Lemma 31 this can be done in polynomial time.

It remains to test whether $\tilde{\varphi}$ is *injective*. Let $\pi : \Delta^* \to F(\mathcal{G})$ and $\psi : F(\mathcal{G}) \to \pi_1(\mathcal{G}, T)$ denote the canonical projections (note that ψ induces an isomorphism $\pi_1(\mathcal{G}, P) \xrightarrow{\sim} \pi_1(\mathcal{G}, T)$). Let $\mathcal{R} \subseteq \Delta^*$ denote the set of reduced words. With slight abuse of notation we use $\pi_1(\mathcal{G}, P)$ also to denote the set of words $g_0 y_1 \cdots g_{n-1} y_n g_n \in \Delta^*$ where $y_1 \cdots y_n$ is a closed path based at P and $g_i \in G_{s(y_{i+1})}$ for $0 \leq i < n$ and $g_n \in G_P$. Testing that $\tilde{\varphi}$ is injective amounts to test whether the language $L = \left(\pi^{-1}(\psi^{-1}(\tilde{\varphi}^{-1}(1))) \cap \pi_1(\mathcal{G}, P) \cap \mathcal{R}\right) \setminus \{1\} \subseteq \Delta^*$ is empty because 1 is the only reduced word in $\pi_1(\mathcal{G}, P)$ representing the identity, by Lemma 4.

Notice that $\pi^{-1}(\psi^{-1}(\tilde{\varphi}^{-1}(1))) = \varphi^{-1}(\mathrm{WP}(G))$. Since $\mathrm{WP}(G)$ is context-free (for virtually free presentations, see Lemma 26) and since context-free languages are closed under inverse homomorphism, $\varphi^{-1}(\mathrm{WP}(G))$ is a context-free language – and a pushdown automaton for it can be computed in polynomial time from the pushdown automaton for $\mathrm{WP}(G)$ (see e. g. [12]). Thus, L is a context-free language and we obtain a pushdown automaton for L, which can be tested for emptiness in polynomial time (see e. g. [12]). ◀

▶ **Theorem 33.** *The following problem is in* $\mathsf{NTIME}(2^{2^{\mathcal{O}(N)}})$*:*
Input: *a context-free grammar $\mathbb{G} = (V, \Sigma, P, S)$ in Chomsky normal form with $\|\mathbb{G}\| \leq N$ which generates the word problem of a group G,*
Compute *a graph of groups with finite vertex groups and fundamental group G.*

Note that if $\mathbb{G}$ is not in Chomsky normal form, it can be transformed into Chomsky normal form in quadratic time. In this case the graph of groups can be computed in $\mathsf{NTIME}(2^{2^{\mathcal{O}(N^2)}})$.

Table 1 Summary of appearing constants. The third column shows a bound in terms of the size of a context-free grammar $\mathbb{G}$ for the word problem (due to Lemma 8, Lemma 13, Proposition 11, and Theorem 12), the fourth column shows a bound in terms of the size of a virtually free presentation $\mathcal{V}$ (due to Lemma 27, Lemma 30, Lemma 10, Lemma 28, and Lemma 29).

N	size of the input	$\|\mathbb{G}\|$	$\|\mathcal{V}\|$
d	degree of Γ	N	N
k	triangulation constant	2^{N+2}	$2N+2$
K	maximal weight of a minimal cut	d^{3k+3}	N^2
$R = \frac{3kK}{2}$	maximal diameter of the boundary of a minimal cut	$\frac{3k}{2}d^{3k+3}$	$3(N+1)N^2$
Ξ	maximum cardinality of a finite subgroup	d^{12k+10}	N
Θ	maximum number of edges in a reduced graph of groups	d^{12k+11}	N

▶ **Theorem 34.** *The following problem is in* NP*:*
Input: *a group G as a virtually free presentation,*
Compute *a graph of groups with finite vertex groups and fundamental group G.*

The proofs of Theorem 34 and Theorem 33 are now straightforward: guess a graph of groups and a map $\varphi : \Delta \to \Sigma^*$ and use Proposition 32 to check that it induces an isomorphism. By Lemma 25 and Table 1, such a guess can be made within the time bounds of the theorems.

6 Slide moves and the isomorphism problem

Given two groups G_1 and G_2 one can calculate the respective graph of groups and then check with Krstić's algorithm by ([15]) whether their fundamental groups are isomorphic. A closer analysis shows that this algorithm runs in polynomial space. As the description is quite involved, we follow a different approach based on Forester's theory of deformation spaces [10, 4].

Let $\mathcal{G}$ be a graph of groups over Y. A *slide move* is the following operation on $\mathcal{G}$: let G_P be a vertex group and G_x, G_y edge groups with $s(x) = s(y) = P$. If G_x^x (the image of G_x in G_P) can be conjugated by an element of G_P into G_y^y i. e., there is some $g \in G_P$ such that $g^{-1}G_x^x g \leq G_y^y$, then x can be slid along y to $Q = t(y)$, i. e., $s(x)$ is changed to Q. The new inclusion of $G_x \to G_Q$ is then given by $\iota_{\overline{y}} \circ \iota_y^{-1} \circ c_g \circ \iota_x$ where ι_x is the inclusion $G_x \to G_P$ (likewise for $\iota_{\overline{y}}, \iota_y$) and c_g is the conjugation with g (i. e., $h \mapsto g^{-1}hg$). A slide move induces an isomorphism φ of the fundamental groups of the two graph of groups by $\varphi(h) = h$ for $h \in G_R$, $R \in V(Y)$, and $\varphi(z) = z$ for $z \in E(Y) \setminus \{x, \overline{x}\}$ and $\varphi(x) = gyx$. The following result is an immediate consequence of [10, Thm. 1.1] and [11, Thm. 7.2] (resp. [4, Cor. 3.5]). Since we are not aware of an explicit reference, we give the details in [23].

▶ **Proposition 35.** *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be reduced finite graphs of groups with finite vertex groups. Then $\pi_1(\mathcal{G}_1, P_1) \cong \pi_1(\mathcal{G}_2, P_2)$ if and only if $\mathcal{G}_1$ can be transformed into $\mathcal{G}_2$ by a sequence of slide moves.*

Clearly, any sequence of slide moves can be performed in linear space. By guessing a sequence of slide moves transforming $\mathcal{G}_1$ into $\mathcal{G}_2$, we obtain the following corollary.

▶ **Corollary 36.** *Given two graph of groups $\mathcal{G}_1$ and $\mathcal{G}_2$ where all vertex groups are given as full multiplication tables, it can be checked in* NSPACE$(\mathcal{O}(n))$ *whether $\pi_1(\mathcal{G}_1, P_1) \cong \pi_1(\mathcal{G}_2, P_2)$.*

In combination with Theorem 33 (and Savitch's theorem) and Theorem 34 this gives an algorithm to solve the isomorphism problem for virtually free groups:

▶ **Theorem 37.** *The isomorphism problem for context-free groups is in* DSPACE($2^{2^{\mathcal{O}(N)}}$). *More precisely, the input is given as two context-free grammars of size at most N which are guaranteed to generate word problems of groups.*

▶ **Theorem 38.** *The isomorphism problem for virtually free groups given as a virtually free presentation is in* PSPACE.

7 Conclusion and open questions

We have shown that the isomorphism problem for virtually free groups is in PSPACE (resp. DSPACE($2^{2^{\mathcal{O}(N)}}$)) depending on the type of input – thus, improving the previous bound (primitive recursive) significantly. The following questions remain open:

- What is the complexity of the isomorphism problem for virtually free groups given as an arbitrary presentation?
- Is the doubly exponential bound $n^{12\cdot 2^n+10}$ on the size of finite subgroups tight or is there a bound $2^{p(n)}$ for some polynomial p? This is closely related to another question:
- What is the minimal size of a context-free grammar of the word problem of a finite group? Can it be $\log\log(n)$ where n is the size of the group?
- Is there a polynomial bound on the number of slide moves necessary to transform two graphs of groups with isomorphic fundamental groups into each? This would lead to an NP algorithm for the isomorphism problem with virtually free presentations as input. We conjecture, however, that this is not true.

References

1. Yago Antolin. On Cayley graphs of virtually free groups. *Groups – Complexity – Cryptology*, 3:301–327, 2011. doi:10.1515/gcc.2011.012.
2. Ron Book and Friedrich Otto. *String-Rewriting Systems*. Springer-Verlag, 1993.
3. W. W. Boone. The Word Problem. *Ann. of Math.*, 70(2):207–265, 1959.
4. Matt Clay and Max Forester. Whitehead moves for G-trees. *Bull. Lond. Math. Soc.*, 41(2):205–212, 2009. doi:10.1112/blms/bdn118.
5. François Dahmani and Vincent Guirardel. The isomorphism problem for all hyperbolic groups. *Geom. Funct. Anal.*, 21(2):223–300, 2011. doi:10.1007/s00039-011-0120-0.
6. Max Dehn. Ueber unendliche diskontinuierliche Gruppen. *Math. Ann.*, 71:116–144, 1911.
7. Warren Dicks and Martin J. Dunwoody. *Groups acting on graphs*. Cambridge University Press, 1989.
8. Volker Diekert and Armin Weiß. Context-Free Groups and Their Structure Trees. *International Journal of Algebra and Computation*, 23:611–642, 2013. doi:10.1142/S0218196713500124.
9. Volker Diekert and Armin Weiß. Context-Free Groups and Bass-Serre Theory. In Juan González-Meneses, Martin Lustig, and Enric Ventura, editors, *Algorithmic and Geometric Topics Around Free Groups and Automorphisms*, Advanced Courses in Mathematics - CRM Barcelona. Birkhäuser, Basel, Switzerland, 2017. doi:10.1007/978-3-319-60940-9.
10. Max Forester. Deformation and rigidity of simplicial group actions on trees. *Geom. Topol.*, 6:219–267, 2002. doi:10.2140/gt.2002.6.219.
11. Vincent Guirardel and Gilbert Levitt. Deformation spaces of trees. *Groups Geom. Dyn.*, 1(2):135–181, 2007. doi:10.4171/GGD/8.
12. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

13 Matthias Jantzen. *Confluent String Rewriting*, volume 14 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1988.

14 A. Karrass, A. Pietrowski, and D. Solitar. Finite and infinite cyclic extensions of free groups. *Journal of the Australian Mathematical Society*, 16(04):458–466, 1973. doi:10.1017/S1446788700015445.

15 Sava Krstic. Actions of finite groups on graphs and related automorphisms of free groups. *Journal of Algebra*, 124:119–138, 1989. doi:10.1016/0021-8693(89)90154-3.

16 David E. Muller and Paul E. Schupp. Groups, the theory of ends, and context-free languages. *J. Comput. Syst. Sci.*, 26:295–310, 1983. doi:10.1016/0022-0000(83)90003-X.

17 David E. Muller and Paul E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theor. Comput. Sci.*, 37(1):51–75, 1985. doi:10.1016/0304-3975(85)90087-8.

18 P. S. Novikov. On the algorithmic unsolvability of the word problem in group theory. *Trudy Mat. Inst. Steklov*, pages 1–143, 1955. In Russian.

19 Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

20 Alan L. Selman. A taxonomy of complexity classes of functions. *J. Comput. Syst. Sci.*, 48(2):357–381, 1994. doi:10.1016/S0022-0000(05)80009-1.

21 Géraud Sénizergues. An effective version of Stalling's theorem in the case of context-free groups. In Andrzej Lingas, Rolf G. Karlsson, and Svante Carlsson, editors, *Proc. 20th International Colloquium Automata, Languages and Programming (ICALP 93), Lund (Sweden)*, volume 700 of *Lecture Notes in Computer Science*, pages 478–495. Springer-Verlag, 1993.

22 Géraud Sénizergues. On the finite subgroups of a context-free group. In Gilbert Baumslag, David Epstein, Robert Gilman, Hamish Short, and Charles Sims, editors, *Geometric and Computational Perspectives on Infinite Groups*, number 25 in DIMACS series in Discrete Mathematics and Theoretical Computer Science, pages 201–212. Amer. Math. Soc., 1996.

23 Géraud Sénizergues and Armin Weiß. The isomorphism problem for finite extensions of free groups is in PSPACE. *CoRR*, To appear soon in arXiv, 2018. URL: https://arxiv.org/abs.

24 Jean-Pierre Serre. *Trees*. Springer, 1980. French original 1977.

25 Carsten Thomassen and Wolfgang Woess. Vertex-transitive graphs and accessibility. *J. Comb. Theory Ser. B*, 58(2):248–268, 1993. doi:10.1006/jctb.1993.1042.

Unambiguous Languages Exhaust the Index Hierarchy

Michał Skrzypczak
University of Warsaw, Banacha 2, 02-097 Warsaw, Poland
mskrzypczak@mimuw.edu.pl
https://orcid.org/0000-0002-9647-4993

Abstract

This work is a study of the expressive power of unambiguity in the case of automata over infinite trees. An automaton is called unambiguous if it has at most one accepting run on every input, the language of such an automaton is called an unambiguous language. It is known that not every regular language of infinite trees is unambiguous. Except that, very little is known about which regular tree languages are unambiguous.

This paper answers the question whether unambiguous languages are of bounded complexity among all regular tree languages. The notion of complexity is the canonical one, called the (parity or Rabin-Mostowski) index hierarchy. The answer is negative, as exhibited by a family of examples of unambiguous languages the cannot be recognised by any alternating parity tree automata of bounded range of priorities.

Hardness of the examples is based on the theory of signatures, previously studied by Walukiewicz. The technical core of the article is a definition of the canonical signatures together with a parity game that compares signatures of a given pair of parity games (of the same index).

2012 ACM Subject Classification Theory of computation → Tree languages

Keywords and phrases unambiguous automata, parity games, infinite trees, index hierarchy

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.140

Related Version For a full version see `https://arxiv.org/abs/1803.06163`.

Funding This work has been supported by Poland's NSC (NCN) grant 2016/21/D/ST6/00491.

Acknowledgements The author would like to thank Szczepan Hummel, Damian Niwiński, and Igor Walukiewicz for inspiring and fruitful discussions on the topic. Moreover, the author is grateful to Bartek Klin, Kamila Łyczek, Filip Murlak, Grzegorz Rząca, and the anonymous referees for a number of editorial suggestions about the paper.

1 Introduction

Non-determinism provides a machine with a very powerful ability to *guess* its choices. Depending on the actual model, it might enhance the expressive power of the considered machines or, while preserving the class of recognised languages, make the machines more succinct or effective. All these benefits come at the cost of algorithmic difficulties when handling non-deterministic devices. This complexity motivates a search of ways of restricting the power of non-determinism. One of the most natural among these restrictions is a semantic notion called *unambiguity*: a non-deterministic machine is called *unambiguous* if it has at most one accepting run on every input.

Unambiguity turns out to be very intriguing in the context of automata theory [7]. In the classical case of finite words it does not enhance the expressive power of the automata, still it

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 140; pp. 140:1–140:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

simplifies some decision problems [16]. The situation is more complex in the case of infinite trees: the language of infinite trees labelled $\{a, b\}$ containing a letter a cannot be recognised by any unambiguous parity automaton [15, 6]. This example makes the impression that very few regular languages of infinite trees are in fact unambiguous (i.e. can be recognised by an unambiguous automaton). However, there is only a couple of distinct examples of ambiguous regular tree languages [3]. Our understanding of how many (or which) regular tree languages are unambiguous is far from being complete, in particular it is not known how to decide if a given regular tree language is unambiguous.

Another way of understanding the power of unambiguous tree languages is aimed at estimating their descriptive complexity. The complexity can be measured either in terms of the topological complexity or of the parity index, i.e. the range of priorities needed for an alternating parity tree automaton to recognise a given language. Initially, it was considered plausible that all unambiguous tree languages are co-analytic ($\mathbf{\Pi}^1_1$); that is topologically not more complex than deterministic ones. Hummel in [11] gave an example of an unambiguous language that is $\mathbf{\Sigma}^1_1$-complete, in particular not $\mathbf{\Pi}^1_1$. Further improvements [8, 12] showed that unambiguous languages reach high into the second level of the index hierarchy. However, the question whether this is an upper bound on their index complexity was left open. In this paper we prove that it is not the case, as expressed by the following theorem.

▶ **Theorem 1.** *For every $i < k$ there exists an unambiguous tree language L that cannot be recognised by any alternating parity tree automaton (*ATA*) that uses priorities $\{i, \ldots, k\}$. In other words, L does not belong to the level (i, k) of the index hierarchy.*

The canonical examples of languages lying high in the index hierarchy [4, 1] are the languages $W_{i,k}$ dating back to [9, 21] (see e.g. the formulae W_n in [1]). Unfortunately, the languages $W_{i,k}$ are not unambiguous—one can interpret the choice problem [6] in such a way that witnessing unambiguously that $t \in W_{1,2}$ would indicate an MSO-definable choice function [10, 5]. Therefore, to prove Theorem 1 we will use the following corollary of [2].

▶ **Corollary 2** ([1, 2])**.** *Let L be a set of trees. If there is a continuous function f s.t. $W_{i,k} = f^{-1}(L)$ then L cannot be recognised by an ATA of index $(i{+}1, k{+}1)$.*

Our aim is to enrich in a continuous way a given tree t with some additional information denoted $f(t)$, such that an unambiguous automaton reading $f(t)$ can verify if $t \in W_{i,k}$. Although this method is based on the topological concept of a continuous mapping f, the construction provided in this paper is purely combinatorial; the core is a definition of a parity game $\mathcal{C}_P$ that compares the difficulty of a given pair of parity games.

2 Basic notions

We use $u \cdot w$ to represent the concatenation of the two sequences. The symbol $\preceq$ stands for the prefix order. By $\omega = \{0, 1, \ldots\}$ we denote the set of natural numbers.

A *(ranked) alphabet* is a non-empty finite set A of *letters* where each letter $a \in A$ comes with its own finite *arity*. A *tree* over an alphabet A is a partial function $t \colon \omega^* \rightharpoonup A$ where the domain $\mathrm{dom}(t)$ is non-empty, prefix-closed, and if $u \in \mathrm{dom}(t)$ is a *node* with a $k$ary letter $a = t(u)$ then $u \cdot l \in \mathrm{dom}(t)$ if and only if $l < k$, i.e. u (the *father*) has *children* $u \cdot 0$, $u \cdot 1$, $\ldots$, $u \cdot (k{-}1)$. The set of all trees over A is denoted Tr_A. The element $\epsilon \in \mathrm{dom}(t)$ is called the *root* of t. A *branch* of a tree t is a sequence $\alpha \in \omega^\omega$ such that for all $n \in \omega$ the finite prefix $\alpha{\restriction}n$ is a node of t. It is easy to encode ranked alphabets using alphabets of fixed arity (or even binary), however for the technical simplicity we will work with ranked ones here.

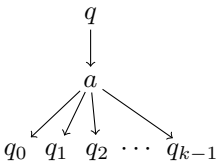


Figure 1 A representation of a transition $(q, a, q_0, \ldots, q_{k-1})$, for a $k$ary letter a.

If $u \in \mathrm{dom}(t)$ is a node of a tree t then by $t{\restriction}u$ we denote the tree $w \mapsto t(u \cdot w)$ with the domain $\{w \mid u \cdot w \in \mathrm{dom}(t)\}$. A tree of the form $t{\restriction}u$ for $u \in \mathrm{dom}(t)$ is called a *subtree* of t in u. If $a \in A$ is a $k$ary letter and $t_0, \ldots, t_{k-1}$ are trees then by $a\big(t_0, \ldots, t_{k-1}\big)$ we denote the unique tree t with $t(\epsilon) = a$ and $t{\restriction}(i) = t_i$ for $i = 0, \ldots, k{-}1$.

If $X \subseteq \mathrm{dom}(t)$ is a set of nodes of a tree t and $u \in \mathrm{dom}(t)$ then by $X{\restriction}u$ we denote the set $\{w \mid u \cdot w \in X\}$, which is a subset of $\mathrm{dom}(t{\restriction}u)$.

Automata. A *non-deterministic parity tree automaton* is a tuple $\mathcal{A} = \langle A, Q, \Delta, I, \Omega\rangle$, where A is a ranked alphabet; Q is a finite set of *states*; Δ is a finite set of *transitions*—tuples of the form $(q, a, q_0, \ldots, q_{k-1})$ where $a \in A$ is a $k$ary letter and $q, q_0, \ldots, q_{k-1}$ are states; $I \subseteq Q$ is a set of *initial states*; and $\Omega\colon Q \to \omega$ is a *priority* mapping.

A *run* of an automaton $\mathcal{A}$ over a tree t over the alphabet A is a function $\rho\colon \mathrm{dom}(t) \to Q$ such that $\rho(\epsilon) \in I$ and for every $u \in \mathrm{dom}(t)$ with a $k$ary letter $a = t(u)$, the tuple $\big(\rho(u), a, \rho(u \cdot 0), \ldots, \rho(u \cdot (k{-}1))\big)$ is a transition in Δ. A run ρ is *accepting* if for every branch α of t the lowest priority of the states appearing infinitely many times along α (i.e. $\liminf_{n\to\infty} \Omega\big(\rho(\alpha{\restriction}n)\big)$) is even. An automaton $\mathcal{A}$ *accepts* a tree t if there exists an accepting run of $\mathcal{A}$ over t. The language of an automaton $\mathcal{A}$ (denoted $\mathrm{L}(\mathcal{A})$) is the set of trees accepted by $\mathcal{A}$. A set of trees over an alphabet A is called *regular* if it is recognised by a non-deterministic parity tree automaton. For a detailed introduction to the theory of automata over infinite trees, see [18].

An automaton $\mathcal{A}$ is *unambiguous* if for every tree t there exists at most one accepting run of $\mathcal{A}$ over t. An automaton $\mathcal{A}$ is *deterministic* if $I = \{q_{\mathrm{I}}\}$ is a singleton and for every $q \in Q$ and $k$ary letter $a \in A$ it has at most one transition of the form $(q, a, q_0, \ldots, q_{k-1})$ in Δ. A language is *unambiguous* (resp. *deterministic*) if it can be recognised by an unambiguous (resp. deterministic) automaton. Clearly each deterministic automaton is unambiguous but the converse is not true. Due to [6] we know that there are regular tree languages that are ambiguous (i.e. not unambiguous).

Games. A *game* with players $\mathbf{1}$ and $\mathbf{2}$ is a tuple $\mathcal{G} = \langle V, E, v_{\mathrm{I}}, W\rangle$ where: $V = V_{\mathbf{1}} \sqcup V_{\mathbf{2}}$ is a set of *positions* split into the $\mathbf{1}$*-positions* $V_{\mathbf{1}}$ and $\mathbf{2}$*-positions* $V_{\mathbf{2}}$; $E \subseteq V \times V$ is a set of *edges*; $v_{\mathrm{I}} \in V$ is an *initial position*; and $W \subseteq V^\omega$ is a *winning condition*. We will denote by P the players, i.e. $P \in \{\mathbf{1}, \mathbf{2}\}$, $\bar{P}$ is the opponent of P. For $v \in V$ by $v \cdot E$ we denote the set of successors $\{v' \mid (v, v') \in E\}$. We assume that for each $v \in V$ the set $v \cdot E$ is non-empty. A non-empty finite or infinite sequence $\Pi \in V^{\leq\omega}$ is a *play* if $\Pi(0) = v_{\mathrm{I}}$ and for each $0 < i < |\Pi|$ there is an edge $\big(\Pi(i{-}1), \Pi(i)\big)$. Notice that if $\langle V, E\rangle$ is a tree then there is an equivalence between finite plays and positions $v \in V$. An infinite play Π is *winning for* $\mathbf{1}$ if $\Pi \in W$; otherwise Π is winning for $\mathbf{2}$.

A non-empty and prefix-closed set of plays Σ with no $\preceq$-maximal element (i.e. no leaf) is called a *behaviour*. We call a behaviour P*-full* if for every play $(v_0, \ldots, v_n) \in \Sigma$ with $v_n \in V_P$ and all $v' \in v \cdot E$ we have $(v_0, \ldots, v_n, v') \in \Sigma$. We call a behaviour P*-deterministic* if for every play $(v_0, \ldots, v_n) \in \Sigma$ with $v_n \in V_P$ there is a unique $v' \in v \cdot E$ such that $(v_0, \ldots, v_n, v') \in \Sigma$.

A *quasi-strategy* of a player P is a behaviour that is $\bar{P}$-full. A *strategy* of P is a quasi-strategy of P that is P-deterministic. A quasi-strategy is *positional* if the fact whether a play $(v_0, \ldots, v_n, v_{n+1})$ belongs to Σ depends only on v_n.

A *partial strategy* of P is a P-deterministic behaviour—it defines the unique choices of P but may not respond to some choices of $\bar{P}$. We say that a play $(v_0, \ldots, v_n, v_{n+1}) \notin \Sigma$ is *not reachable* by a partial strategy Σ if $(v_0, \ldots, v_n) \in \Sigma$ and $v_n \in V_{\bar{P}}$. If Σ is a (partial) strategy of P and $(v_0, \ldots, v_n, v') \in \Sigma$ with $v_n \in V_P$ then we say that Σ *moves to* v' in $(v_0, \ldots, v_n)$.

A strategy Σ of P is *winning* if every *infinite play of* Σ (i.e. Π such that $\forall n \in \omega.\ \Pi{\restriction}n \in \Sigma$) is winning for P. A game is (*positionally*) *determined* if one of the players has a (positional) winning strategy. We say that a position v of a game $\mathcal{G}$ is *winning for* P (resp. *losing for* P) if P (resp. $\bar{P}$) has a winning strategy in the game $\mathcal{G}$ with $v_{\mathrm{I}} := v$.

Topology. In this work we use only basic notions of descriptive set theory and topology, see [13, 19] for a broader introduction. The space Tr_A with the product topology is homeomorphic to the Cantor space. One can take as the basis of this topology the sets of the form $\{t \in \mathrm{Tr}_A \mid t(u_1){=}a_1, t(u_2){=}a_2, \ldots, t(u_n){=}a_n\}$ for finite sequences $(u_1, a_1, u_2, a_2, \ldots, u_n, a_n)$. The open sets in Tr_A are obtained as unions of basic open sets. A function $f\colon X \to Y$ is continuous if the pre-image of each basic open set in Y is open in X.

3 The languages

Let us fix a pair of numbers $i < k$. Our aim is to encode a general parity game with players $\mathbf{1}$ and $\mathbf{2}$ and priorities $\{i, \ldots, k\}$ as a tree over a fixed ranked alphabet $A_{(i,k)}$. That alphabet consists of: unary symbols $[i], [i{+}1], \ldots, [k]$ indicating priorities of positions; and binary symbols $\langle\mathbf{1}\rangle$ and $\langle\mathbf{2}\rangle$ which leave the choice of the subtree to the respective player.

The game induced by a tree $t \in \mathrm{Tr}_{A_{(i,k)}}$ is denoted $\mathcal{G}(t)$. The set of positions of $\mathcal{G}(t)$ is $\mathrm{dom}(t)$ and the edge relation contains pairs father—child. The initial position is ϵ and a position $v \in \mathrm{dom}(t)$ is a $\mathbf{1}$-position iff $t(v) = \langle\mathbf{1}\rangle$. An infinite play of that game is won by $\mathbf{1}$ if and only if the minimal priority j that occurs infinitely often during the play is even[1]. Since the graph of $\mathcal{G}(t)$ is a tree, we identify finite plays in $\mathcal{G}(t)$ with positions $v \in \mathrm{dom}(t)$. Therefore, (quasi / partial) strategies in $\mathcal{G}(t)$ can be seen as specific subsets $\Sigma \subseteq \mathrm{dom}(t)$ and infinite plays of these strategies as branches of t.

For a player $P \in \{\mathbf{1}, \mathbf{2}\}$ the language $\mathrm{W}_{P,(i,k)}$ contains a tree t if P has a winning strategy in $\mathcal{G}(t)$. It is easy to see that $\mathrm{W}_{\mathbf{1},(i,k)}$ is homeomorphic (i.e. topologically equivalent) to $W_{i,k}$ from [21] (the case of $P = \mathbf{2}$ is dual, we put $W_{i+1,k+1}$ then).

As it turns out, the languages $\mathrm{W}_{P,(i,k)}$ are not expressive enough to allow enrichment of a tree t into $f(t)$, see Proposition 22. To enlarge their expressive power we will extend the alphabet with a unary symbol $\sim$ that will correspond to a swap of the players in $\mathcal{G}(t)$. The enhanced alphabet will be denoted $A^{\sim}_{(i,k)}$. We will say that a tree t over the alphabet $A^{\sim}_{(i,k)}$ is *well-formed* if there is no branch with infinitely many symbols $\sim$.

Consider a node $u \in \mathrm{dom}(t)$ in a well-formed tree t over $A^{\sim}_{(i,k)}$. We will say that u is *switched* if there is an odd number of nodes $w \prec u$ such that $t(w) = \sim$. Otherwise u is *kept*. These notions represent the fact that each symbol $\sim$ swaps the players in $\mathcal{G}(t)$.

[1] We restrict our attention to the trees in which every second symbol on each branch is a unary symbol representing a priority (i.e. $[j]$ for $j \in \{i, \ldots, k\}$); every tree can implicitly be transformed into that format by padding with the maximal priority k (such a padding does not influence the winner of $\mathcal{G}(t)$).

If a tree t is well-formed then the game $\mathcal{G}(t)$ is well-defined in a similar way as above, a *kept* position $v \in \mathrm{dom}(t)$ is a $\mathbf{1}$-position iff $t(v) = \langle \mathbf{1} \rangle$; a *switched* position $v \in \mathrm{dom}(t)$ is a $\mathbf{1}$-position iff $t(v) = \langle \mathbf{2} \rangle$. The language $\mathrm{W}^{\sim}_{P,(i,k)}$ contains a well-formed tree t over $A^{\sim}_{(i,k)}$ if P has a winning strategy in $\mathcal{G}(t)$. The games $\mathcal{G}(t)$ have a parity winning condition and therefore are determined [9, 14], so we obtain:

▶ **Fact 3.** *If* $t \in \mathrm{Tr}_{A^{\sim}_{(i,k)}}$ *is well-formed then* $t \in \mathrm{W}^{\sim}_{P,(i,k)}$ *iff* $t \notin \mathrm{W}^{\sim}_{\bar{P},(i,k)}$ *iff* $\sim(t) \in \mathrm{W}^{\sim}_{\bar{P},(i,k)}$.

The additional information added by f will be kept under third children of new ternary variants of the symbols $\langle \mathbf{1} \rangle$ and $\langle \mathbf{2} \rangle$, denoted $\langle \mathbf{1}+ \rangle$ and $\langle \mathbf{2}+ \rangle$. Let the alphabet $A^{+\sim}_{(i,k)}$ be $A^{\sim}_{(i,k)}$ where instead of the symbols $\langle \mathbf{1} \rangle$ and $\langle \mathbf{2} \rangle$ we have $\langle \mathbf{1}+ \rangle$ and $\langle \mathbf{2}+ \rangle$ respectively.

Consider a tree r over the extended alphabet $A^{+\sim}_{(i,k)}$. By $\mathrm{shave}(r)$ we denote the tree over the non-extended alphabet $A^{\sim}_{(i,k)}$, where instead of each subtree of the form $\langle \mathbf{1}+ \rangle(t_{\mathtt{L}}, t_{\mathtt{R}}, t_2)$ one puts the subtree $\langle \mathbf{1} \rangle(t_{\mathtt{L}}, t_{\mathtt{R}})$; the same for $\langle \mathbf{2}+ \rangle$ and $\langle \mathbf{2} \rangle$. Notice that $\mathrm{dom}(\mathrm{shave}(r)) \subseteq \mathrm{dom}(r)$ and the labels of $\mathrm{shave}(r)$ correspond to the labels of r in the respective nodes (up to the additional $+$ in r). We will say that a tree r over the alphabet $A^{+\sim}_{(i,k)}$ is *well-formed* if for every its subtree $r' = r{\restriction}u$ the tree $\mathrm{shave}(r')$ is well-formed in the standard sense. In other words, r is well-formed if there is no branch of r that contains infinitely many symbols $\sim$ but only finitely many directions 2 (the direction 2 corresponds to moving outside $\mathrm{shave}(r')$).

We are now in position to define the witnesses proving Theorem 1. For that we will define an unambiguous automaton $\mathcal{U}$ recognising certain language of trees over the alphabet $A^{+\sim}_{(i,k)}$. In this section we will prove that $\mathcal{U}$ is unambiguous. In the rest of the article we show that $\mathcal{U}$ (with a restricted set of initial states) recognises a language high in the index hierarchy.

▶ **Definition 4.** The set of states of $\mathcal{U}$ is $\{0, \ldots, k{+}2\} \times \{\mathbf{1}, \mathbf{2}\} \times \{\mathtt{D}, \mathtt{L}, \mathtt{R}\}$. Let $\Omega(j, P, d) = j$. The transitions of $\mathcal{U}$ are depicted in Figure 2. The set of initial states of $\mathcal{U}$ contains all the states of the form $(0, \star, \star)$ (recall that as in Figure 2, $\star$ represents all the possible choices).

Intuitively, the first coordinate of a state q of $\mathcal{U}$ is its priority; the second coordinate is the winner for $\mathcal{G}(\mathrm{shave}(r'))$ for the current subtree r'; while the third coordinate indicates the actual strategy if there is ambiguity and $\mathtt{D}$ otherwise. The transition over $\sim$ represents that $\sim$ swaps the players; the next two transitions correspond to positions that are not controlled by a (claimed) winner P over a given subtree; and the last two transitions correspond to a position that is controlled by P. In the lower two transitions the choice of a direction $\mathtt{L}$ or $\mathtt{R}$ depends on the declared winner P in the third child of the current node.

Consider a run ρ of $\mathcal{U}$ over a tree r, let $u \in \mathrm{dom}(r)$, and assume that $\rho(u)$ is of the form $(\star, P, \star)$. In that case one can extract from the third coordinates of ρ a strategy Σ of P in $\mathcal{G}(\mathrm{shave}(r{\restriction}u))$ that will be called the *ρ-strategy in u*. This strategy is defined inductively, preserving the invariant that for each $w \in \Sigma$ the node w is *kept* in $\mathrm{shave}(r{\restriction}u)$ if and only if $\rho(u \cdot w)$ is of the form $(\star, P, \star)$. We start with Σ containing the initial position ϵ. Now consider a position w in Σ. If w is controlled by P (i.e. $r(u \cdot w) = \langle P+ \rangle$ for w *kept* and $\langle \bar{P}+ \rangle$ for w *switched*) then the strategy Σ moves to the position $w \cdot 0$ (resp. $w \cdot 1$) if the state $\rho(u \cdot w) = (\star, \star, d)$ satisfies $d = \mathtt{L}$ (resp. $d = \mathtt{R}$). In the positions $w \in \Sigma$ not controlled by P the strategy Σ has no choice and contains all the children of w in $\mathrm{shave}(r{\restriction}u)$. It is easy to check that the transitions of $\mathcal{U}$ guarantee that the invariant is preserved.

▶ **Lemma 5.** *Let ρ be a run of $\mathcal{U}$ over r. Then ρ is accepting if and only if r is well-formed and for every $u \in \mathrm{dom}(r)$ the ρ-strategy in u is winning in $\mathcal{G}\big(\mathrm{shave}(r{\restriction}u)\big)$.*

Proof. First observe that $\mathcal{U}$ uses states of priority 0 and 1 to deterministically verify that r is well-formed. Consider a well-formed tree r and a run ρ. On the branches following the

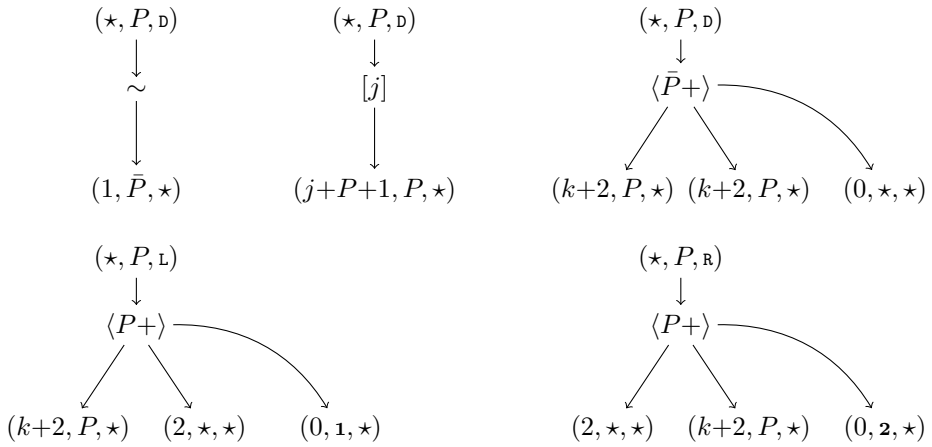


Figure 2 The transitions of the automaton $\mathcal{U}$, where $P \in \{\mathbf{1}, \mathbf{2}\}$ stands for a player; $j \in \{i, \ldots, k\}$ is a priority; and $\star$ represents all the possible choices on a given coordinate.

ρ-strategies, the priorities of ρ correspond to the priorities visited in $\mathcal{G}\big(\mathrm{shave}(r\restriction u)\big)$ (up to a shift by 1 or 2 depending on the current second coordinate of ρ). Thus, ρ satisfies the parity condition on all the branches of r if and only if all the ρ-strategies are winning. ◀

The following fact follows directly from the above lemma.

▶ **Fact 6.** *Assume that ρ is an accepting run of $\mathcal{U}$ over a tree r, $u \in \mathrm{dom}(r)$, and $\rho(u)$ is of the form $(\star, P, \star)$. Then P wins $\mathcal{G}\big(\mathrm{shave}(r\restriction u)\big)$. In particular, if ρ and ρ' are two accepting runs of $\mathcal{U}$ over the same tree r then the second coordinates of ρ and ρ' are equal.*

▶ **Lemma 7.** *If ρ and ρ' are two runs of $\mathcal{U}$ (possibly not accepting) over a tree r and the second coordinates of ρ and ρ' are equal then $\rho = \rho'$.*

Proof. The third coordinate of a run in $u \in \mathrm{dom}(r)$ depends on $r(u)$, the second coordinate of the run in u, and (if one of the lower two transitions from Figure 2 is used) the second coordinate of the run in $u \cdot 2$. Thus, the third coordinates of ρ and ρ' must agree.

The first coordinates of ρ and ρ' in the root are 0. Consider a node u and its child $u' \in \mathrm{dom}(r)$. The first coordinate of a run in u' depends on $r(u)$ and the last two coordinates of the run in u. Therefore, also the first coordinates of ρ and ρ' must agree. ◀

▶ **Definition 8.** Take $P \in \{\mathbf{1}, \mathbf{2}\}$ and let $L_{P,(i,k)}$ be the language recognised by the automaton $\mathcal{U}$ with the set of initial states restricted to the states of the form $(0, P, \star)$.

Fact 6 together with Lemma 7 imply that the languages $L_{P,(i,k)}$ are unambiguous. Thus, to complete the proof of Theorem 1, one needs to prove that the languages $L_{P,(i,k)}$ climb up the index hierarchy. This will be done using Corollary 2 in the next section.

4 Construction of f

Fix a pair $i < k$. In this section we will prove the following proposition.

▶ **Proposition 9.** *There exists a continuous function $f\colon \mathrm{Tr}_{A^{\sim}_{(i,k)}} \to \mathrm{Tr}_{A^{+\sim}_{(i,k)}}$ such that:*

- *If t is well-formed then $f(t)$ is also well-formed.*
- *For a well-formed t and a player P we have: $t \in \mathrm{W}^{\sim}_{P,(i,k)}$ if and only if $f(t) \in L_{P,(i,k)}$.*

Before proving that proposition, notice that a tree over the alphabet $A_{(i,k)}$ can be seen as a well-formed tree over $A^{\sim}_{(i,k)}$ and $\mathrm{W}^{\sim}_{\mathbf{1},(i,k)} \cap \mathrm{Tr}_{A_{(i,k)}} = \mathrm{W}_{\mathbf{1},(i,k)}$ which is topologically equivalent to $W_{i,k}$. Therefore, the above proposition implies that $f{\restriction}\mathrm{Tr}_{A_{(i,k)}}$ satisfies the assumptions of Corollary 2. As $i < k$ are arbitrary, Theorem 1 follows.

To properly define f we first need to introduce a notion that allows to compare how much a player P *prefers* one tree over another. This is achieved by assigning to each tree its *P-signature* and proving that moving to a subtree with an optimal signature guarantees that the player wins whenever possible. The theory of signatures comes from [17] and results of Walukiewicz, e.g. [20, 21]. The notion of signatures used here is more demanding than the classical ones, as we require equalities instead of inequalities in the invariants from Lemma 10.

We say that a number $j \in \omega$ is *P-losing* if $i \leq j \leq k$ and j is odd (resp. even) if $P = \mathbf{1}$ (resp. $P = \mathbf{2}$). A number $j \in \{i, \dots, k\}$ that is not P-losing is called *P-winning*. A P-signature is either ∞ or a tuple of countable ordinals $(\theta_{i'}, \theta_{i'+2}, \dots, \theta_{k'})$, indexed by P-losing numbers—i' is the minimal and k' is the maximal P-losing number. P-signatures that are not ∞ are well-ordered by the lexicographic order $\leq_{\mathrm{lex}}$ in which the ordinals with smaller indices are more important. Let ∞ be the maximal element of $\leq_{\mathrm{lex}}$.

A $\mathbf{1}$-signature $\sigma_{\mathbf{1}}(t) = (42)$ with $i = 0$ and $k = 2$ means that the best what player $\mathbf{1}$ can hope for is to visit at most 42 times a [1]-node (possibly interleaved with nodes of priority 2) before the first [0]-node is visited (if ever). After visiting a [0]-node, the $\mathbf{1}$-signature of the subtree may grow, starting again a counter of nodes of priority 1 to be visited. The $\mathbf{1}$-signature (ω) means that $\mathbf{2}$ can choose a finite number of [1]-nodes that will be visited; however the choice needs to be done before the first such node is seen.

The following two lemmas express the crucial properties of the signatures.

▶ **Lemma 10.** *There exists a unique point-wise minimal pair of assignments $t \mapsto \sigma_P(t)$ for $P \in \{\mathbf{1}, \mathbf{2}\}$ that assign to each well-formed tree t over $A^{\sim}_{(i,k)}$ a P-signature $\sigma_P(t)$ such that:*

1. *$\sigma_P(t) = \infty$ if and only if P loses $\mathcal{G}(t)$;*
2. *$\sigma_P\big(\sim(t)\big) = (0, \dots, 0)$ if P wins $\mathcal{G}\big(\sim(t)\big)$ (i.e. $\bar{P}$ wins $\mathcal{G}(t)$);*
3. *$\sigma_P\big([j](t)\big) = (\theta_{i'}, \dots, \theta_{j-1}, 0, 0 \dots, 0)$ if $\sigma_P(t) = (\theta_{i'}, \dots, \theta_{k'})$ and j is P-winning;*
4. *$\sigma_P\big([j](t)\big) = (\theta_{i'}, \dots, \theta_{j-2}, \theta_j{+}1, 0, \dots, 0)$ if $\sigma_P(t) = (\theta_{i'}, \dots, \theta_{k'})$ and j is P-losing;*
5. *$\sigma_P\big(\langle P\rangle(t_{\mathtt{L}}, t_{\mathtt{R}})\big) = \min\big\{\sigma_P(t_{\mathtt{L}}), \sigma_P(t_{\mathtt{R}})\big\}$;*
6. *$\sigma_P\big(\langle \bar{P}\rangle(t_{\mathtt{L}}, t_{\mathtt{R}})\big) = \max\big\{\sigma_P(t_{\mathtt{L}}), \sigma_P(t_{\mathtt{R}})\big\}$.*

Let us fix the functions σ_P for $P \in \{\mathbf{1}, \mathbf{2}\}$ as above. Consider a well-formed tree t over the alphabet $A^{\sim}_{(i,k)}$. We say that a strategy Σ of a player P in $\mathcal{G}(t)$ is *optimal* (or σ-optimal) if:

- In a position $u \in \mathrm{dom}(t)$ that is *kept* in t and $t{\restriction}u = \langle P\rangle\big(t_{\mathtt{L}}, t_{\mathtt{R}}\big)$ the strategy Σ moves to a subtree of a minimal value of σ_P; i.e. Σ can move to $u \cdot 0$ if $\sigma_P(t_{\mathtt{L}}) \leq_{\mathrm{lex}} \sigma_P(t_{\mathtt{R}})$ and to $u \cdot 1$ if $\sigma_P(t_{\mathtt{L}}) \geq_{\mathrm{lex}} \sigma_P(t_{\mathtt{R}})$. If the values $\sigma_P(t_{\mathtt{L}})$ and $\sigma_P(t_{\mathtt{R}})$ are equal then Σ can move in any of the two directions.
- In a position u that is *switched* in t and $t{\restriction}u = \langle \bar{P}\rangle\big(t_{\mathtt{L}}, t_{\mathtt{R}}\big)$ the strategy Σ uses the same rule as above but uses the function $\sigma_{\bar{P}}$ to compare the $\bar{P}$-signatures of the subtrees.

Notice that according to the above definition there might be more than one optimal strategy.

▶ **Lemma 11.** *If $t \in \mathrm{W}^{\sim}_{P,(i,k)}$ and Σ is an optimal strategy of P in $\mathcal{G}(t)$ then Σ is winning.*

The following lemma claims the combinatorial core of this article: it shows that one can compare the P-signatures using a continuous reduction to the languages $\mathrm{W}^{\sim}_{P,(i,k)}$.

▶ **Lemma 12.** *There exists a continuous function $c_P \colon \big(\mathrm{Tr}_{A^{\sim}_{(i,k)}}\big)^2 \to \mathrm{Tr}_{A^{\sim}_{(i,k)}}$ such that if $t_{\mathtt{L}}$ and $t_{\mathtt{R}}$ are well-formed then so is $c_P(t_{\mathtt{L}}, t_{\mathtt{R}})$ and additionally*

$$c_P(t_{\mathtt{L}}, t_{\mathtt{R}}) \in \mathrm{W}^{\sim}_{\mathbf{1},(i,k)} \quad \text{if and only if} \quad \sigma_P(t_{\mathtt{L}}) \leq_{\mathrm{lex}} \sigma_P(t_{\mathtt{R}}).$$

The rest of this section demonstrates Proposition 9. Lemma 12 is proved in the next section.

Consider a function $f\colon \mathrm{Tr}_{A^{\sim}_{(i,k)}} \to \mathrm{Tr}_{A^{+\sim}_{(i,k)}}$ defined recursively as:

$$\begin{aligned} f\big(\langle P\rangle(t_{\mathtt{L}}, t_{\mathtt{R}})\big) &= \langle P{+}\rangle\Big(f(t_{\mathtt{L}}), f(t_{\mathtt{R}}), f\big(c_P(t_{\mathtt{L}}, t_{\mathtt{R}})\big)\Big) && \text{for } P \in \{\mathtt{1}, \mathtt{2}\},\\ f\big(\sim(t)\big) &= \sim\big(f(t)\big), &&\\ f\big([j](t)\big) &= [j]\big(f(t)\big) && \text{for } j \in \{i, \dots, k\}. \end{aligned}$$

Clearly by the definition of f we know that $\mathrm{shave}\big(f(t)\big) = t$. Additionally, $f(t)$ is defined recursively using c_P which is continuous, therefore f is also continuous. As c_P maps well-formed trees to well-formed trees, so does f.

First assume that $f(t) \in L_{P,(i,k)}$ as witnessed by an accepting run ρ of $\mathcal{U}$ over $f(t)$ with $\rho(\epsilon) = (\star, P, \star)$. Fact 6 says that P wins $\mathcal{G}\big(\mathrm{shave}(f(t))\big) = \mathcal{G}(t)$, so $t \in \mathrm{W}^{\sim}_{P,(i,k)}$.

For the converse assume that P_0 wins $\mathcal{G}(t)$ for a well-formed tree t over $A^{\sim}_{(i,k)}$.

▶ **Lemma 13.** *If t is well-formed then there exists a unique run ρ of $\mathcal{U}$ over $f(t)$ such that for every $u \in \mathrm{dom}\big(f(t)\big)$ we have*

$$\rho(u) = (\star, P, \star) \quad \text{if and only if} \quad P \text{ wins } \mathcal{G}\big(\mathrm{shave}(f(t)\restriction u)\big). \tag{1}$$

Moreover, all the ρ-strategies are winning for the respective players.

Proof. The construction of ρ is inductive from the root preserving (1). The only ambiguity when choosing transitions of $\mathcal{U}$ is when we reach a node $w \in \mathrm{dom}\big(f(t)\big)$ such that $\rho(w)$ is of the form $(\star, P, \star)$ and $f(t)\restriction w = \langle P{+}\rangle\big(f(t_{\mathtt{L}}), f(t_{\mathtt{R}}), f(c_P(t_{\mathtt{L}}, t_{\mathtt{R}}))\big)$. We choose either the left or the right of the two lower transitions of $\mathcal{U}$ depending on the winner in $\mathcal{G}\big(\mathrm{shave}(f(t)\restriction w \cdot 2)\big)$ in such a way to satisfy (1) for $u = w \cdot 2$. By the symmetry assume that we used the left transition. This leaves undeclared the second coordinate of $\rho(w \cdot 1)$ (resp. $\rho(w \cdot 0)$ in the case of the right transition). Again we declare this coordinate accordingly to (1). We need to check that (1) is also satisfied for $u = w \cdot 0$ (resp. $u = w \cdot 1$) i.e. that P wins $\mathcal{G}\big(\mathrm{shave}(f(t)\restriction w \cdot 0)\big)$. To see that, we notice that the following conditions are equivalent (★):

- $\rho(w) = (\star, \star, \mathtt{L})$ [by the form of the transitions of $\mathcal{U}$]
- $\rho(w \cdot 2) = (\star, \mathtt{1}, \star)$ [by the definition of ρ]
- $\mathtt{1}$ wins in $\mathcal{G}\big(\mathrm{shave}(f(t)\restriction(w \cdot 2))\big)$ [by the form of $f(t)\restriction(w \cdot 2)$]
- $\mathtt{1}$ wins in $\mathcal{G}\big(\mathrm{shave}(f(c_P(t_{\mathtt{L}}, t_{\mathtt{R}})))\big)$ [by the equality $\mathrm{shave}\big(f(t')\big) = t'$]
- $\mathtt{1}$ wins in $\mathcal{G}\big(c_P(t_{\mathtt{L}}, t_{\mathtt{R}})\big)$ [by Lemma 12]
- $\sigma_P(t_{\mathtt{L}}) \leq_{\mathrm{lex}} \sigma_P(t_{\mathtt{R}})$.

Thus, if we choose the lower left transition of $\mathcal{U}$, we know that $\sigma_P(t_{\mathtt{L}}) \leq_{\mathrm{lex}} \sigma_P(t_{\mathtt{R}})$. By the inductive invariant we know that $\infty >_{\mathrm{lex}} \sigma_P\big(\mathrm{shave}(f(t)\restriction w)\big) = \sigma_P\big(\langle P\rangle(t_{\mathtt{L}}, t_{\mathtt{R}})\big)$. Therefore, Item 5 of Lemma 10 implies that $\sigma_P(t_{\mathtt{L}}) <_{\mathrm{lex}} \infty$ so in fact P wins $\mathcal{G}(t_{\mathtt{L}}) = \mathcal{G}\big(\mathrm{shave}(f(t_{\mathtt{L}}))\big) = \mathcal{G}\big(\mathrm{shave}(f(t)\restriction w \cdot 0)\big)$. Thus, the invariant (1) is also preserved for $u = w \cdot 0$. This concludes the inductive definition of ρ. Lemma 7 implies uniqueness.

Take $u \in \mathrm{dom}\big(f(t)\big)$ with $\rho(u) = (\star, P, \star)$. Let Σ be the ρ-strategy in u and $r' = f(t)\restriction u$. Consider a node $w \in \mathrm{dom}\big(\mathrm{shave}(r')\big)$ such that $\mathrm{shave}(r')(w) = \langle P'\rangle$ with $P' = P$ if w is *kept* and $P' = \bar{P}$ if w is *switched*. In both cases $f(t)(u \cdot w) = \langle P'{+}\rangle$. By the above equivalence (★) and the definition of a ρ-strategy, Σ makes a σ-optimal move in w. Therefore, Σ is optimal. Invariant (1) says that P wins $\mathcal{G}\big(\mathrm{shave}(r')\big)$, so Lemma 11 implies that Σ is winning. ◀

Fix the run ρ given by the above lemma. Since $\mathcal{G}(t) = \mathcal{G}\big(\mathrm{shave}(f(t))\big)$, $\rho(\epsilon) = (\star, P_0, \star)$. Since all the ρ-strategies are winning, ρ is accepting by Lemma 5 and $f(t) \in L_{P,(i,k)}$. This concludes the proof of Theorem 1 assuming that Lemma 12 holds.

5 Comparing signatures

We will now prove Lemma 12 by defining, given two trees p and s over $A^{\sim}_{(i,k)}$, a game $\mathcal{C}_P(p,s)$. To indicate the difference between the games $\mathcal{C}_P$ and $\mathcal{G}$, we denote the players of $\mathcal{C}_P$ as $\exists = \mathbf{1}$ and $\forall = \mathbf{2}$. The purpose of the game $\mathcal{C}_P$ will be to ensure that $\exists$ wins $\mathcal{C}_P(p,s)$ if and only if $\sigma_P(p) \leq_{\text{lex}} \sigma_P(s)$. The winning condition of the game $\mathcal{C}_P$ will be a parity condition, however, the game will allow certain *lookahead* (see steps (EL) and (AL) in the definition of $\mathcal{C}_P$). Then, the function c_P will just unravel the game $\mathcal{C}_P(p,s)$ into a tree over the alphabet $A^{\sim}_{(i,k)}$.

If i' and k' are the minimal and maximal P-losing numbers; j is a P-losing number; and $\sigma = (\theta_{i'}, \theta_{i'+2}, \dots, \theta_{k'})$ is a P-signature then $\sigma{\restriction}j$ is the tuple $(\theta_{i'}, \theta_{i'+2}, \dots, \theta_j)$. For completeness let $\infty{\restriction}j \stackrel{\text{def}}{=} \infty$. Clearly $\sigma{\restriction}k' = \sigma$. Notice that $\sigma{\restriction}j$ is also a P-signature (with $k = j$) and moreover if $\sigma \leq_{\text{lex}} \sigma'$ then $\sigma{\restriction}j \leq_{\text{lex}} \sigma'{\restriction}j$.

A position of the game $\mathcal{C}_P$ is a triple $\big(p, s, \ell\big)$ where $p, s \in \mathrm{Tr}_{A^{\sim}_{(i,k)}}$ and ℓ is a P-losing number. As $\mathcal{C}_P(p,s)$ we denote the game $\mathcal{C}_P$ with the initial position set to $\big(p, s, k'\big)$. The game is designed in such a way to guarantee the following claim.

▶ **Claim 14.** *A position $\big(p, s, \ell\big)$ is winning for $\exists$ in $\mathcal{C}_P$ if and only if*

$$\sigma_P(p){\restriction}\ell \;\leq_{\text{lex}}\; \sigma_P(s){\restriction}\ell. \tag{2}$$

A single round of the game $\mathcal{C}_P$ consists of a sequence of choices done by the players. It is easy to encode such a sequence using additional intermediate positions of the game. For the sake of readability, we do not specify these positions explicitly. Instead, a round moves the game from a position $\big(p, s, \ell\big)$ into a new position according to the following sequential steps:

(EL) $\exists$ can claim that P loses $\mathcal{G}(s)$. If she does so, the game ends and $\exists$ wins iff $s \notin \mathrm{W}^{\sim}_{P,(i,k)}$.

(AL) $\forall$ can claim that P loses $\mathcal{G}(p)$. If he does so, the game ends and $\forall$ wins iff $p \notin \mathrm{W}^{\sim}_{P,(i,k)}$.

(EI) $\exists$ can modify ℓ into another P-losing number $\ell' < \ell$. In that case the round ends and the next position is $\big(p', s, \ell'\big)$ where $p' = [\ell'](p)$.

(AI) $\forall$ can modify ℓ into another P-losing number $\ell' < \ell$. In that case the round ends and the next position is $\big(p, s, \ell'\big)$.

(↓) If $p = [\ell](p')$ and $s = [\ell](s')$ then the round ends and the next position is (p', s', ℓ).

(↓p) If p is not of the form $[\ell](p'')$ then a step called $\mathit{Step}\exists(p)$ is done, resulting in an immediate win of $\exists$ or a new tree p'. The round ends and the next position is (p', s, ℓ).

(↓s) Otherwise p is of the form $[\ell](p')$ and a step called $\mathit{Step}\forall(s)$ is done, resulting in an immediate win of $\forall$ or a new tree s'. The round ends and the next position is (p, s', ℓ).

The result p' of $\mathit{Step}\exists(p)$ depends on the form of p as follows:

- If $p = \langle P\rangle(p_{\mathtt{L}}, p_{\mathtt{R}})$ then $\exists$ chooses to set $p' = p_{\mathtt{L}}$ or $p' = p_{\mathtt{R}}$.
- If $p = \langle \bar{P}\rangle(p_{\mathtt{L}}, p_{\mathtt{R}})$ then $\forall$ chooses to set $p' = p_{\mathtt{L}}$ or $p' = p_{\mathtt{R}}$.
- If $p = [j](p')$ and $j > \ell$ then p' is defined and that round of $\mathcal{C}_P$ has priority $j{+}1{-}P$.
- Otherwise $\exists$ immediately wins.

Dually, the result s' of $\mathit{Step}\forall(s)$ depends on the form of s as follows:

- If $s = \langle P\rangle(s_{\mathtt{L}}, s_{\mathtt{R}})$ then $\forall$ chooses to set $s' = s_{\mathtt{L}}$ or $s' = s_{\mathtt{R}}$.
- If $s = \langle \bar{P}\rangle(s_{\mathtt{L}}, s_{\mathtt{R}})$ then $\exists$ chooses to set $s' = s_{\mathtt{L}}$ or $s' = s_{\mathtt{R}}$.
- If $s = [j](s')$ with $j > \ell$ then s' is defined and that round of $\mathcal{C}_P$ has priority $j{-}2{+}P$.
- Otherwise $\forall$ immediately wins.

The rounds of $\mathcal{C}_P$ which priority is not declared above have priority k. An infinite play Π of $\mathcal{C}_P$ is won by $\exists$ if the least priority seen infinitely often during Π is even.

▶ **Lemma 15.** *The game $\mathcal{C}_P(p,s)$ can be unravelled as a tree $c_P(p,s)$ over the alphabet $A^{\sim}_{(i,k)}$ in such a way that for well-formed trees p, s, the tree $c_P(p,s)$ is well-formed and $\exists$ wins $\mathcal{C}_P(p,s)$ if and only if $\mathtt{1}$ wins $\mathcal{G}\big(c_P(p,s)\big)$. Moreover, the function c_P is continuous.*

Proof. Notice that in both cases when a round of $\mathcal{C}_P$ has an explicitly declared priority, that priority is j or $j-1$ with $i \leq \ell < j \leq k$. Therefore, the priorities of $\mathcal{C}_P$ are within $\{i,\ldots,k\}$.

The condition $s \notin \mathrm{W}^{\sim}_{\mathbf{1},(i,k)}$ from (EL) boils down to checking if $\mathtt{1}$ wins $\mathcal{G}\big(\sim(s)\big)$. Similarly, $s \notin \mathrm{W}^{\sim}_{\mathbf{2},(i,k)}$ if and only if $\mathtt{1}$ wins $\mathcal{G}\big(s\big)$. The same for the condition $p \in \mathrm{W}^{\sim}_{P,(i,k)}$ from (AL). Continuity and well-formedness follow directly from the definition. ◀

Notice that the rules of the game $\mathcal{C}_P$ do not allow to move from a position with a tree (either p or s) of the form $\sim(t)$ to a position with the respective tree being t. Therefore, we never need to swap the considered player P into $\bar{P}$.

Claim 14 together with Lemma 15 prove Lemma 12. Thus, the rest of this section is devoted to a proof of Claim 14. Since the winning condition of $\mathcal{C}_P$ is a parity condition, that game is positionally determined. Thus, to prove Claim 14 it is enough to show that none of the following two cases is possible for a position (p,s,ℓ) of $\mathcal{C}_P$:

- (2) is true and $\forall$ has a positional winning strategy $\Sigma_\forall$ from (p,s,ℓ),
- (2) is false and $\exists$ has a positional winning strategy $\Sigma_\exists$ from (p,s,ℓ).

In both cases we will confront the assumed strategy with a specially designed positional quasi-strategy of the opponent ($\Sigma^\star_\exists$ and $\Sigma^\star_\forall$ respectively). The quasi-strategy $\Sigma^\star_\exists$ will be defined only in positions that satisfy (2) and the quasi-strategy $\Sigma^\star_\forall$ in the remaining positions.

The quasi-strategy $\Sigma^\star_\exists$ (resp. $\Sigma^\star_\forall$) of a player $\exists$ (resp. $\forall$) in a round starting in a position $\big(p,s,\ell\big)$ performs the following choices in sub-rounds (EL) to (AI):

- In (EL) $\Sigma^\star_\exists$ claims that P loses $\mathcal{G}(s)$ if and only if he really does.
- In (AL) $\Sigma^\star_\forall$ claims that P loses $\mathcal{G}(p)$ if and only if he really does.
- In (EI) $\Sigma^\star_\exists$ modifies ℓ into ℓ' if $\ell' < \ell$ is the minimal P-losing number such that $\sigma_P(p)\restriction\ell' <_{\mathrm{lex}} \sigma_P(s)\restriction\ell'$. If there is no such number, $\Sigma^\star_\exists$ does not declare ℓ'.
- In (AI) $\Sigma^\star_\forall$ modifies ℓ into ℓ' if $\ell' < \ell$ is the minimal P-losing number such that $\sigma_P(p)\restriction\ell' >_{\mathrm{lex}} \sigma_P(s)\restriction\ell'$. If there is no such number, $\Sigma^\star_\forall$ does not declare ℓ'.

Moreover, in $\mathit{Step}\exists(p)$ when $p = \langle P\rangle(p_{\mathtt{L}}, p_{\mathtt{R}})$ the quasi-strategy $\Sigma^\star_\exists$ chooses to set $p' = p_{\mathtt{L}}$ if and only if $\sigma_P(p_{\mathtt{L}}) \leq_{\mathrm{lex}} \sigma_P(p_{\mathtt{R}})$. Dually, in $\mathit{Step}\forall(s)$ when $s = \langle P\rangle(s_{\mathtt{L}}, s_{\mathtt{R}})$ the quasi-strategy $\Sigma^\star_\forall$ chooses to set $s' = s_{\mathtt{L}}$ if and only if $\sigma_P(s_{\mathtt{L}}) \leq_{\mathrm{lex}} \sigma_P(s_{\mathtt{R}})$.

Now it remains to define the choices of the quasi-strategies in the steps $\mathit{Step}\forall(s)$ and $\mathit{Step}\exists(p)$ when the given tree is of the form $\langle\bar{P}\rangle(t_{\mathtt{L}}, t_{\mathtt{R}})$. This is the place where the choices of $\Sigma^\star_\exists$ and $\Sigma^\star_\forall$ are not unique and that is why these are quasi-strategies.

▶ **Definition 16.** The quasi-strategies $\Sigma^\star_\exists$ and $\Sigma^\star_\forall$ need to satisfy the following *preservation guarantees*. First, in $\mathit{Step}\forall(s)$ when $s = \langle\bar{P}\rangle(s_{\mathtt{L}}, s_{\mathtt{R}})$ then $\Sigma^\star_\exists$ can set s' as any of the two $s_{\mathtt{L}}$, $s_{\mathtt{R}}$ that satisfies $\sigma_P(s')\restriction\ell \geq_{\mathrm{lex}} \sigma_P(p)\restriction\ell$. Second, in $\mathit{Step}\exists(p)$ when $p = \langle\bar{P}\rangle(p_{\mathtt{L}}, p_{\mathtt{R}})$ then $\Sigma^\star_\forall$ can set p' as any of the two $p_{\mathtt{L}}$, $p_{\mathtt{R}}$ that satisfies $\sigma_P(p')\restriction\ell >_{\mathrm{lex}} \sigma_P(s)\restriction\ell$.

▶ **Fact 17.** *In both cases the preservation guarantee leaves at least one possible choice.*

▶ **Lemma 18.** *Consider a position $\big(p,s,\ell\big)$. If it satisfies* (2) *and $\exists$ follows her quasi-strategy $\Sigma^\star_\exists$ then either she immediately wins or the next position also satisfies* (2)*.*

Dually, if the position violates (2) *and $\forall$ follows his quasi-strategy $\Sigma^\star_\forall$ then either he immediately wins or the next position also violates* (2)*.*

Notice that each play of $\mathcal{C}_P$ can modify the value of ℓ only bounded number of times. Moreover, because of the conditions in steps ($\downarrow\!\!\downarrow$), ($\downarrow p$), and ($\downarrow s$) we obtain the following fact.

▶ **Fact 19.** *If* Π *is an infinite play of* $\mathcal{C}_P$ *then exactly one of the following three cases holds:*

- Π *makes infinitely many* $(\twoheaddownarrow)$ *steps,*
- *from some point on* Π *makes only* $(\downarrow p)$ *steps,*
- *from some point on* Π *makes only* $(\downarrow s)$ *steps.*

Observe that each step of $\mathcal{C}_P$ of the form $Step\exists(p)$ or $Step\forall(s)$ (if it doesn't mean an immediate win) simulates in fact a round of the game $\mathcal{G}(p)$ and $\mathcal{G}(s)$ respectively. Moreover, the quasi-strategies of the players $\exists$ and $\forall$ simulate optimal strategies of P in these rounds respectively. Thus, P must win these plays, as expressed by the following lemma.

▶ **Lemma 20.** *Consider an infinite play consistent with the quasi-strategy of one of the players* ($\Sigma^\star_\exists$ *or* $\Sigma^\star_\forall$*). Then the play only finitely many times makes the* $(\twoheaddownarrow)$ *step.*

Moreover, if the play follows $\Sigma^\star_\exists$ *and from some point on makes only* $(\downarrow p)$ *steps then it is winning for* $\exists$*. Similarly, if the play follows* $\Sigma^\star_\forall$ *and from some point on makes only* $(\downarrow s)$ *steps then it is winning for* $\forall$*.*

Proof. First take an infinite play Π of the quasi-strategy $\Sigma^\star_\exists$ of $\exists$ starting from a position (p_0, s_0, ℓ_0). The subtrees of the tree p_0 seen during Π follow a (possibly finite) play α of an optimal strategy of P in $\mathcal{G}(p_0)$. Since Π is infinite, $\exists$ does not declare that P loses $\mathcal{G}(s_0)$. Therefore, $\sigma_P(s_0) <_{\text{lex}} \infty$ and by (2) we know that also $\sigma_P(p_0) <_{\text{lex}} \infty$ which implies that $p_0 \in \mathrm{W}^{\sim}_{P,(i,k)}$. By Lemma 11 the play α in $\mathcal{G}(p_0)$ follows a winning strategy of P.

We will show that the step $(\twoheaddownarrow)$ occurs only finitely many times in Π. Assume contrarily and let ℓ be the minimal number ℓ that occurs in the play Π. Using the above assumptions, we know that ℓ is the minimal priority that is seen in the tree p_0 infinitely many times on α. Therefore, the simulated play α in $\mathcal{G}(p_0)$ is infinite and losing for P, which is a contradiction.

Therefore, by Fact 19 the play Π either makes from some point on only $(\downarrow p)$ steps, or from some point on only $(\downarrow s)$ steps. In the first case it follows the play α of $\mathcal{G}(p_0)$ that is winning for P. By the choice of priorities in the step $Step\exists(p)$ we know that $\exists$ wins Π.

The case of the quasi-strategy $\Sigma^\star_\forall$ of $\forall$ is entirely dual: we use the assumption that (2) is violated to know that $\sigma_P(s) <_{\text{lex}} \infty$ so $s_0 \in \mathrm{W}^{\sim}_{P,(i,k)}$. Moreover, the choice of priorities in the step $Step\forall(s)$ implies that if Π makes $Step\forall(s)$ infinitely many times then $\forall$ wins Π. ◀

Now we move to the proof of the first case we need to exclude, i.e. that a position (p, s, ℓ) of $\mathcal{C}_P$ satisfies (2) but $\forall$ has a positional winning strategy $\Sigma_\forall$ from that position. We will prove that such a case is not possible. The second case is dual and the proof is analogous.

By Lemma 18, the positional quasi-strategy $\Sigma^\star_\exists$ always stays within positions satisfying the invariant (2). Moreover, the quasi-strategy never reaches a position that is immediately losing for $\exists$. Similarly, $\Sigma_\forall$ never reaches a position that is immediately losing for $\forall$. Thus, all the plays consistent with both $\Sigma^\star_\exists$ and $\Sigma_\forall$ must be infinite.

Notice that the values of ℓ are non-increasing during the plays of $\mathcal{C}_P$ and therefore, there exists a position that belongs to both $\Sigma^\star_\exists$ and $\Sigma_\forall$ such that the value ℓ stays constant during all the plays from that position. Without loss of generality take this as the starting position.

We can now proceed inductively in the tree obtained by unravelling the intersection of $\Sigma^\star_\exists$ and $\Sigma_\forall$: whenever the currently considered subtree contains anywhere a $(\twoheaddownarrow)$ step, we change the initial position to the result of that step. By Lemma 20 no play consistent with $\Sigma^\star_\exists$ takes the $(\twoheaddownarrow)$ step infinitely many times. Therefore, our inductive procedure has to stop at some point with no $(\twoheaddownarrow)$ steps in the current subtree. Without loss of generality we can assume that the initial position (p_0, s_0, ℓ_0) is the last position from the procedure. We know that the plays consistent with both $\Sigma^\star_\exists$ and $\Sigma_\forall$ never take the $(\twoheaddownarrow)$ step nor modify $\ell = \ell_0$.

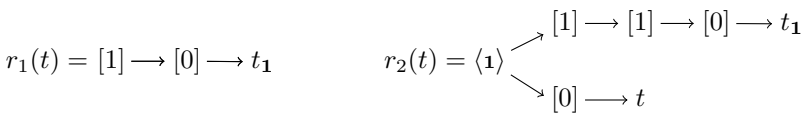


Figure 3 The pair of trees being the result of the reduction $r(t)$ from Proposition 22.

The structure of $\mathcal{C}_P$ guarantees that since the step $(\downarrow\!\!\!\downarrow)$ is not allowed, each play consistent with both $\Sigma^{\star}_{\exists}$ and $\Sigma_{\forall}$ takes only $(\downarrow p)$ steps or takes only $(\downarrow s)$ steps. Lemma 20 implies that in the former case the play would be winning for $\exists$, contradicting the assumption that $\Sigma_{\forall}$ is winning. Thus, all the considered plays take only $(\downarrow s)$ steps. In particular $p = p_0$ is constant.

The intersection of $\Sigma_{\forall}$ and $\Sigma^{\star}_{\exists}$ induces a partial strategy Σ_P of P in $\mathcal{G}(s_0)$—Σ_P is partial because it does not contain positions that cannot be reached by following $\Sigma^{\star}_{\exists}$ because of the preservation guarantees, see Definition 16. The subtrees s' of s_0 in such unreachable positions satisfy $\sigma_P(s')\restriction_{\ell_0} <_{\text{lex}} \sigma_P(p_0)\restriction_{\ell_0}$ by the definition of $\Sigma^{\star}_{\exists}$. In the positions on which Σ_P is defined it never visits a priority j with $j \leq \ell_0$ nor a node labelled $\sim$ because such a move is immediately losing for $\forall$ in $\mathit{Step}\forall(s)$. Because of the choice of the priorities in $\mathit{Step}\forall(s)$ and since $\Sigma_{\forall}$ is winning, Σ_P is winning for P on infinite plays.

Notice that since we take only $(\downarrow s)$ steps, p_0 must be of the form $[\ell_0](p'_0)$. Therefore, $\sigma_P(p_0)\restriction_{\ell_0} = (\theta_{i'}, \dots, \theta_{\ell_0} + 1)$ for $(\theta_{i'}, \dots, \theta_{\ell_0}) \stackrel{\text{def}}{=} \sigma_P(p'_0)\restriction_{\ell_0}$. It means that whenever the partial strategy Σ_P cannot reach a position with a subtree s', we know that in fact $\sigma_P(s')\restriction_{\ell_0} \leq_{\text{lex}} (\theta_{i'}, \dots, \theta_{\ell_0})$. The following lemma says that the existence of such a partial strategy Σ_P witnesses the inequality $\sigma_P(s_0)\restriction_{\ell_0} \leq_{\text{lex}} (\theta_{i'}, \dots, \theta_{\ell_0})$. By the definition of the ordinals θ_j we know that $(\theta_{i'}, \dots, \theta_{\ell_0}) <_{\text{lex}} \sigma_P(p_0)\restriction_{\ell_0}$, what contradicts (2) for (p_0, s_0, ℓ_0).

▶ **Lemma 21.** *Let $P \in \{\mathbf{1}, \mathbf{2}\}$, $t \in \mathrm{W}^{\sim}_{P,(i,k)}$, i' be the minimal P-losing number, and ℓ be some P-losing number. Assume that $(\theta_{i'}, \theta_{i'+2}, \dots, \theta_\ell)$ is a tuple of ordinals and Σ_P is a partial strategy of the player P in $\mathcal{G}(t)$ such that:*

- *Σ_P never reaches a node u with $t(u) = [j]$ with $j \leq \ell$ nor a node u with $t(u) = \sim$,*
- *infinite plays of Σ_P are winning for P,*
- *if a position $u \in \mathrm{dom}(t)$ is not reachable by Σ then $\sigma_P\big(t\restriction_u\big)\restriction_\ell \leq_{\text{lex}} (\theta_{i'}, \dots, \theta_\ell)$.*

Under all these assumptions $\sigma_P(t)\restriction_\ell \leq_{\text{lex}} (\theta_{i'}, \dots, \theta_\ell)$.

This finishes the proof of Claim 14. We conclude this section with a simple argument showing that a similar reduction c does not exist when we disallow the swapping symbol $\sim$, as expressed by the following proposition.

▶ **Proposition 22.** *There is no continuous function $c' \colon \big(\mathrm{Tr}_{A_{(0,k)}}\big)^2 \to \mathrm{Tr}_{A_{(0,k)}}$ such that*

$$c'(t_{\mathtt{L}}, t_{\mathtt{R}}) \in \mathrm{W}_{\mathbf{1},(0,k)} \quad \textit{if and only if} \quad \sigma_{\mathbf{1}}(t_{\mathtt{L}}) \leq_{\text{lex}} \sigma_{\mathbf{1}}(t_{\mathtt{R}}).$$

Proof. Assume that such a function c' exists. Fix a tree $t_{\mathbf{1}} \in \mathrm{W}_{\mathbf{1},(0,k)}$ and consider a function $r \colon \mathrm{Tr}_{A_{(0,k)}} \to \big(\mathrm{Tr}_{A_{(0,k)}}\big)^2$ defined as $r(t) = \big(r_1(t), r_2(t)\big)$ for the pair of trees from Figure 3. Clearly r is continuous. Let $t \in \mathrm{Tr}_{A_{(0,k)}}$ and $r(t) = (t_{\mathtt{L}}, t_{\mathtt{R}})$. Notice that $\sigma_{\mathbf{1}}(t_{\mathtt{L}}) = (1, 0, \dots)$. The value $\sigma_{\mathbf{1}}(t_{\mathtt{R}})$ is either $(0, 0, \dots)$ if $t \in \mathrm{W}_{\mathbf{1},(0,k)}$ or $(2, 0, \dots)$ otherwise. Therefore, $c'\big(r(t)\big) \in \mathrm{W}_{\mathbf{1},(0,k)}$ iff $\sigma_{\mathbf{1}}(t_{\mathtt{L}}) \leq_{\text{lex}} \sigma_{\mathbf{1}}(t_{\mathtt{R}})$ iff $t \notin \mathrm{W}_{\mathbf{1},(0,k)}$ iff $t \in \mathrm{W}_{\mathbf{2},(0,k)}$. Thus, $c' \circ r \colon \mathrm{Tr}_{A_{(0,k)}} \to \mathrm{Tr}_{A_{(0,k)}}$ is a continuous reduction of $\mathrm{W}_{\mathbf{2},(0,k)}$ to $\mathrm{W}_{\mathbf{1},(0,k)}$. This is a contradiction with [2, Lemma 1] (the assumption of contractivity is redundant there by Lemma 2 from the same paper). ◀

6 Conclusions

The main result of this work is the construction of the languages $L_{P,(i,k)}$ that solve the question of index bounds for unambiguous languages. Although the construction is not direct and relies heavily on an involved theory of signatures, these complications seem to be unavoidable when one wants to recognise languages like $W_{i,k}$ in an unambiguous way.

The definition of signatures given in the paper seems to be the canonical one, as witnessed by the point-wise minimality from Lemma 10. The previous ways of using signatures were mainly focused on their monotonicity and well-foundedness, thus it was enough to assume inequalities in the invariants of Lemma 10. Here, we are interested in comparing their actual values, therefore we insist on preserving these values via equalities.

References

1 André Arnold. The mu-calculus alternation-depth hierarchy is strict on binary trees. *ITA*, 33(4/5):329–340, 1999.
2 André Arnold and Damian Niwiński. Continuous separation of game languages. *Fundamenta Informaticae*, 81(1-3):19–28, 2007.
3 Marcin Bilkowski and Michał Skrzypczak. Unambiguity and uniformization problems on infinite trees. In *CSL*, pages 81–100, 2013.
4 Julian C. Bradfield. Simplifying the modal mu-calculus alternation hierarchy. In *STACS*, pages 39–49, 1998.
5 Arnaud Carayol and Christof Löding. MSO on the infinite binary tree: Choice and order. In *CSL*, pages 161–176, 2007.
6 Arnaud Carayol, Christof Löding, Damian Niwiński, and Igor Walukiewicz. Choice functions and well-orderings over the infinite binary tree. *Cent. Europ. J. of Math.*, 8:662–682, 2010.
7 Thomas Colcombet. Forms of determinism for automata. In *STACS*, pages 1–23, 2012.
8 Jacques Duparc, Kevin Fournier, and Szczepan Hummel. On unambiguous regular tree languages of index (0, 2). In *CSL*, pages 534–548, 2015.
9 Allen Emerson and Charanjit Jutla. Tree automata, mu-calculus and determinacy. In *FOCS'91*, pages 368–377, 1991.
10 Yuri Gurevich and Saharon Shelah. Rabin's uniformization problem. *J. Symb. Log.*, 48(4):1105–1119, 1983.
11 Szczepan Hummel. Unambiguous tree languages are topologically harder than deterministic ones. In *GandALF*, pages 247–260, 2012.
12 Szczepan Hummel. *Topological Complexity of Sets Defined by Automata and Formulas*. PhD thesis, University of Warsaw, 2017.
13 Alexander S. Kechris. *Classical descriptive set theory*. Springer-Verlag, New York, 1995.
14 Andrzej W. Mostowski. Games with forbidden positions. Technical report, University of Gdańsk, 1991.
15 Damian Niwiński and Igor Walukiewicz. Ambiguity problem for automata on infinite trees. unpublished, 1996.
16 Richard E. Stearns and Harry B. Hunt. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM Journal of Computing*, 14(3):598–611, 1985.
17 Robert S. Streett and E. Allen Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, 81(3):249–264, 1989.
18 Wolfgang Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, pages 389–455. Springer, 1996.

19 Wolfgang Thomas and Helmut Lescow. Logical specifications of infinite computations. In *REX School/Symposium*, pages 583–621, 1993.

20 Igor Walukiewicz. Pushdown processes: Games and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification*, pages 62–74, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

21 Igor Walukiewicz. Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 275(1–2):311–346, 2002.

The Beta-Bernoulli process and algebraic effects

Sam Staton
Univ. Oxford

Dario Stein
Univ. Oxford

Hongseok Yang
KAIST

Nathanael L. Ackerman
Harvard Univ.

Cameron E. Freer
Borelian

Daniel M. Roy
Univ. Toronto

Abstract

In this paper we use the framework of algebraic effects from programming language theory to analyze the Beta-Bernoulli process, a standard building block in Bayesian models. Our analysis reveals the importance of abstract data types, and two types of program equations, called commutativity and discardability. We develop an equational theory of terms that use the Beta-Bernoulli process, and show that the theory is complete with respect to the measure-theoretic semantics, and also in the syntactic sense of Post. Our analysis has a potential for being generalized to other stochastic processes relevant to Bayesian modelling, yielding new understanding of these processes from the perspective of programming.

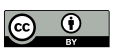
2012 ACM Subject Classification Theory of computation → Probabilistic computation

Keywords and phrases Beta-Bernoulli process, Algebraic effects, Probabilistic programming, Exchangeability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.141

Related Version A full version of the paper is available at [27], `https://arxiv.org/abs/1802.09598`.

Acknowledgements It has been helpful to discuss this work with many people, including Ohad Kammar, Gordon Plotkin, Alex Simpson, and Marcin Szymczak. This work is partly supported by EPSRC grants EP/N509711/1 and EP/N007387/1, a Royal Society University Research Fellowship, and an Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2015-0-00565, Development of Vulnerability Discovery Technologies for IoT Software Security).



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 141; pp. 141:1–141:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

From the perspective of programming, a family of Boolean random processes is implemented by a module that supports the following interface:

```
module type ProcessFactory = sig type process
                                 val new : H → process
                                 val get : process → bool end
```

where H is some type of hyperparameters. Thus one can initialize a new process, and then get a sequence of Booleans from that process. The type of processes is kept abstract so that any internal state or representation is hidden.

One can analyze a module extensionally in terms of the properties of its interactions with a client program. In this paper, we perform this analysis for the Beta-Bernoulli process, an important building block in Bayesian models. We completely axiomatize its equational properties, using the formal framework of algebraic effects [18].

The following modules are our leading examples. (Here flip (r) tosses a coin with bias r.)

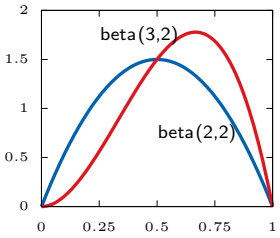
```
module Polya = (struct
  type process = (int * int) ref
  let new(i,j) = ref (i,j)
  let get p = let (i,j) = !p in
    if flip (i/(i+j)) then p := (i+1,j); true
    else p := (i,j+1); false  end : ProcessFactory)
```

```
module BetaBern = (struct
  type process = real
  let new(i,j) = sample_beta(i,j)
  let get(r) = flip(r)

end : ProcessFactory)
```

The left-hand module, Polya, is an implementation of Pólya's urn. An urn in this sense is a hidden state which contains i-many balls marked true and j-many balls marked false. To sample, we draw a ball from the urn at random; before we tell what we drew, we put back the ball we drew as well as an identical copy of it. The contents of the urn changes over time.

The right-hand module, BetaBern, is based on the beta distribution. This is the probability measure on the unit interval $[0,1]$ that measures the bias of a random source (such as a potentially unfair coin) from which true has been observed $(i-1)$ times and false has been observed $(j-1)$ times, as illustrated on the right. For instance beta$(2,2)$ describes the situation where we only know that neither true nor false are impossible; while in beta$(3,2)$ we are still ignorant but we believe that true is more likely.



It turns out that these two modules have the same observable behaviour. This essentially follows from de Finetti's theorem (e.g. [24]), but rephrased in programming terms. The equivalence makes essential use of type abstraction: if we could look into the urn, or ask precise questions about the real number, the modules would be distinguishable.

The module Polya has a straightforward operational semantics (although we don't formalize that here). By contrast, BetaBern has a straightforward denotational semantics [14]. In Section 2, we provide an axiomatization of equality, which is sound by both accounts. We show completeness of our axiomatization with respect to the denotational semantics of BetaBern (§3, Thm. 9). We use this to show that the axiomatization is in fact syntactically complete (§4, Cor. 13), which means it is complete with respect to *any* semantics.

For the remainder of this section, we give a general introduction to our axioms.

Commutativity and discardability. Commutativity and discardability are important program equations [5] that are closely related, we argue, to exchangeability in statistics.

- *Commutativity* is the requirement that when x is not free in u and y is not free in t,

$$\Big(\textsf{let x = t in let y = u in v}\Big) \quad = \quad \Big(\textsf{let y = u in let x = t in v}\Big).$$

- *Discardability* is the requirement that when x is not free in u, $\Big(\textsf{let x = t in u}\Big) = \Big(\textsf{u}\Big)$.

Together, these properties say that data flow, rather than the control flow, is what matters. For example, in a standard programming language, the purely functional total expressions are commutative and discardable. By contrast, expressions that write to memory are typically not commutative or discardable (a simple example is t=u=a++, v=(x,y)). A simple example of a commutative and discardable operation is a coin toss: we can reorder the outcomes of tossing a single coin, and we can drop some of the results (unconditionally) without changing the overall statistics.

We contend that commutativity and discardability of program expressions is very close to the basic notion of exchangeability of infinite sequences, which is central to Bayesian statistics. Informally, an infinite random process, such as an infinite random sequence, is said to be exchangeable if one can reorder and discard draws without changing the overall statistics. (For more details on exchangeable random processes in probabilistic programming languages, see [1, 28], and the references therein.) A client program for the BetaBern module is clearly exchangeable in this sense: this is roughly Fubini's theorem. For the Polya module, an elementary calculation is needed: it is not trivial because memory is involved.

Conjugacy. Besides exchangeability, the following conjugacy equation is crucial:

$$\Big(\textsf{let p=M.new(i,j) in (M.get(p), p)}\Big)$$
$$= \Big(\textsf{if flip (i/(i+j)) then (true, M.new(i+1,j)) else (false, M.new(i,j+1))}\Big).$$

This is essentially the operational semantics of the Polya module, and from the perspective of BetaBern it is the well-known conjugate-prior relationship between the Beta and Bernoulli distributions.

Finite probability. In addition to exchangeability and conjugacy, we include the standard equations of finite, discrete, rational probability theory. To introduce these, suppose that we have a module

```
Bernoulli : sig val get : int * int → bool end
```

which is built so that Bernoulli.get(i,j) samples *with single replacement* from an urn with i-many balls marked true and j-many balls marked false. (In contrast to Pólya's urn, the urn in this simple scheme does not change over time.) So Bernoulli.get(i,j) = flip $(\frac{i}{i+j})$. This satisfies certain laws, first noticed long ago by Stone [29], and recalled in §2.1.

In summary, our main contribution is that these axioms — exchangeability, conjugacy, and finite probability — entirely determine the equational theory of the Beta-Bernoulli process, in the following sense:

- *Model completeness:* Every equation that holds in the measure theoretic interpretation is derivable from our axioms (Thm. 9);
- *Syntactical completeness:* Every equation that is not derivable from our axioms is inconsistent with finite discrete probability (Cor. 13).

We argue that these results open up a new method for analyzing Bayesian models, based on algebraic effects (see §5 and [28][1]).

2 An algebraic presentation of the Beta-Bernoulli process

In this section, we present syntactic rules for well-formed client programs of the Beta-Bernoulli module, and axioms for deriving equations on those programs.

2.1 An algebraic presentation of finite probability

Recall the module Bernoulli from the introduction which provides a method of sampling with odds $(i : j)$. We will axiomatize its equational properties. Algebraic effects provide a way to axiomatize the specific features of this module while putting aside the general properties of programming languages, such as β/η laws. In this situation the basic idea is that each module induces a binary operation ${}_i?_j$ on programs by

$$t \,{}_i?_j\, u \stackrel{\text{def}}{=} \textbf{if}\ \ \mathsf{Bernoulli}.\mathsf{get}(i,j)\ \textbf{then}\ t\ \textbf{else}\ u.$$

Conversely, given a family of binary operations ${}_i?_j$, we can recover $\mathsf{Bernoulli}.\mathsf{get}(i,j) = \mathsf{true}\,{}_i?_j\,\mathsf{false}$. So to give an equational presentation of the Bernoulli module we give a equational presentation of the binary operations ${}_i?_j$. A full programming language will have other constructs and $\beta\eta$-laws but it is routine to combine these with an algebraic theory of effects (e.g. [2, 8, 9, 21]).

▶ **Definition 1.** The *theory of rational convexity* is the first-order algebraic theory with binary operations ${}_i?_j$ for all $i, j \in \mathbb{N}$ such that $i + j > 0$, subject to the axiom schemes

$$w, x, y, z \vdash (w \,{}_i?_j\, x) \,{}_{i+j}?_{k+l}\, (y \,{}_k?_l\, z) = (w \,{}_i?_k\, y) \,{}_{i+k}?_{j+l}\, (x \,{}_j?_l\, z)$$

$$x, y \vdash x \,{}_i?_j\, y = y \,{}_j?_i\, x \qquad x, y \vdash x \,{}_i?_0\, y = x \qquad x \vdash x \,{}_i?_j\, x = x$$

Commutativity $(w \,{}_i?_j\, x) \,{}_k?_l\, (y \,{}_i?_j\, z) = (w \,{}_k?_l\, y) \,{}_i?_j\, (x \,{}_k?_l\, z)$ of operations ${}_k?_l$ and ${}_i?_j$ is a derivable equation, and so is scaling $x \,{}_{ki}?_{kj}\, y = x \,{}_i?_j\, y$ for $k > 0$. Commutativity and discardability $(x \,{}_i?_j\, x = x)$ in this algebraic sense (cf. [15, 22]) precisely correspond to the program equations in Section 1 (see also [9]). The theory first appeared in [29].

2.2 A parameterized algebraic signature for Beta-Bernoulli

In the theory of convex sets, the parameters i, j for get range over the integers. These integers are not a first class concept in our equational presentation: we did not axiomatize integer arithmetic. However, in the Beta-Bernoulli process, or any module M for the ProcessFactory interface, it is helpful to understand the parameters to get as abstract, and new as generating such parameters. To interpret this, we treat these parameters to get as first class. There are still hyperparameters to new, which we do not treat as first class here. (In a more complex hierarchical system with hyperpriors, we might treat them as first class.)

As before, to avoid studying an entire programming language, we look at the constructions

$$\nu_{i,j} p.t \stackrel{\text{def}}{=} \textbf{let}\ p{=}\mathsf{M.new}(i,j)\ \textbf{in}\ t \qquad t \,?_p\, u \stackrel{\text{def}}{=} \textbf{if}\ \ \mathsf{M.get}(p)\ \textbf{then}\ t\ \textbf{else}\ u$$

[1] This paper formalizes and proves a conjecture from [28], which is an unpublished abstract.

There is nothing lost by doing this, because we can recover $\mathsf{M.new}(i,j) = \nu_{i,j}p.\,p$ and $\mathsf{M.get}(p) = \mathsf{true}\,?_p\,\mathsf{false}$. In the terminology of [18], these would be called the 'generic effects' of the algebraic operations $\nu_{i,j}$ and $?_p$. Note that $?_p$ is a parameterized binary operation. Formally, our syntax now has two kinds of variables: x, y as before, ranging over continuations, and now also p, q ranging over parameters. We notate this by having contexts with two zones, and write $x : n$ if x expects n parameters.

▶ **Definition 2.** The term formation rules for the theory of Beta-Bernoulli are:

$$\frac{-}{\Gamma \mid \Delta, x : m, \Delta' \vdash x(p_1 \dots p_m)}\ (p_1 \dots p_m \in \Gamma) \qquad \frac{\Gamma, p \mid \Delta \vdash t}{\Gamma \mid \Delta \vdash \nu_{i,j}p.t}\ (i,j > 0)$$

$$\frac{\Gamma \mid \Delta \vdash t \quad \Gamma \mid \Delta \vdash u}{\Gamma \mid \Delta \vdash t\,?_p\,u}\ (p \in \Gamma) \qquad \frac{\Gamma \mid \Delta \vdash t \quad \Gamma \mid \Delta \vdash u}{\Gamma \mid \Delta \vdash t\,{}_i?_j\,u}\ (i + j > 0)$$

where Γ is a parameter context of the form $\Gamma = (p_1, \dots, p_\ell)$ and Δ is a context of the form $\Delta = (x_1 \colon m_1, \dots, x_k \colon m_k)$. Where $x \colon 0$, we often write x for $x()$. For the sake of a well-defined notion of dimension in 3.2.4, we disallow the formation of $\nu_{i,0}$ and $\nu_{0,i}$.

We work up-to α-conversion and substitution of terms for variables must avoid unintended capture of free parameters. For example, substituting $x\,?_p\,y$ for w in $\nu_{1,1}p.w$ yields $\nu_{1,1}q.(x\,?_p\,y)$, while substituting $x\,?_p\,y$ for $z(p)$ in $\nu_{1,1}p.z(p)$ yields $\nu_{1,1}p.(x\,?_p\,y)$.

2.3 Axioms for Beta-Bernoulli

The axioms for the Beta-Bernoulli theory comprise the axioms for rational convexity (Def. 1) together with the following axiom schemes.

Commutativity. All the operations commute with each other:

$$p, q \mid w, x, y, z : 0 \vdash (w\,?_q\,x)\,?_p(y\,?_q\,z) = (w\,?_p\,y)\,?_q(x\,?_p\,z) \tag{C1}$$

$$- \mid x \colon 2 \vdash \nu_{i,j}p.(\nu_{k,l}q.x(p,q)) = \nu_{k,l}q.(\nu_{i,j}p.x(p,q)) \tag{C2}$$

$$q \mid x, y \colon 1 \vdash \nu_{i,j}p.(x(p)\,?_q\,y(p)) = (\nu_{i,j}p.x(p))\,?_q(\nu_{i,j}p.y(p)) \tag{C3}$$

$$- \mid x, y : 1 \vdash \nu_{i,j}p.(x(p)\,{}_k?_l\,y(p)) = (\nu_{i,j}p.x(p))\,{}_k?_l(\nu_{i,j}p.y(p)) \tag{C4}$$

$$p \mid w, x, y, z : 0 \vdash (w\,{}_i?_j\,x)\,?_p(y\,{}_i?_j\,z) = (w\,?_p\,y)\,{}_i?_j(x\,?_p\,z) \tag{C5}$$

Discardability. All operations are idempotent:

$$- \mid x \colon 0 \vdash (\nu_{i,j}p.x) = x \qquad\qquad p \mid x \colon 0 \vdash x\,?_p\,x = x \tag{D1–2}$$

Conjugacy.

$$- \mid x, y : 1 \vdash \nu_{i,j}p.(x(p)\,?_p\,y(p)) = (\nu_{i+1,j}p.x(p))\,{}_i?_j(\nu_{i,j+1}p.y(p)) \tag{Conj}$$

A theory of equality for terms in context is built, as usual, by closing the axioms under substitution, congruence, reflexivity, symmetry and transitivity. It immediately follows from conjugacy and discardability that $x\,{}_i?_j\,y$ is definable as $\nu_{i,j}p.(x\,?_p\,y)$ for $i, j > 0$.

As an example, consider $t(r) = (r\,?_p\,x)\,?_p(y\,?_p\,r)$ that represents tossing a coin with bias p twice, continuing with x or y if the results are different, or with r otherwise. One can show that $x\,{}_1?_1\,y$ is a unique fixed point of t, i.e. $x\,{}_1?_1\,y = t(x\,{}_1?_1\,y)$; see the full paper [27] for detail. This is exactly von Neumann's trick [31] to simulate a fair coin toss with a biased one.

(For more details on the general axiomatic framework with parameters, see [25, 26], where it is applied to predicate logic, π-calculus, and other effects.)

3 A complete interpretation in measure theory

In this section we give an interpretation of terms using measures and integration operators, the standard formalism for probability theory (e.g. [19, 24]), and we show that this interpretation is complete (Thm. 9). Even if the reader is not interested in measure theory, they may still find value in the syntactical results of §4 which we prove using this completeness result.

By the Riesz–Markov–Kakutani representation theorem, there are two equivalent ways to view probabilistic programs: as probability kernels and as linear functionals. Both are useful.

Programs as probability kernels.

Forgetting about abstract types for a moment, terms in the BetaBern module are first-order probabilistic programs. So we have a standard denotational semantics due to [14] where terms are interpreted as probability kernels and ν as integration. Let $I = [0,1]$ denote the unit interval. We write $\beta_{i,j}$ for the Beta(i,j)-distribution on I, which is given by the density function $p \mapsto \frac{1}{B(i,j)} p^{i-1}(1-p)^{j-1}$, where $B(i,j) = \frac{(i-1)!(j-1)!}{(i+j-1)!}$ is a normalizing constant.

For contexts of the form $\Gamma = (p_1, \dots, p_\ell)$ and $\Delta = (x_1 : m_1, \dots, x_k : m_k)$, we let $[\![\Delta]\!] \stackrel{\text{def}}{=} \sum_{i=1}^k I^{m_i}$ consist of a copy of I^{m_i} for every variable $x_i : m_i$. This has a σ-algebra $\Sigma([\![\Delta]\!])$ generated by the Borel sets. We interpret terms $\Gamma \mid \Delta \vdash t$ as probability kernels $[\![t]\!] : I^\ell \times \Sigma([\![\Delta]\!]) \to [0,1]$ inductively, for $\vec{p} \in I^\ell$ and $U \in \Sigma([\![\Delta]\!])$:

$$[\![x_i(p_{j_1}, \dots, p_{j_m})]\!](\vec{p}, U) = 1 \text{ if } (i, p_{j_1} \dots p_{j_m}) \in U, \text{ 0 otherwise}$$

$$[\![u \mathbin{?_j} v]\!](\vec{p}, U) = \tfrac{1}{i+j}\Big(i([\![u]\!](\vec{p}, U)) + j([\![v]\!](\vec{p}, U))\Big)$$

$$[\![u \mathbin{?_{p_j}} v]\!](\vec{p}, U) = p_j([\![u]\!](\vec{p}, U)) + (1-p_j)([\![v]\!](\vec{p}, U))$$

$$[\![\nu_{i,j} q.t]\!](\vec{p}, U) = \int_0^1 [\![t]\!]((\vec{p}, q), U)\, \beta_{i,j}(\mathrm{d}q) \quad \Big[= \int_0^1 [\![t]\!]((\vec{p}, q), U)\ \tfrac{1}{B(i,j)} q^{i-1}(1-q)^{j-1}\ \mathrm{d}q \Big]$$

▶ **Proposition 3.** *The interpretation is sound: if $\Gamma \mid \Delta \vdash t = u$ is derivable then $[\![t]\!] = [\![u]\!]$ as probability kernels $[\![\Gamma]\!] \times \Sigma([\![\Delta]\!]) \to [0,1]$.*

Proof notes. One must check that the axioms are sound under the interpretation. Each of the axioms are elementary facts about probability. For instance, commutativity (C2) amounts to Fubini's theorem, and the conjugacy axiom (Conj) is the well-known conjugate-prior relationship of Beta- and Bernoulli distributions. ◀

Interpretation as functionals

We write $\mathbb{R}^{I^m}$ for the vector space of continuous functions $I^m \to \mathbb{R}$, endowed with the supremum norm. Given a probability kernel $\kappa : I^\ell \times \Sigma\big(\sum_{j=1}^k I^{m_j}\big) \to [0,1]$ and $\vec{p} \in I^\ell$, we define a linear map $\phi_{\vec{p}} : \mathbb{R}^{I^{m_1}} \times \cdots \times \mathbb{R}^{I^{m_k}} \to \mathbb{R}$, by considering κ as an integration operator:

$$\phi_{\vec{p}}(f_1 \dots f_k) = \int f_j(r_1 \dots r_{m_j})\ \kappa(\vec{p}, \mathrm{d}(j, r_1 \dots r_{m_j}))$$

Here $\phi_{\vec{p}}$ are unital ($\phi(\vec{1}) = 1$) and positive ($\vec{f} \geq 0 \implies \phi(\vec{f}) \geq 0$).

When $\kappa = [\![t]\!]$, this $\phi_{\vec{p}}(\vec{f})$ is moreover continuous in $\vec{p}$, and hence a unital positive linear map $\phi : \mathbb{R}^{I^{m_1}} \times \cdots \times \mathbb{R}^{I^{m_k}} \to \mathbb{R}^{I^\ell}$ [6, Thm. 5.1]. It is informative to spell out the interpretation of terms $p_1, \dots, p_\ell \mid x_1 : m_1, \dots, x_k : m_k \vdash t$ as maps $[\![t]\!] : \mathbb{R}^{I^{m_1}} \times \dots \times \mathbb{R}^{I^{m_k}} \to \mathbb{R}^{I^\ell}$ since it fits the algebraic notation: we may think of the variables $x\colon m$ as ranging over functions $\mathbb{R}^{I^m}$.

▶ **Proposition 4.** *The functional interpretation is inductively given by*

$$[\![x_i(p_{j_1},\dots,p_{j_m})]\!](\vec{f})(\vec{p}) = f_i(p_{j_1},\dots,p_{j_m})$$
$$[\![u \,{}_i?_j\, v]\!](\vec{f})(\vec{p}) = \tfrac{1}{i+j}\Big(i([\![u]\!](\vec{f})(\vec{p})) + j([\![v]\!](\vec{f})(\vec{p}))\Big)$$
$$[\![u \,?_{p_j}\, v]\!](\vec{f})(\vec{p}) = p_j([\![u]\!](\vec{f})(\vec{p})) + (1-p_j)([\![v]\!](\vec{f})(\vec{p}))$$
$$[\![\nu_{i,j}q.t]\!](\vec{f})(\vec{p}) = \int_0^1 [\![t]\!](\vec{f})(\vec{p},q)\,\beta_{i,j}(\mathrm{d}q)$$

For example, $[\![- \mid x,y\colon 0 \vdash x \,{}_1?_1\, y]\!] : \mathbb{R}\times\mathbb{R}\to\mathbb{R}$ is the function $(x,y)\mapsto \frac{1}{2}(x+y)$, and $[\![- \mid x\colon 1 \vdash \nu_{1,1}p.x(p)]\!] : \mathbb{R}^I \to \mathbb{R}$ is the integration functional, $f \mapsto \int_0^1 f(p)\,\mathrm{d}p$.

(We use the same brackets $[\![-]\!]$ for both the measure-theoretic and the functional interpretations; the intended semantics will be clear from context.)

3.1 Technical background on Bernstein polynomials

▶ **Definition 5** (Bernstein polynomials)**.** For $i = 0,\dots,k$, we define the i-th basis Bernstein polynomial $b_{i,k}$ of degree k as $b_{i,k}(p) = \binom{k}{i}p^{k-i}(1-p)^i$. For a multi-index $I = (i_1,\dots,i_\ell)$ with $0 \le i_j \le k$, we let $b_{I,k}(\vec{p}) = b_{i_1,k}(p_1)\cdots b_{i_\ell,k}(p_\ell)$. A Bernstein polynomial is a linear combination of Bernstein basis polynomials.

The family $\{b_{i,k} : i = 0,\dots,k\}$ is indeed a basis of the polynomials of maximum degree k and also a partition of unity, i.e. $\sum_{i=0}^k b_{i,k} = 1$. Every Bernstein basis polynomial of degree k can be expressed as a nonnegative rational linear combination of degree $k+1$ basis polynomials.

The density function of the distribution $\beta_{i,j}$ on $[0,1]$ for $i,j > 0$ is proportional to a Bernstein basis polynomial of degree $i+j-2$. We can conclude that the measures $\{\beta_{i,j} : i,j > 0, i+j = n\}$ are linearly independent for every n. In higher dimensions, the polynomials $\{b_{I,k}\}$ are linearly independent for every k. Moreover, products of beta distributions β_{i_r,j_r} are linearly independent as long as $i_r + j_r = n$ holds for some common n. This will be a key idea for normalizing Beta-Bernoulli terms.

3.2 Normal forms and completeness

For the completeness proof of the measure-theoretic model, we proceed as follows: To decide $\Gamma \mid \Delta \vdash t = u$ for two terms t,u, we transform them into a common normal form whose interpretations can be given explicitly. We then use a series of linear independence results to show that if the interpretations agree, the normal forms are already syntactically equal.
Normalization happens in three stages.

- If we think of a term as a syntax tree of binary choices and ν-binders, we use the conjugacy axiom to push all occurrences of ν towards the leaves of the tree.
- We use commutativity and discardability to stratify the use of free parameters $?_p$.
- The leaves of the tree will now consist of chains of ν-binders, variables and ratio choices ${}_i?_j$. Those can be collected into a canonical form.

We will describe these normalization stages in reverse order because of their increasing complexity.

3.2.1 Stone's normal forms for rational convex sets

Normal forms for the theory of rational convex sets have been described by Stone [29]. We note that if $- \mid x_1 \dots x_k : 0 \vdash t$ is a term in the theory of rational convex sets (Def. 1) then

$[\![t]\!] : \mathbb{R}^k \to \mathbb{R}$ is a unital positive linear map that takes rationals to rationals. From the perspective of measures, this corresponds to a categorical distribution with k categories.

▶ **Proposition 6** (Stone). *The interpretation exhibits a bijective correspondence between terms $- \mid x_1 \dots x_k : 0 \vdash t$ built from $?_i$, modulo equations, and unital positive linear maps $\mathbb{R}^k \to \mathbb{R}$ that take rationals to rationals.*

For instance, the map $\phi(x, y, z) = \frac{1}{10}(2x + 3y + 5z)$ is unital positive linear, and arises from the term $t \stackrel{\text{def}}{=} x \,?_8 (y \,?_5 z)$. This is the only term that gives rise to the ϕ, modulo equations. In brief, one can recover t from ϕ by looking at $\phi(1, 0, 0) = \frac{2}{10}$, then $\phi(0, 1, 0) = \frac{3}{10}$, then $\phi(0, 0, 1) = \frac{5}{10}$. We will write $\begin{pmatrix} ? & x_1 & \dots & x_k \\ & w_1 & \dots & w_k \end{pmatrix}$ for the term corresponding to the linear map $(x_1 \dots x_k) \mapsto \frac{1}{\sum_{i=1}^k w_k}(w_1 x_1 + \dots + w_k x_k)$. These are normal forms for the theory of rational convex sets.

3.2.2 Characterization and completeness for ν-free terms

This section concerns the normalization of terms using free parameters but no ν. Consider a single parameter p. If we think of a term t as a syntactic tree, commutativity and discardability can be used to move all occurrences of $?_p$ to the root of the tree, making it a *tree diagram* of some depth k. Let us label the 2^k leaves with $t_{a_1 \cdots a_k}$, $a_i \in \{0, 1\}$. As a programming language expression, this corresponds to successive bindings

let a_1=M.get(p) **in** ... **let** a_k=M.get(p) **in** $t_{a_1 \cdots a_k}$

Permutations $\sigma \in S_k$ of the k first levels in the tree act on tree diagrams by permuting the leaves via $t_{a_1 \cdots a_k} \mapsto t_{a_{\sigma(1)} \cdots a_{\sigma(k)}}$. By commutativity (C1), those permuted diagrams are still equal to t, so we can replace t by the average over all permuted diagrams, since rational choice is discardable. The average commutes down to the leaves (C5), so we obtain a tree diagram with leaves $m_{a_1 \cdots a_k} = \frac{1}{k!} \sum_\sigma t_{a_{\sigma(1)} \cdots a_{\sigma(k)}}$, where the average is to be read as a rational choice with all weights 1. This new tree diagram is now by construction invariant under permutation of levels in the tree, in particular $m_{a_1 \cdots a_k}$ only depends on the sum $a_1 + \dots + a_k$. That is to say, the counts are a sufficient statistic.

This leads to the following normalization procedure for terms $p_1 \dots p_\ell \mid x_1 \dots x_n : 0 \vdash t$: Write $C_k^{p_j}(t_0, \dots, t_k)$ for the permutation invariant tree diagram of p_j-choices and depth k with leaves $t_{a_1 \cdots a_k} = t_{a_1 + \cdots + a_k}$. Then we can rewrite t as $C_k^{p_1}(t_0, \dots, t_k)$ where each t_i is p_1-free. Recursively normalize each t_i in the same way, collecting the next parameter. By discardability, we can pick the height of all these tree diagrams to be a single constant k, such that the resulting term is a nested structure of tree-diagrams $C_k^{p_j}$. We will use multi-indices $I = (i_1, \dots, i_\ell)$ to write the whole stratified term as $C_k((t_I))$ where each leaf t_I only contains rational choices. The interpretation of such a term can be given explicitly by Bernstein polynomials

$$[\![C_k((t_I))]\!](\vec{x})(\vec{p}) = \textstyle\sum_I b_{I,k}(\vec{p}) \cdot [\![t_I]\!](\vec{x})(\vec{p}).$$

For example, normalizing $(v \,?_p x) ?_p (y \,?_p v)$ gives $(v \,?_p (x \,?_1 y)) ?_p ((x \,?_1 y) \,?_p v) = C_2(v, x ?_1 y, v)$.

From this we obtain the following completeness result:

▶ **Proposition 7.** *There is a bijective correspondence between equivalence classes of terms $p_1 \dots p_\ell \mid x_1 \dots x_n : 0 \vdash t$ and linear unital maps $\phi : \mathbb{R}^n \to \mathbb{R}^{I^\ell}$ such that for every standard basis vector e_j of $\mathbb{R}^n$, $\phi(e_j)$ is a Bernstein polynomial with nonnegative rational coefficients.*

Proof. We can assume all basis polynomials to have the same degree k. If $\phi(e_j) = \sum_I w_{Ij} b_{I,k}$, then the unitality condition $\phi(1,\dots,1) = 1$ means $\sum_I \left(\sum_j w_{Ij}\right) b_{I,k} = 1$, and hence by linear independence and partition of unity, $\sum_j w_{Ij} = 1$ for every I. If we thus let t_I be the rational convex combination of the x_j with weights w_{Ij}, then $[\![C_k((t_I))]\!] = \phi$. Again by linear independence, the weights w_{IJ} are uniquely defined by ϕ. ◀

Geometric characterizations for the assumption of this theorem exist in [20, 3]. For example, a univariate polynomial is a Bernstein polynomial with nonnegative coefficients if and only if it is positive on $(0,1)$. More care is required in the multivariate case.

3.2.3 Normalization of Beta-Bernoulli

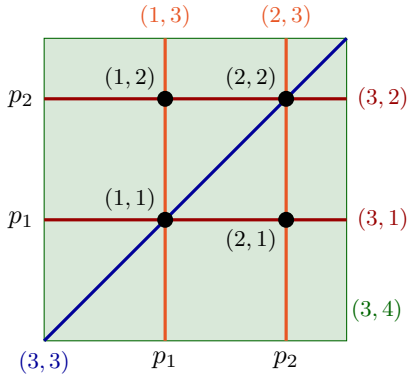
For arbitrary terms $p_1 \dots p_\ell \,|\, x_1 \colon m_1, \dots, x_s \colon m_s \vdash t$, we employ the following normalization procedure. Using conjugacy and the commutativity axioms (C2–C4), we can push all uses of ν towards the leaves of the tree, until we end up with a tree of ratios and free parameter choices only. Next, by conjugacy and discardability, we expand every instance of $\nu_{i,j}$ until they satisfy $i + j = n$ for some fixed, sufficiently large n. We then stratify the free parameters into permutation invariant tree diagrams. That is, we find a number k such that t can be written as $C_k((t_I))$ where the leaves t_I consist of ν and rational choices only.

In each t_I, commuting all the choices up to the root, we are left with a convex combination of chains of ν's of the form $\nu_{i_1,j_1} p_{\ell+1} \dots \nu_{i_d,j_d} p_{\ell+d}.x_j(p_{\tau(1)},\dots,p_{\tau(m)})$ for some $\tau : m \to \ell+d$. By discardability, we can assume that there are no unused bound parameters. We consider two chains equal if they are α-convertible into each other. Now if $c_1,\dots,c_m$ is a list of the distinct chains that occur in any of the leaves, we can give the leaves t_I the uniform shape $t_I = \begin{pmatrix} ? & c_1 & \dots & c_m \\ & w_{I1} & \dots & w_{Im} \end{pmatrix}$ for appropriate weights $w_{Ij} \in \mathbb{N}$. We will show that this representation is a unique normal form.

3.2.4 Proof of completeness

Consider a *chain* $c = \nu_{i_1,j_1} p_{\ell+1} \dots \nu_{i_d,j_d} p_{\ell+d}.\, x(p_{\tau(1)},\dots,p_{\tau(m)})$. Its measure-theoretic interpretation $[\![c]\!](p_1,\dots,p_\ell)$ is a pushforward of a product of d beta distributions, supported on a hyperplane segment that is parameterized by the map $h_\tau : I^d \to I^m, h_\tau(p_{\ell+1},\dots,p_{\ell+d}) = (p_{\tau(1)},\dots,p_{\tau(m)})$. Note that the position of the hyperplane may vary with the free parameters. To capture this geometric information, we call τ the *subspace type* of the chain and d its *dimension*. Because of α-invariance of chains, we identify subspace types that differ by a permutation of $\{\ell+1,\dots,\ell+d\}$.

For example, each chain with two free parameters p_1, p_2 and a variable $x : 2$ gives rise to a parameterized distribution on the unit square. On the right, we illustrate the ten possible supports that such distributions can have, as subspaces of the square. In the graphic we write (i,j) for $\nu p_3.\nu p_4.x(p_i,p_j)$, momentarily omitting the subscripts of ν because they do not affect the support. For instance, the upper horizontal line corresponds to $\nu p_3.x(p_3,p_2)$; the bottom-right dot corresponds to $x(p_2,p_1)$; the diagonal corresponds to $\nu p_3.x(p_3,p_3)$; and the entire square corresponds to



$\nu p_3.\nu p_4.x(p_3,p_4)$. All told there are four subspaces of dimension $d=0$, five with $d=1$, and one with $d=2$. Notice that the subspaces are all distinct as long as $p_1 \neq p_2$.

▶ **Proposition 8.** *If $c_1,\dots,c_s$ are distinct chains with $i_1+j_1=\dots=i_d+j_d=n$, then the family of functionals $\{[\![c_i]\!](-)(\vec{p}) : \mathbb{R}^{I^{m_1}} \times \dots \times \mathbb{R}^{I^{m_s}} \to \mathbb{R}\}_{i=1,\dots,s}$ is linearly independent whenever all parameters p_i are distinct.*

Proof. Fix $\vec{p}$. Chains on different variables are clearly independent, so we can restrict ourselves to a single variable $x:m$. We reason measure-theoretically. The interpretation of a chain c_i of subspace type τ_i is a pushforward measure $h_{i*}(\mu_i)$ where μ_i is a product of d beta distributions, and h_i is the affine inclusion map $h_i(p_{\ell+1},\dots,p_{\ell+d}) = (p_{\tau_i(1)},\dots,p_{\tau_i(m)})$. Let $\sum a_i h_{i*}(\mu_i) = 0$ as a signed measure. We show by induction over the dimension of the chains that all a_i vanish. Assume that $a_i=0$ whenever the dimension of c_i is less than d, and consider an arbitrary subspace τ_j of dimension d. We can define a signed Borel measure on I^d by restriction

$$\rho(A) \stackrel{\text{def}}{=} \sum_i a_i h_{i*}(\mu_i)(h_j(A)) = \sum_i a_i \mu_i(h_i^{-1}(h_j(A)))$$

as h_j sends Borel sets to Borel sets (e.g. [10, §15A]). We claim that $\rho(A) = \sum_{c_i \text{ has type } \tau_j} a_i\mu_i(A)$, as the contributions of chains c_i of different type vanish.

- If c_i has dimension $< d$, $a_i=0$ by the inductive hypothesis.
- If c_i has dimension $> d$, we note that $h_i^{-1}(h_j(A))$ only has at most dimension d. It is therefore a nullset for μ_i.
- If c_i has dimension d but a different type, and all $p_1,\dots,p_\ell$ are assumed distinct, then the hyperplanes given by h_i and h_j are not identical. Therefore their intersection is at most $(d-1)$-dimensional and $h_i^{-1}(h_j(A))$ is a nullset for μ_i.

By assumption, ρ has to be the zero measure, but the μ_i are linearly independent. Therefore $a_i=0$ for all c_i with subspace type τ_j. Repeat this for every subspace type of dimension d to conclude overall linear independence. ◀

▶ **Theorem 9** (Completeness). *If $\Gamma \mid \Delta \vdash t,t'$ and $[\![t]\!]=[\![t']\!]$, then $\Gamma \mid \Delta \vdash t=t'$.*

Proof. From the normalization procedure, we find numbers k,n, a list of distinct chains $c_1,\dots,c_s$ with $i+j=n$ and weights $(w_{Ij}),(w'_{Ij})$ such that $\Gamma \mid \Delta \vdash t = C_k((t_I))$ and $\Gamma \mid \Delta \vdash t' = C_k((t'_I))$ where $t_I = \begin{pmatrix} ? & c_1 & \dots & c_s \\ & w_{I1} & \dots & w_{Is} \end{pmatrix}$ and $t'_I = \begin{pmatrix} ? & c_1 & \dots & c_s \\ & w'_{I1} & \dots & w'_{Is} \end{pmatrix}$. The interpretations of these normal forms are given explicitly by

$$[\![t]\!](\vec{f})(\vec{p}) = \sum_j \frac{w_{Ij}}{w_I} \cdot b_{I,k}(\vec{p}) \cdot [\![c_j]\!](\vec{f})(\vec{p}) \text{ where } w_I = \sum_j w_{Ij}$$

and analogously for t'. Then $[\![t]\!]=[\![t']\!]$ implies that for all $\vec{f}$

$$\sum_j \left(\sum_I \left(\frac{w_{Ij}}{w_I} - \frac{w'_{Ij}}{w'_I} \right) b_{I,k}(\vec{p}) \right) [\![c_j]\!](\vec{f})(\vec{p}) = 0.$$

By Proposition 8, this implies $\sum_I \left(\frac{w_{Ij}}{w_I} - \frac{w'_{Ij}}{w'_I} \right) b_{I,k}(\vec{p}) = 0$ for every j and whenever the parameters p_i are distinct. By continuity of the left hand side, the expression in fact has to vanish for *all* $\vec{p}$. By linear independence of the Bernstein polynomials, we obtain $w_{Ij}/w_I = w'_{Ij}/w'_I$ for all I,j. Thus, all weights agree up to rescaling and we can conclude $\Gamma \mid \Delta \vdash t=t'$. ◀

4 Extensionality and syntactical completeness

In this section we use the model completeness of the previous section to establish some syntactical results about the theory of Beta-Bernoulli. Although the model is helpful in informing the proofs, the statements of the results in this section are purely syntactical.

The ultimate result of this section is equational syntactical completeness (Cor. 13), which says that there can be no further equations in the theory without it becoming inconsistent with discrete probability. In other words, assuming that the axioms we have included are appropriate, they must be sufficient, regardless of any discussion about semantic models or intended meaning. This kind of result is sometimes called 'Post completeness' after Post proved a similar result for propositional logic.

The key steps towards this result are two extensionality results. These are related to the programming language idea of 'contextual equivalence'. Recall that in a programming language we often define a basic notion of equivalence on closed ground terms: these are programs with no free variables that return (say) booleans. This notion is often defined by some operational consideration using some notions of observation. From this we define contextual equivalence by saying that $t \approx u$ if, for all closed ground contexts $\mathcal{C}$, $\mathcal{C}[t] = \mathcal{C}[u]$.

Contextual equivalence has a canonical appearance, but an axiomatic theory of equality, such as the one in this paper, is more compositional and easier to work with. Our notion of equality induces in particular a basic notion of equivalence on closed ground terms. Our extensionality results say that, assuming one is content with this basic notion of equivalence, the equations that we axiomatize coincide with contextual equivalence.

4.1 Extensionality

▶ **Proposition 10** (Extensionality for closed terms). *Suppose* $\Gamma, q \,|\, \Delta \vdash t$ *and* $\Gamma, q \,|\, \Delta \vdash u$. *If* $\Gamma \,|\, \Delta \vdash \nu_{i,j} q.t = \nu_{i,j} q.u$ *for all* i, j, *then also* $\Gamma \,|\, \Delta \vdash t = u$.

Proof. We show the contrapositive. By the model completeness theorem (Thm. 9), we can reason in the model rather than syntactically. So we consider t and u such that $[\![t]\!] \neq [\![u]\!]$ as functions $\mathbb{R}^{I^{m_1}} \times \mathbb{R}^{I^{m_k}} \to \mathbb{R}^{I^{l+1}}$, and show that there are i, j such that $[\![\nu_{i,j} q.t]\!] \neq [\![\nu_{i,j} q.u]\!]$. By assumption there are $\vec{f}$ and $\vec{p}, q$ such that $[\![t]\!](\vec{f})(\vec{p}, q) \neq [\![u]\!](\vec{f})(\vec{p}, q)$ as real numbers.

Now we use the following general reasoning: For any real $q \in I$ we can pick monotone sequences $i_1 < \cdots < i_n < \dots$ and $j_1 < \cdots < j_n < \dots$ of natural numbers so that $\frac{i_n}{i_n + j_n} \to q$ as $n \to \infty$. Moreover, for any continuous $h : I \to \mathbb{R}$, the integral $\int h \, \mathrm{d}\beta_{i_n, j_n}$ converges to $h(q)$ as $n \to \infty$: one way to see this is to notice that the variance of β_{i_n, j_n} vanishes as $n \to \infty$, so by Chebyshev's inequality, $\lim_n \beta_{i_n, j_n}$ is a Dirac distribution at q. Thus, $\int \left([\![t]\!](\vec{f})(\vec{p}, r) - [\![u]\!](\vec{f})(\vec{p}, r) \right) \beta_{i_n, j_n}(\mathrm{d}r)$ is non-zero as $n \to \infty$. By continuity, for some n, $\int [\![t]\!](\vec{f})(\vec{p}, r) \; \beta_{i_n, j_n}(\mathrm{d}r) \neq \int [\![u]\!](\vec{f})(\vec{p}, r) \; \beta_{i_n, j_n}(\mathrm{d}r)$. So, $[\![\nu_{i_n, j_n} q.t]\!] \neq [\![\nu_{i_n, j_n} q.u]\!]$. ◀

▶ **Proposition 11** (Extensionality for ground terms). In brief: *If* $t[{}^{v_1 \dots v_k}/_{x_1 \dots x_k}] = u[{}^{v_1 \dots v_k}/_{x_1 \dots x_k}]$ *for all suitable ground* $v_1 \dots v_k$, *then* $t = u$.

In detail: *Consider* t *and* u *with* $- \,|\, x_1 \colon m_1 \dots x_k \colon m_k \vdash t, u$. *Suppose that whenever* $v_1 \dots v_k$ *are terms with* $(p_1 \dots p_{m_1} \,|\, y, z : 0 \vdash v_1)$, $\dots$, $(p_1 \dots p_{m_k} \,|\, y, z : 0 \vdash v_k)$, *then we have* $- \,|\, y, z : 0 \vdash t[{}^{v_1 \dots v_k}/_{x_1 \dots x_k}] = u[{}^{v_1 \dots v_k}/_{x_1 \dots x_k}]$. *Then we also have* $- \,|\, x_1 \colon m_1 \dots x_k \colon m_k \vdash t = u$.

Proof. Again, we show the contrapositive. Let $\Delta = (x_1 \colon m_1 \dots x_k \colon m_k)$. Suppose we have t and u such that $\neg(- \,|\, \Delta \vdash t = u)$. Then by the model completeness theorem (Thm. 9), we have $[\![t]\!] \neq [\![u]\!]$ as linear functions $\mathbb{R}^{I^{m_1}} \times \cdots \times \mathbb{R}^{I^{m_k}} \to \mathbb{R}$. Since the functions are linear, there is an index $i \leq k$ and a continuous function $f : I^{m_i} \to \mathbb{R}$ with $[\![t]\!](0 \dots 0, f, 0 \dots 0) \neq [\![u]\!](0 \dots 0, f, 0 \dots 0)$. By the Stone-Weierstrass theorem, every such f

is a limit of polynomials, and so since $[\![t]\!]$ and $[\![u]\!]$ are continuous and linear, there has to be a Bernstein basis polynomial $b_{I,k} : \mathbb{R}^{m_i} \to \mathbb{R}$ that already distinguishes them. This function is definable, i.e. there is a a term $p_1, \ldots, p_{m_i} \mid y, z : 0 \vdash w$ with $[\![w]\!](1,0) = b_{I,k}$. Define terms $v_j = w$ for $i = j$ and $v_j = z$ for $i \neq j$. Then

$$[\![t[{}^{v_1\cdots v_k}/_{x_1\ldots x_k}]]\!](1,0) = [\![t]\!](0,\ldots,b_{I,k},\ldots,0) \neq [\![u]\!](0,\ldots,b_{I,k},\ldots,0) = [\![u[{}^{v_1\cdots v_k}/_{x_1\ldots x_k}]]\!](1,0).$$

The required $\neg\big(- \mid y, z : 0 \vdash t[{}^{v_1\cdots v_k}/_{x_1\ldots x_k}] = u[{}^{v_1\cdots v_k}/_{x_1\ldots x_k}]\big)$ follows from the above disequality because of the model soundness property (Props. 3 and 4). ◀

From the programming perspective, a term $- \mid y, z : 0 \vdash t_0$ corresponds to a closed program of type bool, for it has two possible continuations, y and z, depending on whether the outcome is true or false. From this perspective, Proposition 11 says that for closed t, u, if $\mathcal{C}[t] = \mathcal{C}[u]$ for all boolean contexts $\mathcal{C}$, then $t = u$.

4.2 Relative syntactical completeness

▶ **Proposition 12** (Neumann, [17]). *If t, u are terms in the theory of rational convexity (Def. 1), then either $t = u$ is derivable or it implies $x\,{}_i?_j\, y = x\,{}_{i'}?_{j'}\, y$ for all nonzero i, i', j, j'.*

▶ **Corollary 13.** *The theory of Beta-Bernoulli is syntactically complete relative to the theory of rational convexity, in the following sense. For all terms t and u, either $t = u$ is derivable, or it implies $x\,{}_i?_j\, y = x\,{}_{i'}?_{j'}\, y$ for all nonzero i, i', j, j'.*

This is proved by combining Propositions 10, 11 and 12. As an example for extensionality and completeness, consider the equation $\nu_{1,1}p.x(p,p) = \nu_{1,1}p.(\nu_{1,1}q.x(p,q))$. It is not derivable, as can be witnessed by the substitution $x(p,q) = (y\,?_q\,z)\,?_p\,z$. Normalizing yields $y\,{}_1?_2\,z = y\,{}_1?_3\,z$ which is incompatible with discrete probability (see the full paper [27]). In programming syntax, the candidate equation is written

LHS = **let** p = M.new(1,1) **in** (p,p)	RHS = (M.new(1,1) , M.new(1,1))

and the distinguishing context is C[−] = **let** (p,q)=(−) **in if** M.get(p) **then** M.get(q) **else** false. That is to say, the closed ground programs C[LHS] and C[RHS] necessarily have different observable statistics: this follows from the axioms.

4.3 Remark about stateful implementations

In the introduction we recalled the idea of using Pólya's urn to implement a Beta-Bernoulli process using local (hidden) state.

Our equational presentation gives a recipe for understanding the correctness of the stateful implementation. First, one would give an operational semantics, and then a basic notion of observational equivalence on closed ground terms in terms of the finite probabilities associated with reaching certain ground values. From this, an operational notion of contextual equivalence can be defined (e.g. [4, §6], [23, 32]). Then, one would show that the axioms of our theory hold up-to contextual equivalence. Finally one can deduce from the syntactical completeness result that the equations satisfied by this stateful implementation must be exactly the equations satisfied by the semantic model.

In fact, in this argument, it is not necessary to check that axioms (C1) and (D2) hold in the operationally defined contextual equivalence, because the axiomatized equality on closed ground terms is independent of these axioms. To see this, notice that our normalization procedure (§3.2.3) doesn't use (C1) or (D2) when the terms are closed and ground, since

then we can take $n = k = 0$. This is helpful because the remaining axioms are fairly straightforward, e.g. (Conj) is the essence of the urn scheme and (D1) is garbage collection.

5 Conclusion

Exchangeable random processes are central to many Bayesian models. The general message of this paper is that the analysis of exchangeable random processes, based on basic concepts from programming language theory, depends on three crucial ingredients: commutativity, discardability, and abstract types. We have illustrated this message by showing that just adding the conjugacy law to these ingredients leads to a complete equational theory for the Beta-Bernoulli process (Thm. 9). Moreover, we have shown that this equational theory has a canonical syntactic and axiomatic status, regardless of the measure theoretic foundation (Cor. 13). Our results in this paper open up the following avenues of research.

Study of nonparametric Bayesian models: We contend that abstract types, commutativity and discardability are fundamental tools for studying nonparametric Bayesian models, especially hierarchical ones. For example, the Chinese Restaurant Franchise [30] can be implemented as a module with three abstract types, f (franchise), r (restaurant), t (table), and functions $\mathsf{newFranchise}\colon() \to \mathsf{f}$, $\mathsf{newRestaurant}\colon\mathsf{f} \to \mathsf{r}$, $\mathsf{getTable}\colon\mathsf{r} \to \mathsf{t}$, $\mathsf{sameDish}\colon\mathsf{t}*\mathsf{t} \to \mathsf{bool}$. Its various exchangeability properties correspond to commutativity/discardability in the presence of type abstraction. (For other examples, see [28].)

First steps in synthetic probability theory: As is well known, the theory of rational convex sets corresponds to the monad D of rational discrete probability distributions. Commutativity of the theory amounts to commutativity of the monad D [15, 12].

As any parameterized algebraic theory, the theory of Beta-Bernoulli (§2) can be understood as a monad P on the functor category $[\mathbf{FinSet}, \mathbf{Set}]$, with the property that to give a natural transformation $\mathbf{FinSet}(\ell, -) \to P(\coprod_{j=1}^{k} \mathbf{FinSet}(m_k, -))$ is to give a term $(p_1 \dots p_\ell \,|\, x_1 \colon m_1 \dots x_k \colon m_k \vdash t)$, and monadic bind is substitution ([25, Cor. 1], [26, §VIIA]). This can be thought of as an intuitionistic set theory with an interesting notion of probability. As such this is a 'commutative effectus' [7], a synthetic probability theory (see also [13]). Like D, the global elements $1 \to P(2)$ are the rationals in $[0, 1]$ (by Prop. 7) but unlike D, the global elements $1 \to P(P(2))$ include the beta distribution.

Practical ideas for nonparametric Bayesian models in probabilistic programming: A more practical motivation for our work is to inform the design of module systems for probabilistic programming languages. For example, Anglican, Church, Hansei and Venture already support nonparametric Bayesian primitives [11, 33, 16]. We contend that abstract types are a crucial concept from the perspective of exchangeability.

References

1 Nathanael L. Ackerman, Cameron E. Freer, and Daniel M. Roy. Exchangeable random primitives. Workshop on Probabilistic Programming Semantics (PPS 2016), 2016. URL: `http://pps2016.soic.indiana.edu/files/2015/12/xrp.pdf`.

2 Danel Ahman and Sam Staton. Normalization by evaluation and algebraic effects. In *Proc. MFPS 2013*, volume 298 of *Electron. Notes Theor. Comput. Sci*, pages 51–69, 2013.

3 M. Alves Diniz, L. E. Salasar, and R. Bassi Stern. Positive polynomials on closed boxes. *arXiv e-print 1610.01437*, 2016.

4 Ales Bizjak and Lars Birkedal. Step-indexed logical relations for probability. In *Proc. FOSSACS 2015*, pages 279–294, 2015.

5 Carsten Führmann. Varieties of effects. In *Proc. FOSSACS 2002*, pages 144–159, 2002.

6 Robert Furber and Bart Jacobs. From Kleisli categories to commutative C*-algebras: Probabilistic Gelfand duality. *Log. Methods Comput. Sci.*, 11(2), 2015. doi:10.2168/LMCS-11(2:5)2015.

7 Bart Jacobs. From probability monads to commutative effectuses. *J. Log. Algebr. Methods Program.*, 94:200–237, 2018.

8 Patricia Johann, Alex Simpson, and Janis Voigtländer. A generic operational metatheory for algebraic effects. In *Proc. LICS 2010*, pages 209–218, 2010.

9 Ohad Kammar and Gordon D. Plotkin. Algebraic foundations for effect-dependent optimisations. In *Proc. POPL 2012*, pages 349–360, 2012.

10 Alexander Kechris. *Classical Descriptive Set Theory*. Springer, 1995.

11 Oleg Kiselyov and Chung-Chieh Shan. Probabilistic programming using first-class stores and first-class continuations. In *Proc. 2010 ACM SIGPLAN Workshop on ML*, 2010.

12 Anders Kock. Monads on symmetric monoidal closed categories. *Arch. Math.*, 21:1–10, 1970.

13 Anders Kock. Commutative monads as a theory of distributions. *Theory Appl. Categ.*, 26(4):97–131, 2012.

14 Dexter Kozen. Semantics of probabilistic programs. *J. Comput. System Sci.*, 22(3):328–350, 1981.

15 F. E. J. Linton. Autonomous equational categories. *J. Math. Mech.*, 15:637–642, 1966.

16 Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv e-print 1404.0099*, 2014.

17 Walter D. Neumann. On the quasivariety of convex subsets of affine space. *Arch. Math.*, 21:11–16, 1970.

18 Gordon Plotkin and John Power. Algebraic operations and generic effects. *Appl. Categ. Structures*, 11(1):69–94, 2003.

19 David Pollard. *A user's guide to measure theoretic probability*. Cambridge University Press, 2001.

20 Victoria Powers and Bruce Reznick. Polynomials that are positive on an interval. *Trans. Amer. Math. Soc.*, 352(10):4677–4692, 2000.

21 Matija Pretnar. *The Logic and Handling of Algebraic Effects*. PhD thesis, Univ. Edinburgh, 2010.

22 Anna B. Romanowska and Jonathan D. H. Smith. *Modes*. World Scientific, 2002.

23 Davide Sangiorgi and Valeria Vignudelli. Environmental bisimulations for probabilistic higher-order languages. In *Proc. POPL 2016*, 2016.

24 Mark J. Schervish. *Theory of statistics*. Springer, 1995.

25 Sam Staton. An algebraic presentation of predicate logic. In *Proc. FOSSACS 2013*, pages 401–417, 2013.

26 Sam Staton. Instances of computational effects: An algebraic perspective. In *Proc. LICS 2013*, 2013.

27 Sam Staton, Dario Stein, Hongseok Yang, Nathanael L. Ackerman, Cameron E. Freer, and Daniel M. Roy. The Beta-Bernoulli process and algebraic effects. *arXiv e-print 1802.09598*, 2018.

28 Sam Staton, Hongseok Yang, Nathanael Ackerman, Cameron Freer, and Daniel M. Roy. Exchangeable random processes and data abstraction. Workshop on Probabilistic Programming Semantics (PPS 2017), 2017. URL: https://pps2017.soic.indiana.edu/files/2017/01/staton-yang-ackerman-freer-roy.pdf.

29 M. H. Stone. Postulates for the barycentric calculus. *Ann. Mat. Pura Appl. (4)*, 29:25–30, 1949.

30 Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. Hierarchical Dirichlet processes. *J. Amer. Statist. Assoc.*, 101(476):1566–1581, 2006.

31 John von Neumann. Various techniques used in connection with random digits. *Nat. Bur. Stand. Appl. Math. Series*, 12:36–38, 1951.

32 Mitchell Wand, Theophilos Giannakopoulos, Andrew Cobb, and Ryan Culpepper. Contextual equivalence for a probabilistic language with continuous random variables and recursion. Workshop on Probabilistic Programming Semantics (PPS 2018), 2018. URL: `https://pps2018.soic.indiana.edu/files/2018/01/paper.pdf`.

33 Jeff Wu. Reduced traces and JITing in Church. Master's thesis, Mass. Inst. of Tech., 2013.

Uniformization Problems for Synchronizations of Automatic Relations on Words

Sarah Winter
RWTH Aachen University, Germany
winter@cs.rwth-aachen.de

Abstract

A uniformization of a binary relation is a function that is contained in the relation and has the same domain as the relation. The synthesis problem asks for effective uniformization for classes of relations and functions that can be implemented in a specific way.

We consider the synthesis problem for automatic relations over finite words (also called regular or synchronized rational relations) by functions implemented by specific classes of sequential transducers.

It is known that the problem "Given an automatic relation, does it have a uniformization by a subsequential transducer?" is decidable in the two variants where the uniformization can either be implemented by an arbitrary subsequential transducer or it has to be implemented by a synchronous transducer. We introduce a new variant of this problem in which the allowed input/output behavior of the subsequential transducer is specified by a set of synchronizations and prove decidability for a specific class of synchronizations.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases automatic relation, uniformization, synchronization, transducer

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.142

Related Version A full version of the paper is available at `http://arxiv.org/abs/1805.02444`.

Funding Supported by the project (LO 1174/3-1) of the German Research Foundation (DFG).

Acknowledgements The author would like to thank her supervisor Christof Löding for suggesting this topic and his helpful comments and thank the anonymous reviewers of this and an earlier version of the paper for their feedback which greatly improved the presentation.

1 Introduction

A uniformization of a binary relation is a function that selects for each element in the domain of the relation a unique image that is in relation with this element. Of interest to us in this paper are uniformization problems in the setting where the relations and functions on words are defined by finite automata. Relations on words defined by finite automata extend languages defined by finite automata. Unlike for words, different finite automaton models for relations lead to different classes of relations.

Relations defined by asynchronous finite automata are referred to as rational relations. An asynchronous finite automaton is a nondeterministic finite automaton with two tapes whose reading heads can move at different speeds. An equivalent computation model are asynchronous finite transducers (see, e.g., [1]), that is, nondeterministic finite automata whose transitions are labeled by pairs of words.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 142; pp. 142:1–142:13


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A well known subclass of rational relations are synchronized rational relations (see [8]), which are defined by synchronous finite automata, that is, finite automata with two tapes whose reading heads move at the same speed. Equivalently, we speak of definability by synchronous finite transducers. The class of synchronized rational relations is also called automatic or regular, here, we use the term automatic.

One uniformization problem asks for proving that each relation in a given class has a certain kind of uniformization. For example, each rational relation can be uniformized by an unambiguous rational function (see [13]). Here, we are interested in the decision version of the problem: Given a relation from some class, does it have a uniformization in some other class? For the class of uniformizations we consider sequential transducers. A sequential transducer reads the input word in a deterministic manner and produces a unique output word for each input word.

The sequential uniformization problem relates to the synthesis problem, which asks, given a specification that relates possible inputs to allowed outputs, whether there is a program implementing the specification, and if so, construct one. This setting originates from Church's synthesis problem [4], where logical specifications over infinite words are considered. Büchi and Landweber [2] showed that for specifications in monadic second order logic, that is, specifications that can be translated into synchronous finite automata, it is decidable whether it can be realized by a synchronous sequential transducer (see, e.g., [14] for a modern presentation of this result). Later, decidability has been extended to asynchronous sequential transducers [10, 9].

Going from the setting of infinite words to finite words uniformization by subsequential [1] transducers is considered. The problem whether a relation given by a synchronous finite automaton can be realized by a synchronous subsequential transducer is decidable; this result can be obtained by adapting the proof from the infinite setting. Decidability has been extended to subsequential transducers [3]. Furthermore, for classes of asynchronous finite automata decidability results for synthesis of subsequential transducers have been obtained in [7].

A semi-algorithm in this spirit was introduced by [11], the algorithm is tasked to synthesize a subsequential transducer that selects the length lexicographical minimal output word for each input word from a given rational relation.

The decision problems that have been studied so far either ask for uniformization by a synchronous subsequential or by an arbitrary subsequential transducer. Our aim is to study the decision problem: Given a rational relation, does it have a uniformization by a subsequential transducer in which the allowed input/output behavior is specified by a given language of synchronizations? The idea is to represent a pair of words by a single word where each position is annotated over $\{1, 2\}$ indicating whether it came from the input or output component. The annotated string provides a synchronization of the pair. It is known that the class of rational relations is synchronized by regular languages [12]. More recently, main subclasses of rational relations have been characterized by their synchronizations [6].

We show decidability for a given automatic relation and a given set of synchronizations that synchronizes an automatic relation. Thus our decidability result generalizes the previously known decidability result for synthesis of synchronous subsequential transducers from automatic relations.

[1] A subsequential transducer can make a final output depending on the last state reached in a run whereas a sequential transducer can only produce output on its transitions.

The paper is structured as follows. First, in Sec. 2, we fix our notations and recap characterizations of synchronization languages established in [6]. In Sec. 3, we introduce uniformization problems with respect to synchronization languages and compare our setting with known results. In Sec. 4, we prove decidability of the question whether an automatic relation has a uniformization by a subsequential transducer in which the input/output behavior is specified by a set of synchronizations that synchronizes an automatic relation.

A full version of this paper can be found online.

2 Synchronizations of relations

Let $\mathbb{N}$ denote the set of all non-negative integers $\{0, 1, \dots\}$, and for every $k \in \mathbb{N}$, let $\mathbf{k}$ denote the set $\{1, \dots, k\}$. Given a finite set A, let $|A|$ denote its cardinality and 2^A its powerset.

Languages and relations of finite words. An *alphabet* Σ is a finite set of letters, a finite *word* is a finite sequence over Σ. The set of all finite words is denoted by Σ^* and the empty word by ε. The length of a word $w \in \Sigma^*$ is denoted by $|w|$, the number of occurrences of a letter $a \in \Sigma$ in w by $\#_a(w)$. Given $w \in \Sigma^*$, $w[i]$ stands for the ith letter of w, and $w[i, j]$ for the subword $w[i] \dots w[j]$.

A *language* L over Σ is a subset of Σ^*, and $\mathit{Pref}(L)$ is the set $\{u \in \Sigma^* \mid \exists v : uv \in L\}$ of its prefixes. The prefix relation is denoted by $\sqsubseteq$. A *relation* R over Σ is a subset of $\Sigma^* \times \Sigma^*$. The *domain* of a relation R is the set $\mathrm{dom}(R) = \{u \mid (u, v) \in R\}$, the *image* of a relation R is the set $\mathrm{img}(R) = \{v \mid (u, v) \in R\}$. For $u \in \Sigma^*$, let $R(u) = \{v \mid (u, v) \in R\}$ and write $R(u) = v$, if $R(u)$ is a singleton.

A *regular expression* r over Σ has the form $\emptyset$, ε, $\sigma \in \Sigma$, $r_1 \cdot r_2$, $r_1 + r_2$, or r_1^* for regular expressions r_1, r_2. The term r^+ is short for $r \cdot r^*$. The concatenation operator $\cdot$ is often omitted. The language associated to r is defined as usual, denoted $L(r)$, or conveniently, r.

▶ **Definition 1** (synchronization, L-controlled [6]). For $c \in \{\mathtt{i}, \mathtt{o}\}$, referring to input and output, respectively, we define two morphisms $\pi_c \colon (\mathbf{2} \times \Sigma) \to \Sigma \cup \{\varepsilon\}$ by $\pi_{\mathtt{i}}((i, a)) = a$ if $i = 1$, otherwise $\pi_{\mathtt{i}}((i, a)) = \varepsilon$, and likewise for $\pi_{\mathtt{o}}$ with $i = 2$. These morphisms are lifted to words over $(\mathbf{2} \times \Sigma)$.

A word $w \in (\mathbf{2} \times \Sigma)^*$ is a *synchronization* of a uniquely determined pair (w_1, w_2) of words over Σ, where $w_1 = \pi_{\mathtt{i}}(w)$ and $w_2 = \pi_{\mathtt{o}}(w)$. We write $[\![w]\!]$ to denote (w_1, w_2). Naturally, a set $S \subseteq (\mathbf{2} \times \Sigma)^*$ of synchronizations defines the relation $[\![S]\!] = \{[\![w]\!] \mid w \in S\}$.

A word $w = (i_1, a_1) \dots (i_n, a_n) \in (\mathbf{2} \times \Sigma)^*$ is the convolution $u \otimes v$ of two words $u = i_1 \dots i_n \in \mathbf{2}^*$ and $v = a_1 \dots a_n \in \Sigma^*$. Given a language $L \subseteq \mathbf{2}^*$, we say w is *L-controlled* if $u \in L$. A language $S \subseteq (\mathbf{2} \times \Sigma)^*$ is *L-controlled* if all its words are.

A language $L \subseteq \mathbf{2}^*$ is called a *synchronization language*. For a regular language $L \subseteq \mathbf{2}^*$, $\textsc{Rel}(L) = \{[\![S]\!] \mid S \text{ is a regular } L\text{-controlled language}\}$ is the set of relations that can be given by L-controlled synchronizations. Let $\mathcal{C}$ be a class of relations, we say L *synchronizes* $\mathcal{C}$ if $\textsc{Rel}(L) \subseteq \mathcal{C}$.

▶ **Definition 2** (lag, shift, shiftlag [6]). Given a word $w \in \mathbf{2}^*$, a position $i \leq |w|$, and $\gamma \in \mathbb{N}$. We say i is *γ-lagged* if $|\#_1(w[1, i]) - \#_2(w[1, i])| = \gamma$, and likewise, we define *$>\gamma$-lagged* and *$<\gamma$-lagged*. A *shift* of w is a position $i \in \{1, \dots, |w| - 1\}$ such that $w[i] \neq w[i + 1]$. Two shifts $i < j$ are *consecutive* if there is no shift l such that $i < l < j$. Let $\mathit{shift}(w)$ be the number of shifts in w, let $\mathit{lag}(w)$ be the maximum lag of a position in w, and let $\mathit{shiftlag}(w)$ be the maximum $n \in \mathbb{N}$ such that w contains n consecutive shifts which are $>n$-lagged.

We lift these notions to languages by taking the supremum in $\mathbb{N} \cup \{\infty\}$, e.g., $\mathit{shift}(L) = \sup\{\mathit{shift}(w) \mid w \in L\}$, and likewise for $\mathit{lag}(L)$ and $\mathit{shiftlag}(L)$.

The following characterizations for well known subclasses of rational relations were shown in [6]. Recall, rational relations are definable by asynchronous finite automata, automatic relations by synchronous finite automata, and recognizable relations are definable as finite unions of products of regular languages. We omit a formal definition of these models since it is not relevant to this paper.

▶ **Theorem 3** ([6]). *Let $L \subseteq \mathbf{2}^*$ be a regular language. Then:*
1. *L synchronizes recognizable relations iff $\mathit{shift}(L) < \infty$,*
2. *L synchronizes automatic relations iff $\mathit{shiftlag}(L) < \infty$,*
3. *L synchronizes rational relations.*

For ease of presentation, let $\Sigma_{\mathtt{io}}$, $\Sigma_{\mathtt{i}}$, and $\Sigma_{\mathtt{o}}$ be short for $\mathbf{2} \times \Sigma$, $\{1\} \times \Sigma$, and $\{2\} \times \Sigma$, respectively. If convenient, we use distinct symbols for input and output, instead of symbols annotated with 1 or 2.

For the results shown in this paper, it is useful to lift some notions introduced in [6] from words and languages over $\mathbf{2}$ to words and languages over $\Sigma_{\mathtt{io}}$.

▶ **Definition 4.** We lift the notions of *lag*, *shift*, and *shiftlag* from words and languages over $\mathbf{2}$ to words and languages over $\Sigma_{\mathtt{io}}$ in the natural way.

Furthermore, given a language $T \subseteq \Sigma_{\mathtt{io}}^*$, we say a word $w \in \Sigma_{\mathtt{io}}^*$ is T-controlled if $w \in T$. A language $S \subseteq \Sigma_{\mathtt{io}}^*$ is T-controlled if all its words are, namely, if $S \subseteq T$.

Automata on finite words. We fix our notations concerning finite automata on finite words. A *nondeterministic finite automaton* (*NFA*) is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$, where Q is a finite set of states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and $F \subseteq Q$ is the set of final states. A *run* ρ of $\mathcal{A}$ on $w = a_1 \dots a_n \in \Sigma^*$ is a sequence of states $p_0 p_1 \dots p_n$ such that $(p_i, a_{i+1}, p_{i+1}) \in \Delta$ for all $i \in \{0, \dots, n-1\}$. Shorthand, we write $\mathcal{A} : p_0 \xrightarrow{w} p_n$. A run is *accepting* if it starts in q_0 and ends in a state from F. The language *recognized* by $\mathcal{A}$, written $L(\mathcal{A})$, is the set of words $w \in \Sigma^*$ that admit an accepting run of $\mathcal{A}$ on w. For $q \in Q$, let $\mathcal{A}_q$ denote the NFA obtained from $\mathcal{A}$ by setting its initial state to q. The class of languages recognized by NFAs is the class of regular languages. An NFA is *deterministic* (a *DFA*) if for each state $q \in Q$ and $a \in \Sigma$ there is at most one outgoing transition. In this case, it is more convenient to express Δ as a (partial) function $\delta : Q \times \Sigma \to Q$. Furthermore, let δ^* denote the usual extension of δ from letters to words.

We introduce some notions only applicable if an NFA recognizes a set of synchronizations. Given a regular $S \subseteq \Sigma_{\mathtt{io}}^*$, let $\mathcal{A} = (Q, \Sigma_{\mathtt{io}}, q_0, \Delta, F)$ be an NFA that recognizes S. We define $Q^{\mathtt{i}} = \{p \in Q \mid \exists a \in \Sigma, q \in Q : (p, (1, a), q) \in \Delta\}$ and $Q^{\mathtt{o}} = \{p \in Q \mid \exists a \in \Sigma, q \in Q : (p, (2, a), q) \in \Delta\}$ as the sets of states that have outgoing transitions from which input and output can be consumed, respectively. If $(Q^{\mathtt{i}}, Q^{\mathtt{o}})$ is a partition of Q, we write $Q = Q^{\mathtt{i}} \mathbin{\dot\cup} Q^{\mathtt{o}}$. We call $\mathcal{A}$ *sequential* if $\mathcal{A}$ is deterministic, and $Q = Q^{\mathtt{i}} \mathbin{\dot\cup} Q^{\mathtt{o}}$, and each $q \in Q^{\mathtt{o}}$ has at most one outgoing transition. For short, we refer to a sequential DFA as *sDFA*. Finally, we define the input automaton $\mathcal{A}_D$ of $\mathcal{A}$ as $(Q, \Sigma, q_0, \Delta', F)$, where $\Delta' = \{(p, a, q) \mid \mathcal{A} : p \xrightarrow{w} q \text{ and } \pi_{\mathtt{i}}(w) = a \in \Sigma\}$. A comparison to standard transducer models is given in the next section.

3 Uniformization problems

A *uniformization* of a relation $R \subseteq \Sigma^* \times \Sigma^*$ is a complete function $f_R : \mathrm{dom}(R) \to \Sigma^*$ with $(u, f_R(u)) \in R$ for all $u \in \mathrm{dom}(R)$. If such a function is given as a relation R_f, we write $R_f \subseteq_{\mathtt{u}} R$ to indicate that R_f is a uniformization of R.

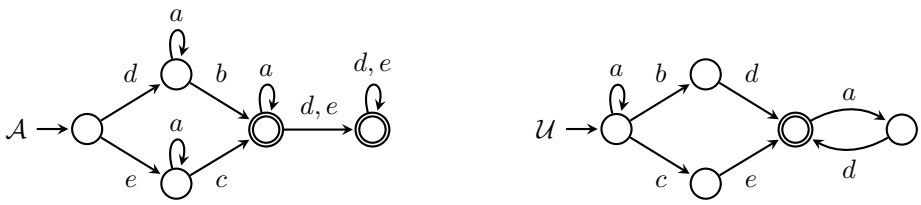


Figure 1 Cf. Ex.6; $S = L(\mathcal{A})$ and $U = L(\mathcal{U})$, we have $[\![U]\!] \subseteq_{\mathtt{u}} [\![S]\!]$.

▶ **Definition 5** (Resynchronized uniformization problem)**.** The *resynchronized uniformization problem* asks, given a regular source language $S \subseteq \Sigma^*_{\mathtt{i}\mathtt{o}}$ and a regular target language $T \subseteq \Sigma^*_{\mathtt{i}\mathtt{o}}$, whether there exists a regular language $U \subseteq T$ recognized by a sequential DFA such that $[\![U]\!] \subseteq_{\mathtt{u}} [\![S]\!]$.

▶ **Example 6.** Let $\Sigma_{\mathtt{i}} = \{a, b, c\}$ and $\Sigma_{\mathtt{o}} = \{d, e\}$, let $S \subseteq \Sigma^*_{\mathtt{i}\mathtt{o}}$ be given by $\mathcal{A}$ depicted in Fig. 1. The recognized relation is $[\![S]\!] = \{(a^i b a^j, d(d+e)^k) \mid i, j, k \geq 0\} \cup \{(a^i c a^j, e(d+e)^k) \mid i, j, k \geq 0\}$. Furthermore, let $T = \Sigma^*_{\mathtt{i}}(\Sigma_{\mathtt{i}}\Sigma_{\mathtt{o}})^+$. A T-controlled uniformization U is given by the sequential DFA $\mathcal{U}$ depicted in Fig. 1. The recognized relation is $[\![U]\!] = \{(a^i b a^j, dd^j) \mid i, j, k \geq 0\} \cup \{(a^i c a^j, ed^j) \mid i, j \geq 0\}$.

Comparing our definition of sequential DFAs with standard transducer models we notice that sequential transducers directly correspond to sequential DFAs. See, e.g., [1] for an introduction to transducers. Our model can be modified to correspond to subsequential transducers (which can make a final output after the word has ended) by slightly modifying the representation of the relation by adding a dedicated endmarker in the usual way.

In the remainder it is implicitly assumed that every given source and target language is represented with endmarkers, thus our stated results correspond to uniformization by subsequential transducers.

Our main result is the decidability of the resynchronized uniformization problem for a given automatic relation and a given set of synchronizations controlled by a language that synchronizes automatic relations. In Sec. 4 we see that our decidability result is obtained by a reduction to the following simpler uniformization problem.

▶ **Definition 7** (Subset uniformization problem)**.** The *subset uniformization problem* asks, given a regular language $S \subseteq \Sigma^*_{\mathtt{i}\mathtt{o}}$, whether there exists a regular language $U \subseteq S$ recognized by a sequential DFA such that $[\![U]\!] \subseteq_{\mathtt{u}} [\![S]\!]$.

The notion of subset uniformization directly corresponds to the notion of sequential $\mathbb{I}$-uniformization introduced in [7]. It was shown that deciding the sequential $\mathbb{I}$-uniformization problem reduces to deciding which player has a winning strategy in a safety game between In and Out. Hence, we directly obtain the following result.

▶ **Theorem 8** ([7])**.** *The subset uniformization problem is decidable.*

Now that we have formulated our uniformization problems, we link these to known uniformization problems. Asking whether a relation has a $\Sigma^*_{\mathtt{i}\mathtt{o}}$-controlled subsequential uniformization is equivalent to asking whether it has a uniformization by an arbitrary subsequential transducer. Asking whether a relation has a $(\Sigma_{\mathtt{i}}\Sigma_{\mathtt{o}})^*(\Sigma^*_{\mathtt{i}} + \Sigma^*_{\mathtt{o}})$- resp. $\Sigma^*_{\mathtt{i}}\Sigma^*_{\mathtt{o}}$-controlled subsequential uniformization is equivalent to asking whether it has a uniformization by a synchronous subsequential transducer resp. by a transducer that reads the complete input before producing output.

Table 1 Overview over decidability results. The columns list the type of relation to be uniformized. The rows list the type of synchronization used as uniformization parameter; the upper three rows list fixed languages of synchronizations, the lower three rows list parameter classes, where 'rational' means the given set of allowed synchronizations is controlled by an arbitrary synchronization language, 'automatic' (resp. 'recognizable') means the given set of allowed synchronizations is controlled by a synchronization language that synchronizes automatic (resp. recognizable) relations.

sync. \ relation	rational	deterministic rational	finite-valued	automatic	recognizable
$\Sigma^*_{\mathtt{i}\circledcirc}$	undec. [3]	dec. [7]	dec. [7]	dec. [3]	dec.
$(\Sigma_{\mathtt{i}}\Sigma_{\circledcirc})^*(\Sigma^*_{\mathtt{i}} + \Sigma^*_{\circledcirc})$	undec. [3]	?	?	dec. [2]	dec.
$\Sigma^*_{\mathtt{i}}\Sigma^*_{\circledcirc}$	?	?	?	dec. [3]	dec.
rational	undec.	?	?	?	dec.
automatic	undec.	?	?	**dec.**	dec.
recognizable	?	?	?	dec.	dec.

Table 1 provides an overview over known and new decidability results of the resynchronized uniformization problem for different types of relations and synchronization parameters. Our main result is the decidability for a given automatic relation and a given set of allowed synchronizations that is controlled by a synchronization language that synchronizes automatic relations. The decidability results in the rightmost column can be shown by a simple reduction to the subset uniformization problem which is presented in the full version of the paper. The other entries in the lower three rows are simple consequences of the results presented in the upper three rows resp. our main result.

Regarding the table entry where the relation is automatic and a desired uniformizer is $(\Sigma_{\mathtt{i}}\Sigma_{\circledcirc})^*(\Sigma^*_{\mathtt{i}} + \Sigma^*_{\circledcirc})$-controlled, there is an alternative formulation of the decision problem in the case that the given relation is $(\Sigma_{\mathtt{i}}\Sigma_{\circledcirc})^*(\Sigma^*_{\mathtt{i}} + \Sigma^*_{\circledcirc})$-controlled (the usual presentation for automatic relations, e.g., by a synchronous transducer). In this case the problem can also be stated as the question whether the relation has a subset uniformization.

We now generalize this to Parikh-injective synchronization languages. Given some $L \subseteq \mathbf{2}^*$, let $\Pi_L : L \to \mathbb{N}^2$ be the function that maps a word $w \in L$ to its *Parikh image*, that is to the vector $(\#_1(w), \#_2(w))$. We say L is *Parikh-injective* if Π_L is injective.

▶ **Proposition 9.** Let $L \subseteq \mathbf{2}^*$ be a regular Parikh-injective language, let $S \subseteq \Sigma^*_{\mathtt{i}\circledcirc}$ be an L-controlled regular language and let $T = \{w \in \Sigma^* \mid w \text{ is } L\text{-controlled}\}$. Every T-controlled uniformization of S is a subset uniformization of S.

Given L, S and T as in Proposition 9, it directly follows that the resynchronized uniformization problem is equivalent to the subset uniformization problem, which is decidable by Theorem 8.

4 Automatic uniformizations of automatic relations

Here we present our main result stating that it is decidable whether a given automatic relation has a uniformization by a subsequential transducer whose induced set of synchronizations is controlled by a given regular language that synchronizes automatic relations.

▶ **Theorem 10.** *Given a regular source language with finite shiftlag and a regular target language with finite shiftlag. Then, the resynchronized uniformization problem is decidable.*

In [6], it is shown that $(12)^*(1^* + 2^*)$ is an effective canonical representative of the class RL_{FSL} of regular languages with finite *shiftlag*. Meaning that for every $L \in \mathsf{RL}_{FSL}$ and every $R \in \textsc{Rel}(L)$, there is an effectively constructible $(12)^*(1^* + 2^*)$-controlled regular language S so that $[\![S]\!] = R$.

In the remainder of this section, let $S \subseteq \Sigma^*_{\mathtt{io}}$ be a regular source language with finite *shiftlag*. Also, let S_{can} be the equivalent $(12)^*(1^*+2^*)$-controlled language with $[\![S_{can}]\!] = [\![S]\!]$. Furthermore, let $T \subseteq \Sigma^*_{\mathtt{io}}$ be a regular target language with finite *shiftlag*.

▶ **Assumption 11.** We assume that S_{can} is recognized by a DFA $\mathcal{A} = (Q_\mathcal{A}, \Sigma_{\mathtt{io}}, q_0^\mathcal{A}, \Delta_\mathcal{A}, F_\mathcal{A})$, T is recognized by a DFA $\mathcal{B} = (Q_\mathcal{B}, \Sigma_{\mathtt{io}}, q_0^\mathcal{B}, \Delta_\mathcal{B}, F_\mathcal{B})$ and $\mathit{shiftlag}(T) < n$.

For notational convenience, given $x \in \Sigma^*_{\mathtt{i}}$ and $y \in \Sigma^*_{\mathtt{o}}$, we write $\delta^*_\mathcal{A}(q, (x, y))$ to mean $\delta^*_\mathcal{A}(q, w)$, where $w \in \Sigma_{\mathtt{io}}$ is the canonical synchronization of x and y, i.e., w is the $(12)^*(1^*+2^*)$-controlled synchronization of the pair (x, y).

The remainder of this section is devoted to the proof of Theorem 10. The proof is split in two main parts; the goal of the first part is to show that if S has a T-controlled uniformization by an sDFA, then S has a T_k-controlled uniformization by an sDFA for a regular $T_k \subseteq T$ that is less complex than T, cf. Lemma 23. The goal of the second part is to show that the set $T_k(S)$ defined by $\{w \mid w \in T_k \text{ and } [\![w]\!] \in [\![S]\!]\}$ is regular and computable (due to the form of T_k), cf. Lemma 24. Then, to conclude the proof, we show that the question whether S has a T-controlled uniformization by an sDFA can be reduced to the question whether $T_k(S)$ has a subset uniformization by an sDFA, which is decidable by Theorem 8.

Towards giving an exact description of T_k, consider the following auxiliary lemma characterizing the form of regular synchronization languages with finite *shiftlag*. Given $\nu \in \mathbb{N}$, we denote by $L_{\leq\nu}$ the regular set of words over $\mathbf{2}$ with $\leq\nu$-lagged positions, i.e., $L_{\leq\nu} = \{u \in \mathbf{2}^* \mid \mathit{lag}(u) \leq \nu\}$; we denote by $T_{\leq\nu}$ the regular set of words over $\Sigma_{\mathtt{io}}$ with $\leq\nu$-lagged positions, i.e., $T_{\leq\nu} = \{w \in \Sigma^*_{\mathtt{io}} \mid \mathit{lag}(w) \leq \nu\}$.

▶ **Lemma 12** ([6]). *Given a regular language $L \subseteq \mathbf{2}^*$ with $\mathit{shiftlag}(L) < m$. It holds that $L \subseteq L_{\leq\nu} \cdot (1^* + 2^*)^m$ with ν chosen as $2\,(m(|Q| + 1) + 1)$, where Q is the state set of an NFA recognizing L.*

Clearly, this lemma can be lifted to regular languages over $\Sigma_{\mathtt{io}}$. Based on Asm. 11 and Lemma 12, we can make the following assumption.

▶ **Assumption 13.** Assume that $T \subseteq T_{\leq\gamma} \cdot (\Sigma^*_{\mathtt{i}} + \Sigma^*_{\mathtt{o}})^n$ with $\gamma = 2\,(n(|Q_\mathcal{B}| + 1) + 1)$.

Now, we can be more specific about $T_k \subseteq T$.

▶ **Definition 14.** For $i \geq 0$, let T_i be the set $T \cap \left(T_{\leq\gamma} \cdot (\Sigma^*_{\mathtt{i}} + \Sigma^{\leq i}_{\mathtt{o}})^n\right)$, that is, the set of $w \in T$ such that after a position in w is more than γ-lagged, the number of output symbols per block is at most i.

Our aim is to show that there is a bound k such that S has either a T_k-controlled uniformization by an sDFA or no T-controlled uniformization by an sDFA. From now on, we call an sDFA implementing a uniformization simply a uniformizer.

The main difficulty in solving the resynchronized uniformization problem is that in general a uniformizer can have unbounded lag, because the waiting time between shifts can be arbitrarily long. The key insight for the proof is that if such a long waiting time for a shift from input to output is necessary, then, in order to determine the next output block, it is not necessary to store the complete input that is ahead. We show that it suffices to consider an abstraction of the input that is ahead. Therefore we will introduce input profiles based on state transformation trees we define below.

Similarly, to deal with the situation where there is a long waiting time for a shift from output to input, we introduce output profiles as an abstraction of output that is ahead.

The bound on the length of output blocks will be chosen based on the profiles. Before defining profiles, we introduce some necessary definitions and notions.

Trees. A *finite unordered unranked tree* over an alphabet, a tree for short, is a finite non-empty directed graph with a distinguished root node, such that for any node, there exists exactly one path from the root to this node. Additionally, a mapping from the nodes of the graph to the alphabet is given. More formally, a *tree* t over Σ is given by a tuple (V_t, E_t, v_t, val_t), where V_t is a non-empty set of nodes, $E_t \subseteq V_t \times V_t$ is a set of edges, v_t is the root of t, also denoted $root(t)$, and val_t is a mapping $V_t \to \Sigma$. Furthermore, it is satisfied that any node is reached by a unique path from the root. Let T_Σ denote the set of all trees over Σ. We only distinguish trees up to isomorphism.

Given a tree t and a node v of t, let $t|_v$ denote the *subtree* of t rooted at v.

An $a \in \Sigma$ can also be seen as a tree $a \in T_\Sigma$ defined by $(\{v\}, \emptyset, v, val_a)$, where $val_a(v) = a$.

For two trees t_1 and t_2 with $val_{t_1}(root(t_1)) = val_{t_2}(root(t_2))$, i.e., with the same root label, we define $t_1 \circ t_2$ as the tree t given by $(V_t, E_t, root(t_1), val_t)$, where $V_t = V_{t_1} \cup V_{t_2} \setminus \{root(t_2)\}$, $E_t = E_{t_1} \cup \{(root(t), v) \mid (root(t_2), v) \in E_{t_2}\} \cup (E_{t_2} \setminus \{(root(t_2), v) \in E_{t_2}\})$ and val_t as $val_{t_1} \cup val_{t_2}$ over nodes in V_t (assuming $V_{t_1} \cap V_{t_2} = \emptyset$).

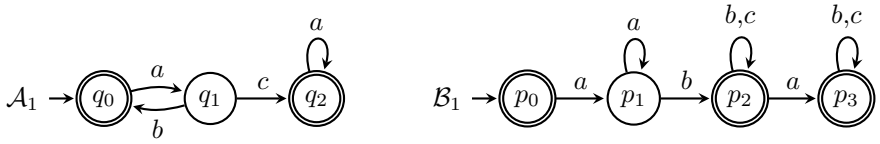
Given $a \in \Sigma$ and trees $t_1, \ldots, t_n$, we define $a(t_1 \ldots t_n)$ to be the tree (V_t, E_t, r, val_t), where $V_t = \bigcup_{i=1}^n V_{t_i} \cup \{r\}$ with a new node r, $E_t = \bigcup_{i=1}^n E_{t_i} \cup \{(r, root(t_i)) \mid 1 \leq i \leq n\}$ and val_t is defined as $val_t(r) = a$ and $\bigcup_{i=1}^n val_{t_i}$ (assuming $V_{t_i} \cap V_{t_j} = \emptyset$ for all $i \neq j$).

State transformation trees. Now that we have fixed our notations, we explain what kind of information we want to represent using state transformation trees. Basically, for an input segment that is ahead and causes lag, we are interested in how the input segment can be combined with output segments of same or smaller length and how this output can be obtained.

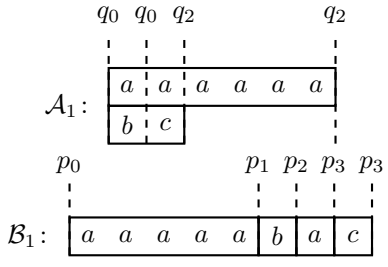
In the following we give an intuitive example.

▶ **Example 15.** Let $\Sigma_{\mathtt{i}} = \{a\}$ and $\Sigma_{\mathtt{o}} = \{b, c\}$. Consider the language $S_1 \subseteq \Sigma_{\mathtt{io}}^*$ given by the DFA $\mathcal{A}_1$ depicted in Fig. 2a, and the language $T_1 \subseteq \Sigma_{\mathtt{io}}^*$ given by the DFA $\mathcal{B}_1$ depicted in Fig. 2a. As we can see, S_1 is $(12)^*(1^* + 2^*)$-controlled, thus, already in its canonical form, and T_1 is $1^*2^*1^*2^*$-controlled. Both languages have finite *shiftlag*.

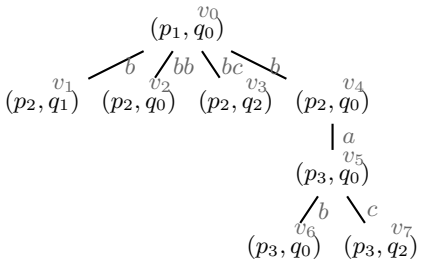
Generally, a T_1-controlled uniformizer of S_1 can have arbitrary large lag. We take a look at the runs starting from q_0 in $\mathcal{A}_1$ and starting from p_0 in $\mathcal{B}_1$ that the computation of such a uniformizer can induce. However, $\mathcal{A}_1$ can only be simulated on the part where the lag is recovered, but arbitrarily large lag can occur, thus our goal is to find an abstraction of the part that causes lag. E.g., assume that such a uniformizer reads aa without producing output. Towards defining an abstraction of aa, we are interested in how aa could be combined with outputs of same or smaller length and how these outputs could be produced by some T_1-controlled uniformizer. Such a uniformizer could read some more as and eventually must produce output. Reading as leads from p_0 to p_1 in $\mathcal{B}_1$. There are a few possibilities how output of length at most two can be produced such that it is valid from p_1 and the simulation from q_0 can be continued. It is possible to output b ($\delta^*_{\mathcal{B}_1}(p_1, b) = p_2$, $\delta^*_{\mathcal{A}_1}(q_0, aba) = q_1$), bb ($\delta^*_{\mathcal{B}_1}(p_1, bb) = p_2$, $\delta^*_{\mathcal{A}_1}(q_0, abab) = q_0$) or bc ($\delta^*_{\mathcal{B}_1}(p_1, bc) = p_2$, $\delta^*_{\mathcal{A}_1}(q_0, abac) = q_2$). Alternatively, it is possible to output b ($\delta^*_{\mathcal{B}_1}(p_1, b) = p_2$, $\delta^*_{\mathcal{A}_1}(q_0, ab) = q_0$) read another a ($\delta^*_{\mathcal{B}_1}(p_2, a) = p_3$) and then produce b ($\delta^*_{\mathcal{B}_1}(p_3, b) = p_3$, $\delta^*_{\mathcal{A}_1}(q_0, ab) = q_0$) or c ($\delta^*_{\mathcal{B}_1}(p_3, c) = p_3$, $\delta^*_{\mathcal{A}_1}(q_0, ac) = q_2$). We see that the outputs bb and bc can each be obtained in two different ways. Namely, as one single output block, or as two output blocks with an input block in



(a) $\Sigma_{\mathtt{i}} = \{a\}$, $\Sigma_{\mathtt{o}} = \{b, c\}$. $\mathcal{A}_1$ recognizes S_1, $\mathcal{B}_1$ recognizes T_1. S_1 is $(12)^*(1^* + 2^*)$-controlled and T_1 is $1^*2^*1^*2^*$-controlled, thus both have finite *shiftlag*. S_1 is already in the canonical form.



(b) Runs of $\mathcal{A}_1$ and $\mathcal{B}_1$ on synchronizations of $(aaaaaa, bc)$. $\mathcal{A}_1$ runs on the canonical synchronization, i.e., on *abacaaaa*. To illustrate this, input and output are drawn one above the other.



(c) $\mathrm{STT}^1(aa, p_1, q_0)$. The combination of both runs shown in Fig. 2b is reflected by the rightmost path in the state transformation tree.

Figure 2 A source language S_1 and a target language T_1 are given in Fig. 2a. A pair and two different synchronizations of said pair as well as runs are given in Fig. 2b. The state transformation tree $\mathrm{STT}^1(aa, p_1, q_0)$ is given in Fig. 2c, its edges are labeled with the respective associated words and its vertices are named for easier reference in Ex. 15. For a formal definition of STTs see Def. 16, for an explanation for this specific tree see Ex. 15.

between (w.r.t. $\mathcal{B}_1$, we do not care about the number of blocks w.r.t. $\mathcal{A}_1$). The maximal number of considered output blocks (w.r.t. the target synchronization) is parameterized in the formal definition.

We take a look at the tree in Fig. 2c, this tree contains all the state transformations that can be induced by the described possibilities. The possibilities to produce output in one single block is reflected by the edges (v_0, v_1), (v_0, v_2) and (v_0, v_3) representing the state transformation induced by the respective output block. The possibilities to produce output in two blocks is reflected by the edges (v_0, v_4) representing the state transformation induced by the first output block, (v_4, v_5) representing the state transformation induced by the intermediate input block, (v_5, v_6) and (v_5, v_7) representing the state transformation induced by the respective second output block.

Now that we have given some intuition, we formally introduce input state transformation trees, a graphical representation of the construction of input state transformation trees is given in Fig. 3. As seen in the example, each edge of such a tree represents the state transformation induced by an output resp. input block, alternatively.

▶ **Definition 16** (Input state transformation tree). For $i \geq 0$, $p \in Q_{\mathcal{B}}$, $q \in Q_{\mathcal{A}}$ and $x \in \Sigma_{\mathtt{i}}^*$, the *state transformation tree* $\mathrm{STT}^i(x, p, q)$ is a tree over $Q_{\mathcal{B}} \times Q_{\mathcal{A}}$ defined inductively.

- For $i = 0$, the tree $\mathrm{STT}^0(x, p, q)$ is built up as follows.
  Let $\mathrm{Reach}_0 \subseteq Q_{\mathcal{B}} \times Q_{\mathcal{A}}$ be the smallest set such that $(p', q') \in \mathrm{Reach}_0$ if there is some $y \in \Sigma_{\mathtt{o}}^*$ with $|y| \leq |x|$ such that $\delta_{\mathcal{A}}^*(q, (x, y)) = q'$ and $\delta_{\mathcal{B}}^*(p, y) = p'$.
  (This set represents state transformations induced by output blocks that fully consume x.)
  Then the tree $\mathrm{STT}^0(x, p, q)$ is defined as $(p, q)(r_1 \dots r_n)$ for $\mathrm{Reach}_0 = \{r_1, \dots, r_n\}$, meaning it contains a child for every state transformation that can be induced w.r.t. $\mathcal{A}$ and $\mathcal{B}$ starting from q and p, respectively, by the input segment x together with an output segment that consumes x (w.r.t. $\mathcal{A}$) consisting of a single output block (w.r.t. $\mathcal{B}$).

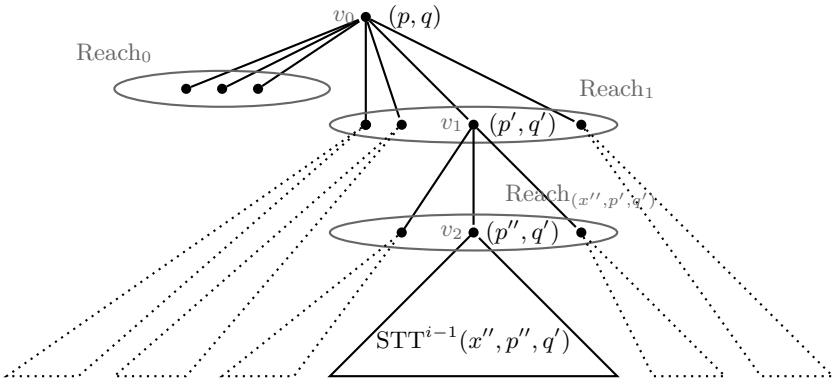


Figure 3 Schema of the input state transformation tree $\text{STT}^i(x, p, q)$ for some $i > 0$. Cf. Def. 16. Let $x'x''$ be a factorization of x with $x', x'' \in \Sigma_{\mathtt{i}}^+$, and let $y \in \Sigma_{\mathtt{o}}^+$ be such that $|x'| = |y|$ and $\delta_{\mathcal{A}}^*(q, (x', y)) = q'$ and $\delta_{\mathcal{B}}^*(p, y) = p'$, and let $\delta_{\mathcal{B}}^*(p', w) = p''$ for some $w \in \Sigma_{\mathtt{i}}^+$, then $\text{STT}^i(x, p, q)$ contains a path $v_0v_1v_2$ labeled $(p, q)(p', q')(p'', q')$ such that v_0 is the root, v_1 is the root of $t^{i-1}_{(x'',p',q')}$, and v_2 is the root of $\text{STT}^{i-1}(x'', p'', q')$.

- For $i > 0$, the tree $\text{STT}^i(x, p, q)$ is built up as follows.
  Let $\text{Reach}_1 \subseteq \Sigma_{\mathtt{i}}^* \times Q_{\mathcal{B}} \times Q_{\mathcal{A}}$ be the smallest set such that $(x'', p', q') \in \text{Reach}_1$ if
  - $x = x'x''$ with $x'' \in \Sigma_{\mathtt{i}}^+$ for an $x' \in \Sigma_{\mathtt{i}}^+$ such that there is a $y \in \Sigma_{\mathtt{o}}^+$ with $|y| = |x'|$, and
  - $\delta_{\mathcal{A}}^*(q, (x', y)) = q'$ and $\delta_{\mathcal{B}}^*(p, y) = p'$.

  (This set represents state transformations induced by output blocks that partially consume x.)
  For $(x'', p', q') \in \text{Reach}_1$, let $\text{Reach}_{(x'',p',q')} \subseteq \Sigma_{\mathtt{i}}^* \times Q_{\mathcal{B}} \times Q_{\mathcal{A}}$ be the smallest set such that $(x'', p'', q') \in \text{Reach}_{(x'',p',q')}$ if $\delta_{\mathcal{B}}^*(p', w) = p''$ for some $w \in \Sigma_{\mathtt{i}}^+$.
  (These sets represents state transformations induced by intermediate input blocks.)
  Furthermore, let the tree $t^{i-1}_{(x'',p',q')}$ be defined as $(p', q')(\text{STT}^{i-1}r_1 \dots \text{STT}^{i-1}r_n)$ for $\text{Reach}_{(x'',p',q')} = \{r_1, \dots, r_n\}$.
  Then the tree $\text{STT}^i(x, p, q)$ is defined as

  $$\text{STT}^0(x, p, q) \circ (p, q)(t^{i-1}_{s_1} \dots t^{i-1}_{s_n})$$

  for $\text{Reach}_1 = \{s_1, \dots, s_n\}$, meaning it contains a path for every sequence of state transformations that can be induced w.r.t. $\mathcal{A}$ and $\mathcal{B}$ starting from q and p, respectively, by the input segment x together with an output segment that consumes x (w.r.t. $\mathcal{A}$) consisting of at most $i + 1$ output blocks (w.r.t. $\mathcal{B}$). Additionally, for output segments that have a common prefix of output blocks the state transformations induced by the common prefix of blocks are represented by the same nodes in the tree.

Intuitively, edges in such a tree are associated with the words that induced the state transformation, e.g., as shown in Fig 2c.

Given a tree as in Def. 16, the maximal degree of such a tree depends on the input word used as parameter. Our goal is to have state transformation trees where the maximum degree is independent of this parameter. Therefore, we introduce *reduced trees*. The idea is that if for some input word different outputs induce the same state transformations then only one representation is kept in the input state transformation tree.

▶ **Definition 17** (Reduced tree). A tree $t \in T_\Sigma$ over some alphabet Σ is called *reduced* if for each node v there exist no two children u, u' of v such that the subtrees rooted at u and u' are isomorphic.

For a tree $t \in T_\Sigma$, let $red(t) \in T_\Sigma$ denote its reduced variant. The reduced variant of a tree can easily be obtained by a bottom-up computation where for each node duplicate subtrees rooted at its children are removed.

Note that for each i, the set of reduced input state transformation trees with parameter i is a finite set.

Hitherto, we have discussed how to capture state transformations induced by an input word together with output words of same or smaller length. Additionally, we need to capture state transformations induced by an output word together with input words of same or smaller length. Therefore, we introduce a notion similar to input state transformation trees, namely, *output state transformation trees*. A formal definition can be found in the full version.

Furthermore, we need a notion that captures state transformations that can be induced by an input resp. output word alone, see Def. 18 below. Then, we are ready to define profiles.

▶ **Definition 18** (State transformation function)**.** For each $w \in \Sigma_{\mathtt{i}}^* \cup \Sigma_{\mathtt{o}}^*$, we define the function $\tau_w \colon Q_{\mathcal{B}} \to Q_{\mathcal{B}}$ with $\tau_w(p) = q$ if $\delta_{\mathcal{B}}^*(p, w) = q$ called *state transformation function w.r.t. w*.

Profiles. Recall, $T \subseteq T_{\leq\gamma} \cdot (\Sigma_{\mathtt{i}}^* + \Sigma_{\mathtt{o}}^*)^n$, and our goal is to show that there is a bound k such that it suffices to focus on constructing T_k-controlled uniformizers instead of T-controlled uniformizers, meaning that we can focus on uniformizers in which the length of output blocks is bounded by k after the lag has exceeded γ at some point.

The core of the proof is to show that if the lag between input and output becomes very large ($\gg \gamma$), it is not necessary to consider the complete input that is ahead to determine the next output block, but an abstraction (in the form of profiles) suffices. Note that if the lag has exceeded γ at some point the number of remaining output blocks is at most $\lceil n/2 \rceil$.

As a result, given an input word $x \in \Sigma_{\mathtt{i}}^*$, we are interested in the state transformation that is induced by $(x, \pi_{\mathtt{o}}(w))$ in $\mathcal{A}$ (recognizing S_{can}) and by w in $\mathcal{B}$ (recognizing T) for each word $w \in \Sigma_{\mathtt{io}}^*$ such that $|\pi_{\mathtt{o}}(w)| \leq |x|$ and $shift(w) \leq \lceil n/2 \rceil$. In words, we are interested in the state transformations that can be induced by x together with outputs of same or smaller length that are composed of at most $\lceil n/2 \rceil$ different output blocks.

For $x \in \Sigma_{\mathtt{i}}^*$, this kind of information is accurately represented by the set of all reduced input state transformation trees with parameters x and $\lceil n/2 \rceil$.

The same considerations with switched input and output roles apply for an output word $y \in \Sigma_{\mathtt{o}}^*$.

▶ **Definition 19** (Input profile)**.** For $x \in \Sigma_{\mathtt{i}}^*$, we define its *profile* P_x as $(\tau_x, \mathrm{STT}_x^{\lceil n/2 \rceil})$, where

$$\mathrm{STT}_x^{\lceil n/2 \rceil} = \bigcup_{(p,q) \in Q_{\mathcal{B}} \times Q_{\mathcal{A}}} \{ red\big(\mathrm{STT}^{\lceil n/2 \rceil}(x, p, q)\big) \}.$$

Similarly, we define *output profiles*, a formal definition can be found in the full version.

A note on the number of different profiles. Profiles are based on reduced STTs with parameter $\lceil n/2 \rceil$, where n bounds $shiftlag(T)$. The size of the set of these STTs is non-elementary in n, hence also the number of profiles. This implies a non-elementary complexity of our decision procedure.

Furthermore, let $\mathcal{P}_{\mathtt{i}}$ be the set $\bigcup_{x \in \Sigma_{\mathtt{i}}^*} \{P_x\}$ of all input profiles and $\mathcal{P}_{\mathtt{o}}$ be the set $\bigcup_{y \in \Sigma_{\mathtt{o}}^*} \{P_y\}$ of all output profiles. For a $P \in \mathcal{P}_{\mathtt{i}} \cup \mathcal{P}_{\mathtt{o}}$, let z be a *representative* of P if z is a shortest word such that $P = P_z$.

We show that from the profiles of two words x_1 and x_2 one can compute the profile of the word $x_1 x_2$. Hence, the set of profiles can be equipped with a concatenation operation, i.e., for words x_1 and x_2 we let $P_{x_1} P_{x_2} = P_{x_1 x_2}$. We obtain the following.

▶ **Lemma 20.** *The set of input profiles is a monoid with concatenation; the set of output profiles is a monoid with concatenation.*

A word $x \in \Sigma_{\mathtt{i}}^*$ and its profile P_x are called *idempotent* if $P_x = P_{xx}$. As a consequence of Ramsey's Theorem (see e.g., [5]) we obtain the following lemma.

▶ **Lemma 21** (Consequence of Ramsey)**.** *There is a computable $r \in \mathbb{N}$ such that each word $x \in \Sigma_{\mathtt{i}}^*$ with $|x| \geq r$ contains a non-empty idempotent factor for the concatenation of profiles.*

Now, we have the right tools to prove that the existence of a T-controlled uniformizer implies that there also exists a T_k-controlled uniformizer for a computable k. For the remainder, we fix two bounds.

▶ **Assumption 22.** Assume r_1 is chosen as in Lemma 21 and r_2 is chosen as the smallest bound on the length of representatives of output profiles. Wlog, assume $r_1, r_2 > \gamma$.

Finally, we are ready to prove the key lemma, that is, Lemma 23, which shows that it is sufficient to consider uniformizers in which the length of output blocks is bounded.

Recall, a uniformizer works asynchronously, which leads to large lag. First, we show that if the output is lagged more than r_1 symbols, meaning, the input that is ahead contains an idempotent factor, it suffices to consider output blocks whose length depends on the idempotent factor. Secondly, we show that it suffices to consider uniformizers in which the output is ahead at most r_2 symbols. The combination of both results yields Lemma 23.

Recall, by Asm. 13, $T \subseteq T_{\leq\gamma} \cdot (\Sigma_{\mathtt{i}}^* + \Sigma_{\mathtt{o}}^*)^n$ and by Def. 14, $T_i = T \cap \left(T_{\leq\gamma} \cdot (\Sigma_{\mathtt{i}}^* + \Sigma_{\mathtt{o}}^{\leq i})^n\right)$ for $i \geq 0$.

▶ **Lemma 23.** *If S has a T-controlled uniformizer, then S has a T_k-controlled uniformizer for a computable $k \geq 0$.*

The proof of the above lemma yields that k can be chosen as $r_1 + r_2$. This concludes the first part of the proof of Theorem 10. For the second part, we prove that the problem whether S has a T_i-controlled uniformizer for an i reduces to the question whether $T_i(S)$ has a subset uniformizer for a suitable $T_i(S)$ as defined below in Lemma 24.

Reduction. The next lemma shows that from S a regular $T_i(S)$ can be obtained such that $T_i(S)$ consists of all T_i-controlled synchronizations w with $[\![w]\!] \in [\![S]\!]$.

▶ **Lemma 24.** *For $i \geq 0$, the language $T_i(S) = \{w \in \Sigma_{\mathtt{io}}^* \mid w \in T_i \text{ and } [\![w]\!] \in [\![S]\!]\}$ is a T_i-controlled effectively constructible regular language.*

We are ready to prove the main theorem of this paper.

Proof sketch of Theorem 10. By Lemma 23 we know that if S has a T-controlled uniformizer, then S has a T_k-controlled uniformizer for a computable $k \geq 0$. Let $T_k(S)$ be defined as in Lemma 24.

We can show that S has a T-controlled uniformizer iff $\mathrm{dom}([\![S]\!]) = \mathrm{dom}([\![T_k(S)]\!])$ and $T_k(S)$ has a subset uniformizer which is decidable by Theorem 8. ◀

5 Conclusion

In this paper we considered uniformization by subsequential transducers in which the allowed input/output behavior is specified by a regular set of synchronizations, the so-called resynchronized uniformization problem. An overview over our results can be found in Table 1.

For future work we want to study other problems of this kind, e.g., study whether the resynchronized uniformization problem is decidable for a given rational relation as source language and a given 'recognizable' target language in the sense that the target language is controlled by a synchronization language that synchronizes recognizable relations.

References

1 Jean Berstel. Transductions and context-free languages `http://www-igm.univ-mlv.fr/~berstel/`, 2009. URL: `http://www-igm.univ-mlv.fr/~berstel/`.

2 J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969. `doi:10.1090/S0002-9947-1969-0280205-0`.

3 Arnaud Carayol and Christof Löding. Uniformization in Automata Theory. In *Proceedings of the 14th Congress of Logic, Methodology and Philosophy of Science Nancy, July 19-26, 2011*, pages 153–178. London: College Publications, 2014.

4 Alonzo Church. Logic, arithmetic and automata. In *Proceedings of the International Congress of Mathematicians*, pages 23–35, 1962.

5 R. Diestel. *Graph Theory, 2nd Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2000.

6 Diego Figueira and Leonid Libkin. Synchronizing relations on words. *Theory Comput. Syst.*, 57(2):287–318, 2015.

7 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 125:1–125:14, 2016. `doi:10.4230/LIPIcs.ICALP.2016.125`.

8 Christiane Frougny and Jacques Sakarovitch. Synchronized rational relations of finite and infinite words. *Theor. Comput. Sci.*, 108(1):45–82, 1993. `doi:10.1016/0304-3975(93)90230-Q`.

9 Michael Holtmann, Łukasz Kaiser, and Wolfgang Thomas. Degrees of lookahead in regular infinite games. In *Foundations of Software Science and Computational Structures*, volume 6014 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 2010. `doi:/10.1007/978-3-642-12032-9_18`.

10 Frederick A. Hosch and Lawrence H. Landweber. Finite delay solutions for sequential conditions. In *ICALP*, pages 45–60, 1972.

11 J. Howard Johnson. Uniformizing rational relations for natural language applications using weighted determinization. In *Proceedings of the 15th International Conference on Implementation and Application of Automata*, CIAA'10, pages 173–180, Berlin, Heidelberg, 2011. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=1964285.1964304`.

12 M. Nivat. Transductions des langages de Chomsky. *Ann. de l'Inst. Fourier*, 18:339–456, 1968. in french.

13 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

14 Wolfgang Thomas. Church's problem and a tour through automata theory. In *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *Lecture Notes in Computer Science*, pages 635–655. Springer, 2008.

Congestion-Free Rerouting of Flows on DAGs

Saeed Akhoondian Amiri
Max-Planck Institute of Informatics, Germany
samiri@mpi-inf.mpg.de

Szymon Dudycz
University of Wroclaw, Poland
szymon.dudycz@gmail.com

Stefan Schmid
University of Vienna, Austria
stefan_schmid@univie.ac.at

Sebastian Wiederrecht
TU Berlin, Germany
sebastian.wiederrecht@tu-berlin.de

Abstract

Changing a given configuration in a graph into another one is known as a reconfiguration problem. Such problems have recently received much interest in the context of algorithmic graph theory. We initiate the theoretical study of the following reconfiguration problem: How to reroute k unsplittable flows of a certain demand in a capacitated network from their current paths to their respective new paths, in a congestion-free manner? This problem finds immediate applications, e.g., in traffic engineering in computer networks. We show that the problem is generally NP-hard already for $k = 2$ flows, which motivates us to study rerouting on a most basic class of flow graphs, namely DAGs. Interestingly, we find that for general k, deciding whether an unsplittable multi-commodity flow rerouting schedule exists, is NP-hard even on DAGs. Our main contribution is a polynomial-time (fixed parameter tractable) algorithm to solve the route update problem for a bounded number of flows on DAGs. At the heart of our algorithm lies a novel decomposition of the flow network that allows us to express and resolve reconfiguration dependencies among flows.

2012 ACM Subject Classification Networks → Network algorithms, Theory of computation → Network flows

Keywords and phrases Unsplittable Flows, Reconfiguration, DAGs, FPT, NP-Hardness

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.143

Funding The research of Saeed Amiri and Sebastian Wiederrecht was partly supported by the ERC consolidator grant DISTRUCT, agreement No 648527. Stefan Schmid was supported by the Danish VILLUM foundation project *ReNet*.

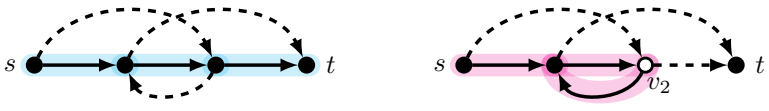
Acknowledgements We would like to thank Stephan Kreutzer, Arne Ludwig and Roman Rabinovich for discussions on this problem.

1 Introduction

Reconfiguration problems are combinatorial problems which ask for a transformation of one configuration into another one, subject to some (reconfiguration) rules. Reconfiguration problems are fundamental and have been studied in many contexts, including puzzles and games (such as Rubik's cube) [24], satisfiability [15], independent sets [16], vertex coloring [9], or matroid bases [17], to just name a few.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 143; pp. 143:1–143:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



initial network invalid update: no transient flow (loop!)

Figure 1 *Example:* We are given an initial network consisting of exactly one active flow F^o (solid edges) and the inactive edges (i.e., inactive forwarding rules) of the new flow F^u to which we want to reroute (dashed edges). Together we call the two flows an (update) pair $P = (F^o, F^u)$. Updating the outgoing edges of a vertex means activating all previously inactive outgoing edges of F^u, and deactivating all other edges of the old flow F^o. Initially, the blue flow is a valid (transient) (s,t)-flow. If the update of vertex v_2 takes effect first, an invalid (not transient) flow is introduced (in pink): traffic is forwarded in a loop, hence (temporarily) invalidating the path from s to t.

Reconfiguration problems also naturally arise in the context of networking applications and routing. For example, a fundamental problem in computer networking regards the question of how to reroute traffic from the current path p_1 to a given new path p_2, by changing the forwarding rules at routers (the *vertices*) one-by-one, while maintaining certain properties *during* the reconfiguration (e.g., short path lengths [7]). Route reconfigurations (or *updates*) are frequent in computer networks: paths are changed, e.g., to account for changes in the security policies, in response to new route advertisements, during maintenance (e.g., replacing a router), to support the migration of virtual machines, etc. [13].

This paper initiates the study of a basic *multi-commodity flow rerouting problem*: how to reroute a set of *unsplittable flows* (with certain bandwidth demands) in a capacitated network, from their current paths to their respective new paths *in a congestion-free manner*. The problem finds immediate applications in traffic engineering [4], whose main objective is to avoid network congestion. Interestingly, while congestion-aware routing and traffic engineering problems have been studied intensively in the past [1, 10, 11, 12, 18, 19, 20, 22], surprisingly little is known today about the problem of how to reconfigure resp. *update* the routes of flows. Only recently, due to the advent of Software-Defined Networks (SDNs), the problem has received much attention in the networking community [3, 8, 14, 21].

Figure 1 presents a simple example of the consistent rerouting problem considered in this paper, for just a *single* flow: the flow needs to be rerouted from the solid path to the dashed path, by changing the forwarding links at routers one-by-one. The example illustrates a problem that might arise from updating the vertices in an invalid order: if vertex v_2 is updated first, a forwarding loop is introduced: the transient flow from s to t becomes invalid. Thus, router updates need to be scheduled intelligently over time: A feasible sequence of updates for this example is given in Figure 2. Note that the example is kept simple intentionally: when moving from a single flow to multiple flows, additional challenges are introduced, as the flows may compete for bandwidth and hence interfere.

Contributions. This paper initiates the algorithmic study of a fundamental unsplittable multicommodity flow rerouting problem. We present a rigorous formal model and show that the problem of rerouting flows in a congestion-free manner is NP-hard already for two flows on general graphs. This motivates us to focus on a most fundamental type of flow graphs, namely the DAG. The main results presented in this paper are the following:

1. Deciding whether a consistent network update schedule exists in general graphs is NP-hard, already for 2 flows.
2. For constant k, we present a linear-time (fixed parameter tractable) algorithm which finds a feasible update schedule on DAGs in time and space $2^{O(k \log k)}O(|G|)$, whenever such a consistent update schedule exists.

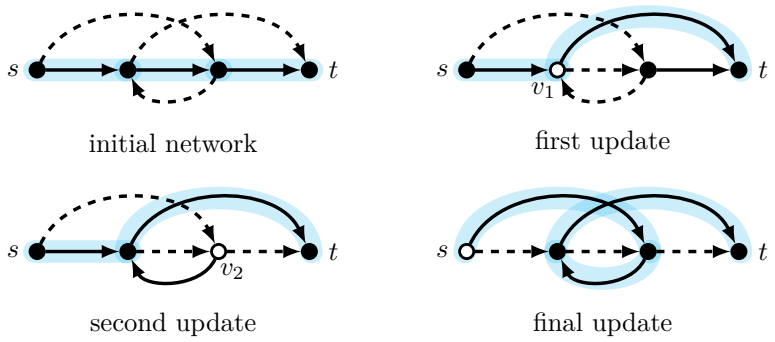


Figure 2 *Example:* We revisit the network of Figure 1 and reroute from F^o to F^u without interrupting the connection between s and t along a unique (transient) path (in blue). To avoid the problem seen in Figure 1, we first update the vertex v_1 in order to establish a shorter connection from s to t. Once this update has been performed, the update of v_2 can be performed without creating a loop. Finally, by updating s, we complete the rerouting.

3. For general k, deciding whether a feasible schedule exists is NP-hard even on loop-free networks (i.e., DAGs).

Against the backdrop that the problem of *routing* disjoint paths on DAGs is known to be $W[1]$-hard [23] and computing routes *subject to congestion* even harder [1], our finding that the multicommodity flow *rerouting* problem is fixed parameter tractable on DAGs is intriguing.

Technical Novelty. Our algorithm is based on a novel decomposition of the flow graph into so-called *blocks*. This block decomposition allows us to express dependencies between flows, and we represent dependencies between blocks by a (directed) dependency graph D. The structure of D is sophisticated, hence to analyze it, we first construct a helper graph H. In our first main technical lemma, we show that if there is an independent set I in H, then the dependency graph that corresponds to the vertices of I is a DAG (Lemma 11). So we may concentrate on a subgraph of D with a simpler structure, which we use to prove the next main technical lemma: there is a congestion-free rerouting if and only if the maximum independent set in H is large enough (Lemma 14). We are left with the challenge that finding a maximum independent set is a hard problem, even in our very restricted graph classes. We hence carefully modify H to obtain a much simpler graph of *bounded pathwidth*, without losing any critical properties. Thanks to these lemmas, the proof of the main theorem will follow.

In addition to our algorithmic contributions, we present NP-hardness proofs. These hardness proofs are based on novel and non-trivial insights into the flow rerouting problem, which might be helpful for similar problems in the future.

2 Model and Definitions

The problem can be described in terms of edge capacitated directed graphs. In what follows, we will assume basic familiarity with directed graphs and we refer the reader to [5] for more background. We denote a directed edge e with head v and tail u by $e = (u, v)$. For an undirected edge e between vertices u, v, we write $e = \{u, v\}$; u, v are called endpoints of e.

A **flow network** is a directed capacitated graph $G = (V, E, s, t, c)$, where s is the *source*, t the *terminal*, V is the set of vertices with $s, t \in V$, $E \subseteq V \times V$ is a set of ordered pairs known as edges, and $c\colon E \to \mathbb{N}$ a capacity function assigning a capacity $c(e)$ to every edge $e \in E$.

Our problem, as described above is a multi-commodity flow problem and thus may have *multiple* source-terminal pairs. To simplify the notation but without loss of generality, in what follows, we define flow networks to have exactly one source and one terminal. In fact, we can model any number of different sources and terminals by adding one super source with edges of unlimited capacity to all original sources, and one super terminal with edges of unlimited capacity leading there from all original terminals.

An (s,t)-*flow* F of capacity $d \in \mathbb{N}$ is a *directed path* from s to t in a flow network such that $d \leq c(e)$ for all $e \in E(F)$. Given a family $\mathcal{F}$ of (s,t)-flows $F_1, \ldots, F_k$ with demands $d_1, \ldots, d_k$ respectively, we call $\mathcal{F}$ a **valid flow set**, or simply **valid**, if $c(e) \geq \sum_{i\colon e \in E(F_i)} d_i$.

Recall that we consider the problem of how to reroute a current (old) flow to a new (update) flow, and hence we will consider such flows in "update pairs": An **update flow pair** $P = (F^o, F^u)$ consists of two (s,t)-flows F^o, the *old flow*, and F^u, the *update flow*, each of demand d. A graph $G = (V, E, \mathcal{P}, s, t, c)$, where (V, E, s, t, c) is a flow network, and $\mathcal{P} = \{P_1, \ldots, P_k\}$ with $P_i = (F_i^o, F_i^u)$, a family of update flow pairs of demand d_i, $V = \bigcup_{i\in[k]} V(F_i^o \cup F_i^u)$ and $E = \bigcup_{i\in[k]} E(F_i^o \cup F_i^u)$, is called **update flow network** if the two families $\mathcal{P}^o = \{F_1^o, \ldots, F_k^o\}$ and $\mathcal{P}^u = \{F_1^u, \ldots, F_k^u\}$ are valid. For an illustration, recall the initial network in Figure 2: The old flow is presented as the directed path made of solid edges and the new one is represented by the dashed edges.

Given an update flow network $G = (V, E, \mathcal{P}, s, t, c)$, an **update** is a pair $\mu = (v, P) \in V \times \mathcal{P}$. An update (v, P) with $P = (F^o, F^u)$ is *resolved* by deactivating all outgoing edges of F^o incident to v and activating all of its outgoing edges of F^u. Note that at all times, there is at most one outgoing and at most one incoming edge, for any flow at a given vertex. So the deactivated edges of F^o can no longer be used by the flow pair P (but now the newly activated edges of F^u can).

For any set of updates $U \subset V \times \mathcal{P}$ and any flow pair $P = (F^o, F^u) \in \mathcal{P}$, $G(P, U)$ is the update flow network consisting exactly of the vertices $V(F^o) \cup V(F^u)$ and the edges of P that are active after resolving all updates in U.

As an illustration, after the second update in Figure 2, one of the original solid edges is still not deactivated. However, already two of the new edges have become solid (i.e., active). So in the picture of the second update, the set $U = \{(v_1, P), (v_2, P)\}$ has been resolved.

We are now able to determine, for a given set of updates, which edges we can and which edges we cannot use for our routing. In the end, we want to describe a process of reconfiguration steps, starting from the *initial state*, in which no update has been resolved, and finishing in a state where the only active edges are exactly those of the new flows, of every update flow pair.

The flow pair P is called **transient** for some set of updates $U \subseteq V \times \mathcal{P}$, if $G(P, U)$ contains a unique valid (s,t)-flow $T_{P,U}$. If there is a family $\mathcal{P} = \{P_1, \ldots P_k\}$ of update flow pairs with demands $d_1, \ldots, d_k$ respectively, we call $\mathcal{P}$ a **transient family** for a set of updates $U \subseteq V \times \mathcal{P}$, if and only if every $P \in \mathcal{P}$ is transient for U. The family of transient flows after all updates in U are resolved is denoted by $\mathcal{T}_{\mathcal{P},\mathcal{U}} = \{T_{P_1,U}, \ldots, T_{P_k,U}\}$.

We again refer to Figure 2. In each of the different states, the transient flow is depicted as the light blue line connecting s to t and covering only solid (i.e., active) edges.

An **update sequence** $(\sigma_i)_{i\in[|V\times\mathcal{P}|]}$ is an ordering of $V \times \mathcal{P}$. We denote the set of updates that is resolved after step i by $U_i = \bigcup_{j=1}^{i} \sigma_i$, for all $i \in [|V \times \mathcal{P}|]$.

▶ **Definition 1** (Consistency Rule)**.** Let σ be an update sequence. We require that for any $i \in [|V \times \mathcal{P}|]$, there is a family of transient flow pairs $\mathcal{T}_{\mathcal{P},\mathcal{U}_i}$.

To ease the notation, we will denote an update sequence $(\sigma)_{i\in[|V\times\mathcal{P}|]}$ simply by σ and for any update (u, P) we write $\sigma(u, P)$ for the the position i of (u, P) within σ. An update sequence is **valid**, if every set U_i, $i \in [|V \times \mathcal{P}|]$, obeys the consistency rule.

We note that this consistency rule models and consolidates the fundamental properties usually studied in the literature, such as congestion-freedom [8] and loop-freedom [21].

▶ **Definition 2** (k-Network Flow Update Problem)**.** Given an update flow network G with k update flow pairs, is there a feasible update sequence σ?

3 On Hardness of 2-Flow Update in General Graphs

It is easy to see that for an update flow network with a single flow pair, feasibility is always guaranteed. However, it turns out that for two flows, the problem becomes hard in general.

▶ **Theorem 3.** *Deciding whether a feasible network update schedule exists is NP-hard already for $k = 2$ flows.*

The proof, briefly sketched in the following, is by reduction from 3-SAT. Let C be any 3-SAT formula with n variables and m clauses. Denote the variables by $X_1, \ldots, X_n$ and the clauses by $C_1, \ldots, C_m$. The resulting update flow network is denoted by $G(C)$. Assume that the variables are ordered by their indices, and their appearance in each clause respects this order.

We create 2 update flow pairs, a blue one $B = (B^o, B^u)$ and a red one $R = (R^o, R^u)$, both of demand 1. The pair B contains gadgets corresponding to the variables. The order in which the edges of each of those gadgets are updated will correspond to assigning a value to the variable. The pair R on the other hand contains gadgets representing the clauses: they have edges that are "blocked" by the variable edges of B. Therefore, we will need to update B to enable the updates of R.

4 Rerouting Flows in DAGs

We now consider the flow rerouting problem when the underlying flow graph is acyclic. In particular, we identify an important substructure arising for flow-pairs in acyclic graphs, which we call *blocks*. These blocks will play a major role in both the hardness proof and the algorithm presented in this section.

Let $G = (V, E, \mathcal{P}, s, t, c)$ be an acyclic update flow network, i.e., we assume that the graph (V, E) is a DAG. Let $\prec$ be a topological order on the vertices $V = \{v_1, \ldots, v_n\}$. Let $P_i = (F_i^o, F_i^u)$ be an update flow pair of demand d and let $v_1^i, \ldots, v_{\ell_i^o}^i$ be the induced topological order on the vertices of F_i^o; analogously, let $u_1^i, \ldots, v_{\ell_i^u}^i$ be the order on F_i^u. Furthermore, let $V(F_i^o) \cap V(F_i^u) = \{z_1^i, \ldots, z_{k_i}^i\}$ be ordered by $\prec$ as well.

The subgraph of $F_i^o \cup F_i^u$ induced by the set $\{v \in V(F_i^o \cup F_i^u) \mid z_j^i \prec v \prec z_{j+1}^i\}$, $j \in [k_i - 1]$, is called the jth *block* of the update flow pair F_i, or simply the jth *i-block*. We will denote this block by b_j^i.

For a block b, we define $\mathscr{S}(b)$ to be the *start of the block*, i.e., the smallest vertex w.r.t. $\prec$; similarly, $\mathscr{E}(b)$ is the *end of the block*: the largest vertex w.r.t. $\prec$.

Let $G = (V, E, \mathcal{P}, s, t, c)$ be an update flow network with $\mathcal{P} = \{P_1, \ldots, P_k\}$ and let $\mathcal{B}$ be the set of its blocks. We define a binary relation $<$ between two blocks as follows. For two

blocks $b_1, b_2 \in \mathcal{B}$, where b_1 is an i-block and b_2 a j-block, $i, j \in [k]$, we say $b_1 < b_2$ (b_1 *is smaller than* b_2) if one of the following holds.

i $\mathscr{S}(b_1) \prec \mathscr{S}(b_2)$,
ii if $\mathscr{S}(b_1) = \mathscr{S}(b_2)$ then $b_1 < b_2$, if $\mathscr{E}(b_1) \prec \mathscr{E}(b_2)$,
iii if $\mathscr{S}(b_1) = \mathscr{S}(b_2)$ and $\mathscr{E}(b_1) = \mathscr{E}(b_2)$ then $b_1 < b_2$, if $i < j$.

Let b be an i-block and P_i the corresponding update flow pair. For a feasible update sequence σ, we will denote the round $\sigma(\mathscr{S}(b), P_i)$ by $\sigma(b)$. We say that an i-block b is *updated*, if all edges in $b \cap F_i^u$ are active and all edges in $b \cap F_i^o \setminus F_i^u$ are inactive. We will make use of a basic, but important observation on the structure of blocks and how they can be updated. This structure is the key to our flow reconfiguration algorithm (presented below), as it allows us to consider the update of blocks as a whole, rather than vertex-by-vertex.

▶ **Lemma 4.** *Let b be a block of the flow pair $P = (F^u, F^o)$. Then in a feasible update sequence σ, all vertices (resp. their outgoing edges belonging to P) in $F^u \cap b - \mathscr{S}(b)$ are updated strictly before $\mathscr{S}(b)$. Moreover, all vertices in $b - F^u$ are updated strictly after $\mathscr{S}(b)$ is updated.*

▶ **Lemma 5.** *Let G be an update flow network and σ a valid update sequence for G. Then there exists a feasible update sequence σ' which updates every block in consecutive rounds.*

Recall that G is acyclic and every flow pair in G forms a single block. Let σ be a feasible update sequence of G. We suppose in σ, every block is updated in consecutive rounds (Lemma 5). For a single flow F, we write $\sigma(F)$ for the round where the last edge of F was updated.

4.1 Updating k-Flows in DAGs is NP-complete

We first show that if k is part of the input, the congestion-free flow reconfiguration problem is even hard on the DAG. Hence the algorithm presented in the following is essentially tight. To prove the theorem, we use a polynomial time reduction from the 3-SAT problem.

▶ **Theorem 6.** *Finding a feasible update sequence for k-flows is NP-complete, even if the update graph G is acyclic.*

4.2 Linear Time Algorithm for Constant Number of Flows on DAGs

By Theorem 6 we cannot hope to find a polynomial time algorithm that finds a feasible update sequence. However, if the problem is parameterized by the number k of flows, a rerouting sequence can be computed in FPT-linear time if the update graph is acyclic. In this subsection we describe an algorithm to solve the network update problem on DAGs in time $2^{O(k \log k)}O(|G|)$, for arbitrary k. In the remainder of this section, we assume that every block has at least 3 vertices (otherwise, postponing such block updates will not affect the solution).

We say a block b_1 *touches* a block b_2 (denoted by $b_1 \succ b_2$) if there is a vertex $v \in b_1$ such that $\mathscr{S}(b_2) \prec v \prec \mathscr{E}(b_2)$, or there is a vertex $u \in b_2$ such that $\mathscr{S}(b_1) \prec u \prec \mathscr{E}(b_1)$. If b_1 does not touch b_2, we write $b_1 \not\succ b_2$. Clearly, the relation is symmetric, i.e., if $b_1 \succ b_2$ then $b_2 \succ b_1$.

For some intuition, consider a drawing of G which orders vertices w.r.t. $\prec$ in a line. Project every edge on that line as well. Then two blocks touch each other if they have a common segment on that projection.

Proof Sketch: Before delving into details, we provide the main ideas behind our algorithm. We can think about the update problem on DAGs as follows. Our goal is to compute a feasible update order for the (out-)edges of the graph. There are at most k flows to be updated for each edge, resulting in $k!$ possible orders and hence a brute force complexity of $O(k!^{|G|})$ for the entire problem. We can reduce this complexity by considering blocks instead of edges.

The update of a given i-block b_i might depend on the update of a j-block sharing at least one edge of b_i. These dependencies can be represented as a directed graph. If this graph does not have any directed cycles, it is rather easy to find a feasible update sequence, by iteratively updating sink vertices.

There are several issues here: First of all these dependencies are not straight-forward to define. As we will see later, they may lead to representation graphs of exponential size. In order to control the size we might have to relax our definition of dependency, but this might lead to a not necessarily acyclic graph which will then need further refinement. This refinement is realized by finding a suitable subgraph, which alone is a hard problem in general. To overcome the above problems, we proceed as follows.

Let TouchSeq(b) contain all feasible update sequences for the blocks that touch b: still a (too) large number, but let us consider them for now. For two distinct blocks b, b', we say that two sequences $s \in \text{TouchSeq}(b), s' \in \text{TouchSeq}(b')$ are *consistent*, if the order of any common pair of blocks is the same in both s, s'. If for some block b, $\text{TouchSeq}(b) = \emptyset$, there is no feasible update sequence for G: b cannot be updated.

We now consider a graph H whose vertices correspond to elements of TouchSeq(b), for all $b \in \mathcal{B}$. Connect all pairs of vertices originating from the same TouchSeq(b). Connect all pairs of vertices if they correspond to inconsistent elements of different TouchSeq(b). If (and only if) we find an independent set of size $|\mathcal{B}|$ in the resulting graph, the update orders corresponding to those vertices are mutually consistent: we can update the entire network according to those orders. In other words, the update problem can be reduced to finding an independent set in the graph H.

However, there are two main issues with this approach. First, H can be very large. A single TouchSeq(b) can have exponentially many elements. Accordingly, we observe that we can assume a slightly different perspective on our problem: we linearize the lists TouchSeq(b) and define them sequentially, bounding their size by a function of k (the number of flows). The second issue is that finding a maximum independent set in H is hard. The problem is equivalent to finding a clique in the complement of H, a $|\mathcal{B}|$-partite graph where every partition has bounded cardinality. We can prove that for an n-partite graph where every partition has bounded cardinality, finding an n-clique is NP-complete. So, in order to solve the problem, we either should reduce the number of partitions in H (but we cannot) or modify H to some other graph, further reducing the complexity of the problem. We do the latter by trimming H and removing some extra edges, turning the graph into a very simple one: a graph of *bounded path width*. Then, by standard dynamic programming, we find the independent set of size $|\mathcal{B}|$ in the trimmed version of H: this independent set matches the independent set I of size $|\mathcal{B}|$ in H (if it exists). At the end, reconstructing a correct update order sequence from I needs some effort. As we have reduced the size of TouchSeq(b) and while not all possible update orders of all blocks occur, we show that they suffice to cover all possible feasible solutions. We provide a way to construct a valid update order accordingly.

With these intuitions in mind, we now present a rigorous analysis. Let $\pi_{S_1} = (a_1, \ldots, a_{\ell_1})$ and $\pi_{S_2} = (a'_1, \ldots, a'_{\ell_2})$ be permutations of sets S_1 and S_2. We define the *core* of π_{S_1} and π_{S_2} as $core(\pi_{S_1}, \pi_{S_2}) := S_1 \cap S_2$. We say that two permutations π_1 and π_2 are *consistent*, $\pi_1 \approx \pi_2$, if there is a permutation π of symbols of $core(\pi_1, \pi_2)$ such that π is a subsequence of both π_1 and π_2.

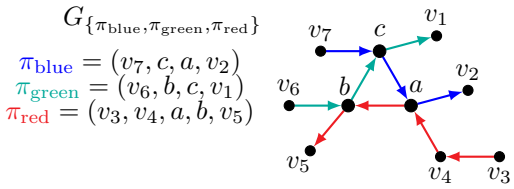


Figure 3 *Example:* The dependency graph of three pairwise consistent permutations π_{blue}, π_{green} and π_{red}. Each pair of those permutation has exactly one vertex in common and with this the cycle (a, b, c) is created. With such cycles being possible, a dependency graph does not necessarily contain sink vertices. To get rid of them, we certainly need some more refinements.

The **dependency graph** is a labelled graph defined recursively as follows. The dependency graph of a single permutation $\pi = (a_1, \ldots, a_\ell)$, denoted by G_π, is a directed path $v_1, \ldots, v_\ell$, and the label of the vertex $v_i \in V(G_\pi)$ is the element a with $\pi(a) = i$. We denote by $\text{Labels}(G_\pi)$ the set of all labels of G_π.

Let G_Π be a dependency graph of the set of permutations Π and $G_{\Pi'}$ the dependency graph of the set Π'. Then, their union (by identifying the same vertices) forms the dependency graph $G_{\Pi \cup \Pi'}$ of the set $\Pi \cup \Pi'$. Note that such a dependency graph is not necessarily acyclic (see Figure 3).

We call a permutation π of blocks of a subset $\mathcal{B}' \subseteq \mathcal{B}$ *congestion free*, if the following holds: it is possible to update the blocks in π in the graph $G_{\mathcal{B}}$ (the graph on the union of blocks in $\mathcal{B}$), in order of their appearance in π, without violating any edge capacities in $G_{\mathcal{B}}$. Note that we do not respect all conditions of our *Consistency Rule* (Definition 1) here.

In the approach we are taking, one of the main advantages we have is the nice properties of blocks when it comes to updating. The following algorithm formalizes the procedure already described in Lemma 5. The correctness follows directly from said lemma. Let $P = (F^o, F^u)$ be a given flow pair.

Algorithm 1. **Update a Free Block b**

1. Resolve (v, P) for all $v \in F^u \cap b - \mathscr{S}(b)$.
2. Resolve $(\mathscr{S}(b), P)$.
3. Resolve (v, P) for all $v \in (b - F^u)$.
4. For any edge in $E(b \cap F^u)$ check whether d_{F^u} together with the other loads currently active on e exceed $c(e)$. If so output: *Fail*.

▶ **Lemma 7.** *Let π be a permutation of the set $\mathcal{B}_1 \subseteq \mathcal{B}$. Whether π is congestion free can be determined in time $O(k \cdot |G|)$.*

The smaller relation defines a total order on all blocks in G. Let $\mathcal{B} = \{b_1, \ldots, b_{|\mathcal{B}|}\}$ and suppose the order is $b_1 < \ldots < b_{|\mathcal{B}|}$.

We define an auxiliary graph H which will help us find a suitable dependency graph for our network. We first provide some high-level definitions relevant to the construction of the graph H only. Exact definitions will follow in the construction of H, and will be used throughout the rest of this section.

Recall that $\mathcal{B}$ is the set of all blocks in G. We define another set of blocks $\mathcal{B}'$ and initialize it as $\mathcal{B}$; the construction of H is iterative, and in each iteration, we eliminate a block from $\mathcal{B}'$. At the end of the construction of H, $\mathcal{B}'$ is empty. For every block $b \in \mathcal{B}'$, we also define the set $\text{TouchingBlocks}(b)$ of blocks which touch the block b, note that this

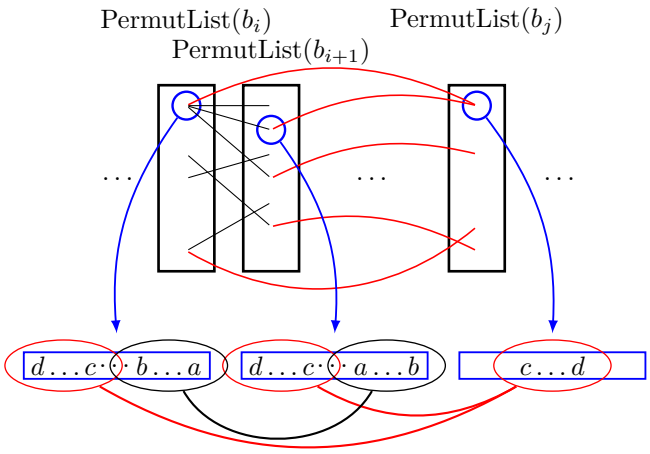


Figure 4 *Example:* The graph H consists of vertex sets PermutList(b_i), $i \in [|\mathcal{B}|]$, where each such partition contains all congestion free sequences of the at most k iteratively chosen touching blocks. In the whole graph, we then create edges between the vertices of two such partitions if and only if the corresponding sequences are inconsistent with each other, as seen in the three highlighted sequences. Later we will distinguish between such edges connecting vertices of neighbouring partitions (w.r.t. the topological order of their corresponding blocks), PermutList(b_i) and PermutList(b_{i+1}), and partitions that are further away, PermutList(b_i) and PermutList(b_j). Edges of the latter type, depicted as red in the figure, are called long edges and will be deleted in the trimming process of H.

set is dynamically defined: it depends on $\mathcal{B}'$. Another set which is defined for every block b is the set PermutList(b); this set actually corresponds to a set of vertices, each of which corresponds to a valid congestion free permutation of blocks in TouchingBlocks(b). Clearly if TouchingBlocks(b) does not contain any congestion-free permutation, then PermutList(b) is an empty set. As we already mentioned, every vertex $v \in$ PermutList(b) comes with a **label** which corresponds to some congestion-free permutation of elements of TouchingBlocks(b). We denote that permutation by Label (v).

Construction of H: We recursively construct a labelled graph H from the blocks of G as follows.

i Set $H \coloneqq \emptyset$, $\mathcal{B}' \coloneqq \mathcal{B}$, PermutList $\coloneqq \emptyset$.
ii For $i \coloneqq 1, \dots, |\mathcal{B}|$ do

1 Let $b \coloneqq b_{|\mathcal{B}|-i+1}$.
2 Let TouchingBlocks(b) $\coloneqq \{b'_1, \dots, b'_t\}$ be the set of blocks in $\mathcal{B}'$ touched by b.
3 Let $\pi \coloneqq \{\pi_1, \dots, \pi_\ell\}$ be the set of congestion free permutations of TouchingBlocks(b).
4 Set PermutList(b) $\coloneqq \emptyset$.
5 For $i \in [\ell]$ create a vertex v_{π_i} with Label(v_{π_i}) $= \pi_i$ and set PermutList(b) $\coloneqq$ PermutList(b) $\cup\, v_{\pi_i}$.
6 Set $H \coloneqq H \cup$ PermutList(b).
7 Add edges between all pairs of vertices in H[PermutList(b)].
8 Add an edge between every pair of vertices $v \in H$[PermutList(b)] and $u \in V(H) -$ PermutList(b) if the labels of v and u are inconsistent.
9 Set $\mathcal{B}' \coloneqq \mathcal{B}' - b$.

We have the following lemmas based on our construction.

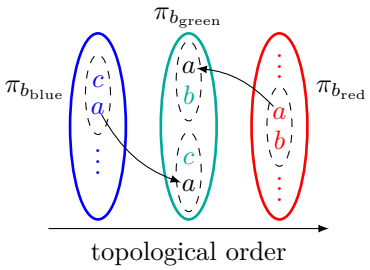


Figure 5 *Example:* Select one of the permutations of length at most k from every PermutList(b). These permutations obey the Touching Lemma. Taking the three permutations from the example in Figure 3, we can see that the Touching Lemma forces a to be in the green permutation as well. Assuming consistency, this would mean a to come *before* b and *after* c. Hence $a <_{\pi_{\text{green}}} b$ and $b <_{\pi_{\text{green}}} a$, a contradiction. So if our permutations are derived from H and are consistent, we will show that cycles cannot occur in their dependency graph.

▶ **Lemma 8.** *For Item (ii2) of the construction of H, $t \leq k$ holds.*

▶ **Lemma 9** (Touching Lemma)**.** *Let $b_{j_1}, b_{j_2}, b_{j_3}$ be three blocks (w.r.t. $<$) where $j_1 < j_2 < j_3$. Let b_z be another block such that $z \notin \{j_1, j_2, j_3\}$. If in the process of constructing H, b_z is in the touch list of both b_{j_1} and b_{j_3}, then it is also in the touch list of b_{j_2}.*

For an illustration of the property described in the Touching Lemma, see Figure 5: it refers to the dependency graph of Figure 3. This example also points out the problem with directed cycles in the dependency graph and the property of the Touching Lemma, which is crucial for Observation 10 and Lemma 11.

We prove a series of lemmas in regard to the dependency graph of elements of H, to establish the base of the inductive proof for Lemma 13.

▶ **Observation 10.** *Let π be a permutation of a set S. Then the dependency graph G_π does not contain a cycle.*

▶ **Lemma 11.** *Let π_1, π_2 be permutations of sets S_1, S_2 such that π_1, π_2 are consistent. Then the dependency graph $G_{\pi_1 \cup \pi_2}$ is acyclic.*

In the next lemma, we need a closure of the dependency graph of permutations which we define as follows.

▶ **Definition 12** (Permutation Graph Closure)**.** The *permutation graph closure*, or simply *closure*, of a permutation π is the graph G_π^+ obtained from taking the transitive closure of G_π, i.e. its vertices and labels are the same as G_π and there is an edge (u, v) in G_π^+ if there is a path starting at u and ending at v in G_π. Similarly the *permutation graph closure* of a set of permutations $\Pi = \{\pi_1, \ldots, \pi_n\}$ is the graph obtained by taking the union of $G_{\pi_i}^+$'s (for $i \in [n]$) by identifying vertices of the same label.

In the above definition, note that if Π is a set of permutations, then $G_\Pi \subseteq G_\Pi^+$. The following lemma generalizes Lemma 11 and Observation 10 and uses them as the base of its inductive proof.

▶ **Lemma 13.** *Let $I = \{v_{\pi_1}, \ldots, v_{\pi_\ell}\}$ be an independent set in H. Then the dependency graph G_Π, for $\Pi = \{\pi_1, \ldots, \pi_\ell\}$, is acyclic.*

Proof. Instead of working on G_Π, we can work on its closure G_Π^+ as defined above. First we observe that every edge in G_Π also appears in G_Π^+, so if there is a cycle in G_Π, the same cycle exists in G_Π^+.

We prove that there is no cycle in G_Π^+. By Lemma 11 and Observation 10 there is no cycle of length at most 2 in G_Π^+; otherwise there is a cycle in G_Π which consumes at most two consistent permutations.

For the sake of contradiction, suppose G_Π^+ has a cycle and let $C = (a_1, \ldots, a_n) \subseteq G_\Pi^+$ be a shortest cycle in G_Π^+. By Lemma 11 and Observation 10 we know that $n \geq 3$.

In the following, because we work on a cycle C, whenever we write any index i we consider it w.r.t. its cyclic order on C, in fact $i \mod |C| + 1$. So for example, $i = 0$ and $i = n$ are identified as the same indices; similarly for $i = n + 1, i = 1$, etc.

Recall the construction of the dependency graph where every vertex $v \in C$ corresponds to some block b_v. In the remainder of this proof we do not distinguish between the vertex v and the block b_v.

Let π_v be the label of a given vertex $v \in I$. For each edge $e = (a_i, a_{i+1}) \in C$, there is a permutation π_{v_i} such that (a_i, a_{i+1}) is a subsequence of π_{v_i} and additionally the vertex v_i is in the set I. So there is a block b^i such that π_{v_i} is a permutation of the set TouchingBlocks(b^i).

The edge $e = (a_i, a_{i+1})$ is said to *represent* b^i, and we call it the representative of π_{v_i}. For each i we fix one block b^i which is represented by the edge (a_i, a_{i+1}) (note that one edge can represent many blocks, but here we fix one of them). We define the set of those blocks as $B^I = \{b^1, \ldots, b^\ell\}$ and state the following claim.

Claim 1. For every two distinct vertices $a_i, a_j \in C$, either there is no block $b \in B^I$ such that $a_i, a_j \in$ TouchingBlocks(b) or if $a_i, a_j \in$ TouchingBlocks(b) then (a_i, a_j) or (a_j, a_i) is an edge in C. Additionally $\left|B^I\right| = |C|$.

By the above claim we have $\ell = n$. W.l.o.g. suppose $b^1 < b^2 < \ldots < b^n$. There is an $i \in [n]$ such that (a_{i-1}, a_i) represents b^1, we fix this i.

Claim 2. If (a_{i-1}, a_i) represents b^1 then (a_{i-2}, a_{i-1}) represents b^2.

Similarly we can prove the endpoints of the edges, that have a_i as their head, are in b^2.

Claim 3. If (a_{i-1}, a_i) represents b^1 then (a_i, a_{i+1}) represents b^2.

By Claims 2 and 3 we have that both (a_{i-2}, a_{i-1}) and (a_i, a_{i+1}) represent b^2 hence by Claim 1 they are the same edge. Thus there is a cycle on the vertices a_{i-1}, a_i in G_Π^+ and this gives a cycle in G_Π on at most 2 consistent permutations which is a contradiction according to Lemma 11. ◀

The following lemma is the key to establish a link between independent sets in H and feasible update sequences of the corresponding update flow network G.

▶ **Lemma 14.** *There is a feasible sequence of updates for an update network G on k flow pairs, if and only if there is an independent set of size $|\mathcal{B}|$ in H. Additionally if the independent set $I \subseteq V(H)$ of size $|\mathcal{B}|$ together with its vertex labels are given, then there is an algorithm which can compute a feasible sequence of updates for G in $O(k \cdot |G|)$.*

With Lemma 14, the update problem boils down to finding an independent set of size $|\mathcal{B}|$ in H.

Finding an independent set of size $|\mathcal{B}|$ in H is a hard problem already on very restricted class families. Hence, we trim H to avoid the above problem. We will use the special properties of the touching relation of blocks. We say that an edge $e \in E(H)$ is *long*, if one end of e is in PermutList(b_i), and the other in PermutList(b_j) where $j > i+1$. The *length* of e is $j-i$. *Delete* all long edges from H to obtain the graph R_H. We prove the following lemmas.

▶ **Lemma 15.** *There is an algorithm which computes R_H in time $O((k \cdot k!)^2 |G|)$.*

▶ **Lemma 16.** *H has an independent set I of size $|\mathcal{B}|$ if, and only if, I is also an independent set of size $|\mathcal{B}|$ in R_H.*

R_H is a much simpler graph compared to H, which helps us find a large independent set of size $|\mathcal{B}|$ (if exists). We have the following lemma.

▶ **Lemma 17.** *There is an algorithm that finds an independent set I of size exactly $|\mathcal{B}|$ in R_H if such an independent set exists; otherwise it outputs that there is no such an independent set. The running time of this algorithm is $O(|R_H|)$.*

Our main theorem is now a corollary of the previous lemmas and algorithms.

▶ **Theorem 18.** *There is a linear time FPT algorithm for the network update problem on an acyclic update flow network G with k flows (the parameter), which finds a feasible update sequence, if it exists; otherwise it outputs that there is no feasible solution for the given instance. The algorithm runs in time $O(2^{O(k \log k)} |G|)$.*

5 Conclusion

This paper initiated the study of a natural and fundamental reconfiguration problem: the congestion-free rerouting of unsplittable flows. Interestingly, we find that while *computing* disjoint paths on DAGs is $W[1]$-hard [23] and finding routes under congestion as well [1], *reconfiguring* multicommodity flows is fixed parameter tractable on DAGs. However, we also show that the problem is NP-hard for an arbitrary number of flows.

In future work, it will be interesting to chart a more comprehensive landscape of the computational complexity for the network update problem. In particular, it would be interesting to know whether the complexity can be reduced further, e.g., to $2^{O(k)}O(|G|)$. More generally, it will be interesting to study other flow graph families, especially more sparse graphs or graphs of bounded DAG width [2, 6]. Finally, besides feasibility, it remains to study algorithms to efficiently compute *short* schedules.

References

1 Saeed Akhoondian Amiri, Stephan Kreutzer, Dániel Marx, and Roman Rabinovich. Routing with congestion in acyclic digraphs. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS*, pages 7:1–7:11, 2016.

2 Saeed Akhoondian Amiri, Stephan Kreutzer, and Roman Rabinovich. Dag-width is pspace-complete. *Theor. Comput. Sci.*, 655:78–89, 2016.

3 Saeed Akhoondian Amiri, Arne Ludwig, Jan Marcinkowski, and Stefan Schmid. Transiently consistent sdn updates: Being greedy is hard. In *23rd International Colloquium on Structural Information and Communication Complexity, SIROCCO*, 2016.

4 D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. Rsvp-te: Extensions to rsvp for lsp tunnels. In *RFC 3209*, 2001.

5 Jørgen Bang-Jensen and Gregory Gutin. *Digraphs - theory, algorithms and applications*. Springer, 2002.

6 Dietmar Berwanger, Anuj Dawar, Paul Hunter, Stephan Kreutzer, and Jan Obdrzálek. The dag-width of directed graphs. *J. Comb. Theory, Ser. B*, 102(4):900–923, 2012.

7 Paul Bonsma. The complexity of rerouting shortest paths. *Theoretical computer science*, 510:1–12, 2013.

8 Sebastian Brandt, Klaus-Tycho Förster, and Roger Wattenhofer. On Consistent Migration of Flows in SDNs. In *Proc. 36th IEEE International Conference on Computer Communications (INFOCOM)*, 2016.

9 Luis Cereceda, Jan Van Den Heuvel, and Matthew Johnson. Finding paths between 3-colorings. *Journal of graph theory*, 67(1):69–82, 2011.

10 Chandra Chekuri, Alina Ene, and Marcin Pilipczuk. Constant congestion routing of symmetric demands in planar directed graphs. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP*, 2016.

11 Chandra Chekuri, Sreeram Kannan, Adnan Raja, and Pramod Viswanath. Multicommodity flows and cuts in polymatroidal networks. *SIAM J. Comput.*, 44(4):912–943, 2015.

12 Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

13 Klaus-Tycho Foerster, Stefan Schmid, and Stefano Vissicchio. Survey of consistent network updates. In *ArXiv Technical Report*, 2016.

14 Klaus-Tycho Förster, Ratul Mahajan, and Roger Wattenhofer. Consistent Updates in Software Defined Networks: On Dependencies, Loop Freedom, and Blackholes. In *Proc. 15th IFIP Networking*, 2016.

15 Parikshit Gopalan, Phokion G Kolaitis, Elitza Maneva, and Christos H Papadimitriou. The connectivity of boolean satisfiability: computational and structural dichotomies. *SIAM Journal on Computing*, 38(6):2330–2355, 2009.

16 Robert A Hearn and Erik D Demaine. Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1-2):72–96, 2005.

17 Takehiro Ito, Erik Demaine, Nicholas Harvey, Christos Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Algorithms and Computation*, pages 28–39, 2008.

18 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Stephan Kreutzer. An excluded half-integral grid theorem for digraphs and the directed disjoint paths problem. In *Proc. Symposium on Theory of Computing (STOC)*, pages 70–78, 2014.

19 Jon M. Kleinberg. Decision algorithms for unsplittable flow and the half-disjoint paths problem. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 530–539, 1998.

20 Frank Thomson Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

21 Arne Ludwig, Jan Marcinkowski, and Stefan Schmid. Scheduling loop-free network updates: It's good to relax! In *Proc. ACM PODC*, 2015.

22 Martin Skutella. Approximating the single source unsplittable min-cost flow problem. In *Proc. IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2000.

23 Aleksandrs Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM Journal on Discrete Mathematics*, 24(1):146–157, 2010.

24 Jan van den Heuvel. The complexity of change. *Surveys in combinatorics*, 409(2013):127–160, 2013.

Practical and Provably Secure Onion Routing

Megumi Ando
Computer Science Department, Brown University, Providence, RI 02912 USA
mando@cs.brown.edu

Anna Lysyanskaya
Computer Science Department, Brown University, Providence, RI 02912 USA
anna@cs.brown.edu

Eli Upfal
Computer Science Department, Brown University, Providence, RI 02912 USA
eli@cs.brown.edu

Abstract

In an onion routing protocol, messages travel through several intermediaries before arriving at their destinations; they are wrapped in layers of encryption (hence they are called "onions"). The goal is to make it hard to establish who sent the message. It is a practical and widespread tool for creating anonymous channels.

For the standard adversary models – passive and active – we present practical and provably secure onion routing protocols. Akin to Tor, in our protocols each party independently chooses the routing paths for his onions. For security parameter λ, our differentially private solution for the active adversary takes $O(\log^2 \lambda)$ rounds and requires every participant to transmit $O(\log^4 \lambda)$ onions in every round.

2012 ACM Subject Classification Security and privacy → Security protocols

Keywords and phrases Anonymity, traffic analysis, statistical privacy, differential privacy

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.144

Related Version A full version of the paper is available at `https://arxiv.org/abs/1706.05367`.

Funding This work was supported in part by NSF grants IIS-1247581 and CNS-1422361.

Acknowledgements We'd like to thank the reviewers for their thoughtful comments.

1 Introduction

Anonymous channels are a prerequisite for protecting user privacy. But how do we achieve anonymous channels in an Internet-like network that consists of point-to-point links?

If a user Alice wishes to send a message m to a user Bob, she may begin by encrypting her message m under Bob's public key to obtain the ciphertext $c_{Bob} = \mathsf{Enc}(\mathsf{pk}_{Bob}, m)$. But sending c_{Bob} directly to Bob would allow an eavesdropper to observe that Alice is in communication with Bob. So instead, Alice may designate several intermediate relays, called "mix-nodes" (typically chosen at random) and send the ciphertext through them, "wrapped" in several layers of encryption so that the ciphertext received by a mix-node cannot be linked to the ciphertext sent out by the mix-node. Each node decrypts each ciphertext it receives ("peels off" a layer of encryption) and discovers the identity of the next node and the ciphertext to send along. This approach to hiding who is talking to whom is called "onion

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 144; pp. 144:1–144:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

routing" [10] (sometimes it is also called "anonymous remailer" [13]) because the ciphertexts are layered, akin to onions; from now on we will refer to such ciphertexts as "onions".

Onion routing is attractive for several reasons: (1) simplicity: users and developers understand how it works; the only cryptographic tool it uses is encryption; (2) fault-tolerance: it can easily tolerate and adapt to the failure of a subset of mix-nodes; (3) scalability: its performance remains the same even as more and more users and mix-nodes are added to the system. As a result, onion routing is what people use to obscure their online activities. According to current statistics published by the Tor Project, Inc., Tor is used by millions of users every day to add privacy to their communications [14, 15][1].

In spite of its attractiveness and widespread use, the security of onion routing is not well-understood.

The definitional question – what notion of security do we want to achieve? – has been studied [3,20,21,28]. The most desirable notion, which we will refer to as "statistical privacy", requires that the adversary's view in the protocol be distributed statistically independently of who is trying to send messages to whom[2]. Unfortunately, a network adversary observing the traffic flowing out of Alice and flowing into Bob can already make inferences about whether Alice is talking to Bob. For example, if the adversary knows that Alice is sending a movie to someone, but there isn't enough traffic flowing into Bob's computer to suggest that Bob is receiving a movie, then Bob cannot be Alice's interlocutor. (Participants' inputs may also affect others' privacy in other ways [20].)

So let us consider the setting in which, in principle, statistical privacy can be achieved: every party wants to anonymously send and receive just one short message to and from some other party. Let us call this "the simple input-output (I/O) setting". In the simple I/O setting, anonymity can be achieved even against an adversary who can observe the entire network if there is a trusted party through whom all messages are routed. Can onion routing that does not rely on one trusted party emulate such a trusted party in the presence of a powerful adversary?

Specifically, we may be dealing with *the network adversary* that observes all network traffic; or the stronger *passive adversary* that, in addition to observing network traffic, also observes the internal states of a fraction of the network nodes; or the most realistic *active adversary* that observes network traffic and also controls a fraction of the nodes. Prior work analyzing Tor [3, 20, 21] did not consider these standard adversary models. Instead, they focused on the adversary who was entirely absent from some regions of the network, but resourceful adversaries (such as the NSA) and adversaries running sophisticated attacks (such as BGP hijacking [29]) may receive the full view of the network traffic, and may also infiltrate the collection of mix-nodes.

Surprisingly, despite its real-world importance, we were the first to consider this question.

Warm-up: An *oblivious permutation algorithm* between a memory-constrained client and an untrusted storage server enables the client to permute a sequence of (encrypted) data blocks stored on the server without the server learning anything (in the statistical sense) about the permutation.

---

[1] Tor stands for "the onion router", and even though the underlying mechanics are somewhat different from what we described above (instead of using public-key encryption, participants carry out key exchange so that the rest of the communication can be more efficient), the underlying theory is still the same.

[2] Technically, since onion routing uses encryption, the adversary's view cannot be statistically independent of the input, but at best computationally independent. However, as we will see, if we work in an idealized encryption model, such as in Canetti's $\mathcal{F}_{\mathsf{Enc}}$-hybrid model [7], statistical privacy makes sense.

▶ **Theorem 1.** *Any oblivious permutation algorithms can be adapted into a communications protocol for achieving statistical privacy from the network adversary.*

As an example, Ohrimenko et al. [26] presented a family of efficient oblivious permutation algorithms. This can be adapted into a secure and "tunable" OR protocol that can trade off between low server load and latency. Letting λ denote the security parameter, for any $B \in [\frac{\sqrt{N}}{\log^2 \lambda}]$, this protocol can be set to run in $O(\frac{\log N}{\log B})$ rounds with communication complexity overhead $O(\frac{B \log N \log^2 \lambda}{\log B})$ and server load $O(B \log^2 \lambda)$.

However, to be secure from the passive adversary, we need more resources. We prove for the first time that onion routing can provide statistical privacy from the passive adversary, while being efficient.

1. We prove that our solution, Π_p, is statistically private from any passive adversary capable of monitoring any constant $\kappa \in [0, 1)$ of the mix-nodes, while having communication complexity overhead $O(\log^2 \lambda)$, server load $O(\log^2 \lambda)$, and latency $O(\log^2 \lambda)$, where λ denotes the security parameter. (See Section 4.)

However, for most realistic input settings (not constrained to the simple I/O setting), statistical privacy is too ambitious a goal. It is not attainable even with a trusted third party. Following recent literature [3, 31], for our final result, let us *not* restrict users' inputs, and settle for a weaker notion of privacy, namely, differential privacy.

Our definition of differential privacy requires that the difference between the adversary's view when Alice sends a message to Bob and its view when she does not send a message at all or sends it to Carol instead, is small. This is meaningful; showing that the protocol achieves differential privacy gives every user a guarantee that sending her message through does not change the adversary's observations very much.

2. Our solution, Π_a, can defend against the active adversary while having communication complexity overhead $O(\log^6 \lambda)$, server load $O(\log^4 \lambda)$, and latency $O(\log^2 \lambda)$. This is the first provably secure peer-to-peer solution that also provides a level of robustness; unless the adversary forces the honest players to abort the protocol run, most messages that are not dropped by the adversary are delivered to their final destinations. (See Section 5.)

To prepare onions, we use a cryptographic scheme that is strong enough that, effectively, the only thing that the active adversary can do with onions generated by honest parties is to drop them (see the onion cryptosystem by Camenisch and Lysyanskaya [6] for an example of a sufficiently strong cryptosystem). Unfortunately, even with such a scheme, it is still tricky to protect Alice's privacy against an adversary that targets Alice specifically. Suppose that an adversarial Bob is expecting a message of a particular form from an anonymous interlocutor, and wants to figure out if it was Alice or not. If the adversary succeeds in blocking all of Alice's onions and not too many of the onions from other parties, and then Bob never receives the expected message, then the adversary's hunch that it was Alice will be confirmed.

How do we prevent this attack? For this attack to work, the adversary would have to drop a large number of onions – there is enough cover traffic in our protocol that dropping just a few onions does not do much. But once a large enough number of onions is dropped, the honest mix-nodes will detect that an attack is taking place, and will shut down before any onions are delivered to their destinations. Specifically, if enough onions survive half of the rounds, then privacy is guaranteed through having sufficient cover; otherwise, privacy is guaranteed because *no* message reaches its final destination with overwhelming probability. So the adversary does not learn anything about the destination of Alice's onions.

In order to make it possible for the mix-nodes to detect that an attack is taking place, our honest users create "checkpoint" onions. These onions don't carry any messages; instead, they are designed to be "verified" by a particular mix-node in a particular round. These checkpoint onions are expected by the mix-node, so if one of them does not arrive, the mix-node in question realizes that something is wrong. If enough checkpoint onions are missing, the mix-node determines that an attack is underway and shuts down. Two different users, Alice and Allison, use a PRF with a shared key (this shared key need not be pre-computed, but can instead be derived from a discrete-log based public-key infrastructure under the decisional Diffie-Hellman assumption) in order to determine whether Alice should create a checkpoint onion that will mirror Allison's checkpoint onion.

Related work

Encryption schemes that are appropriate for onion routing are known [2, 6]. Several papers attempted to define anonymity for communications protocols and to analyze Tor [20, 21, 28]. Backes et al. [3] were the first to consider a notion inspired by differential privacy [18] but, in analyzing Tor, they assume an adversary with only a partial view of the network. There are also some studies on anonymity protocols, other than onion routing protocols, that were analyzed using information-theoretic measures [1, 4, 8, 16, 24]. In contrast, all the protocols presented in this paper are provably secure against powerful adversaries that can observe all network traffic. The system, Vuvuzela [31], assumes that all messages travel through the same set of dedicated servers and is, therefore, impractical compared to Tor. Recently proposed systems, Stadium [30] and Atom [25] are distributed but not robust; they rely on verifiable shuffling to detect and abort. A variant of Atom is robust at a cost in security; it only achieves k-anonymity [25]. In contrast, our solution for the active adversary is distributed while maintaining low latency, and robust while being provably secure.

Achieving anonymous channels using heavier cryptographic machinery has been considered also. One of the earliest examples is Chaum's dining cryptographer's protocol [9]. Rackoff and Simon [27] use secure multiparty computation for providing security from active adversaries. Other cryptographic tools used in constructing anonymity protocols include oblivious RAM (ORAM) and private information retrieval (PIR) [11, 12]. Corrigan-Gibbs et al.'s Riposte solution makes use of a global bulletin board and has a latency of a couple of days [12]. The aforementioned Stadium [25] is another solution for a public forum. Blaze et al. [5] provided an anonymity protocol in the wireless (rather than point-to-point) setting.

2 Preliminaries

Notation

By the notation $[n]$, we mean the set $\{1, \ldots, n\}$ of integers. The output a of an algorithm A is denoted by $a \leftarrow A$. For a set S, we write $s \leftarrow S$ to represent that s is a uniformly random sample from the set S and $|S|$, to represent its cardinality. A realization d of a distribution D is denoted $d \sim D$; by $d \sim \mathbf{Binomial}(N, p)$, we mean that d is a realization of a binomial random variable with parameters N and p. By $\log(n)$, we mean the logarithm of n, base 2; and by $\ln(n)$, we mean the natural log of n.

A function $f : \mathbb{N} \to \mathbb{R}$ is negligible in λ, written $f(\lambda) = \mathbf{negl}(\lambda)$, if for every polynomial $p(\cdot)$ and all sufficiently large λ, $f(\lambda) < 1/p(\lambda)$. When λ is the security parameter, we say that an event occurs with overwhelming probability if it is the complement of an event with probability negligible in λ. Two families of distributions $\{D_{0,\lambda}\}_{\lambda \in \mathbb{N}}$ and $\{D_{1,\lambda}\}_{\lambda \in \mathbb{N}}$ are

statistically close if the statistical distance between $D_{0,\lambda}$ and $D_{1,\lambda}$ is negligible in λ; we abbreviate this notion by $D_0 \approx_s D_1$ when the security parameter is clear by context. We use the standard notion of a pseudorandom function [22, Ch. 3.6].

Onion routing

Following Camenisch and Lysyanskaya's work on cryptographic onions [6], an *onion routing scheme* is a triple of algorithms: $(\mathsf{Gen}, \mathsf{FormOnion}, \mathsf{ProcOnion})$. The algorithm, Gen, generates a public-key infrastructure for a set of parties. The algorithm, $\mathsf{FormOnion}$, forms onions; and the algorithm, $\mathsf{ProcOnion}$, processes onions.

Given a set $[N]$ of parties, for every $i \in [N]$, let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$ be the key pair generated for party $i \in [N]$, where λ denotes the security parameter.

$\mathsf{FormOnion}$ takes as input: a message m, an ordered list $(P_1, \ldots, P_{L+1})$ of parties from $[N]$, and the public-keys $(\mathsf{pk}_{P_1}, \ldots, \mathsf{pk}_{P_{L+1}})$ associated with these parties, and a list $(s_1, \ldots, s_L)$ of (possibly empty) strings that are nonces associated with layers of the onion. The party P_{L+1} is interpreted as the *recipient* of the message, and the list $(P_1, \ldots, P_{L+1})$ is the *routing path* of the message. The output of $\mathsf{FormOnion}$ is a sequence $(O_1, \ldots, O_{L+1})$ of onions. Because it is convenient to think of an onion as a layered encryption object, where processing an onion O_r produces the next onion O_{r+1}, we sometimes refer to the process of revealing the next layer of an onion as "decrypting the onion", or "peeling the onion". For every $r \in [L]$, only party P_r can peel onion O_r to reveal the next layer, $(P_{r+1}, O_{r+1}, s_{r+1}) \leftarrow \mathsf{ProcOnion}(\mathsf{sk}_{P_r}, O_r, P_r)$, of the onion containing the "peeled" onion O_{r+1}, the "next destination" P_{r+1}, and the nonce s_{r+1}. Only the recipient P_{L+1} can peel the innermost onion O_{L+1} to reveal the message, $m \leftarrow \mathsf{ProcOnion}(\mathsf{sk}_{P_{L+1}}, O_{L+1}, P_{L+1})$.

Let O_0 be an onion formed from running $\mathsf{FormOnion}(m_0, P^0, \mathsf{pk}^0, s^0)$, and let O_1 be another onion formed from running $\mathsf{FormOnion}(m_1, P^1, \mathsf{pk}^1, s^1)$. Importantly, a party that can't peel either onion can't tell which input produced which onion. See Camenisch and Lysyanskaya's paper [6] for formal definitions.

In our protocols, a sender of a message m to a recipient j "forms an onion" by generating nonces and running the $\mathsf{FormOnion}$ algorithm on the message m, a routing path $(P_1, \ldots, P_L, j)$, the public keys $(\mathsf{pk}_{P_1}, \ldots, \mathsf{pk}_{P_L}, \mathsf{pk}_j)$ associated with the parties on the routing path, and the generated nonces; the "formed onion" is the first onion O_1 from the list of outputted onions. The sender sends O_1 to the first party P_1 on the routing path, who processes it and sends the peeled onion O_2 to the next destination P_2, and so on, until the last onion O_{L+1} is received by the recipient j, who processes it to obtain the message m.

3 Definitions

We model the network as a graph with N nodes, and we assume that these nodes are synchronized. This way, any onion can be sent from any sender to any receiver, and also its transmission occurs within a single round.

Every participant is a user client, and some user clients also serve as mix-nodes. In all the definitions, the N users participating in an communications protocol Π are labeled $1, \ldots, N$; and the number N of users is assumed to be polynomially-bounded in the security parameter λ. Every input to a protocol is an N-dimensional vector. When a protocol runs on input $\sigma = (\sigma_1, \ldots, \sigma_N)$, it means that the protocol is instantiated with each user i receiving σ_i as input. $\mathcal{M}$ denotes the (bounded) message space. A message pair (m, j) is properly formed if $m \in \mathcal{M}$ and $j \in [N]$. The input σ_i to each user $i \in [N]$ is a collection of properly formed message pairs, where $(m, j) \in \sigma_i$ means that user i intends on sending message m to

user j. Let $M(\sigma)$ denote the "messages in σ". It is the multiset of all message pairs in σ, that is $M(\sigma_1, \ldots, \sigma_N) = \bigcup_{i=1}^{N} \{(m, j) \in \sigma_i\}$.

For analyzing our solutions, it is helpful to first assume an idealized version of an encryption scheme, in which the ciphertexts are information-theoretically unrelated to the plaintexts that they encrypt and reveal nothing but the length of the plaintext. Obviously, such encryption schemes do not exist computationally, but only in a hybrid model with an oracle that realizes an ideal encryption functionality, such as that of Canetti [7]. When used in forming onions, such an encryption scheme gives rise to onions that are information-theoretically independent of their contents, destinations, and identities of the mix-nodes. Our real-life proposal, of course, will use standard computationally secure encryption [17]. We discuss the implications of this in the full version of this paper.

Views and outputs

We consider the following standard adversary models, in increasing order of capabilities:

1. **Network adversary.** A network adversary can observe the bits flowing on every link of the network. (Note that if the peer-to-peer links are encrypted in an idealized sense, then the only information that the adversary can use is the volume flow.)
2. **Passive adversary.** In addition to the capabilities of a network adversary, a passive adversary can monitor the internal states and operations of a constant fraction of the parties. The adversary's choices for which parties to monitor are made non-adaptively over the course of the execution run.
3. **Active adversary.** In addition to the capabilities of a network adversary, an active adversary can corrupt a constant fraction of the parties. The adversary's choices for which parties to corrupt are made non-adaptively over the course of the execution run. The adversary can change the behavior of corrupted parties to deviate arbitrarily from the protocol.

Let Π be a protocol, and let σ be a vector of inputs to Π. Given an adversary $\mathcal{A}$, the view $V^{\Pi,\mathcal{A}}(\sigma)$ of $\mathcal{A}$ is its observables from participating in Π on input σ plus any randomness used to make its decisions. With idealized secure peer-to-peer links, the observables for a network adversary are the traffic volumes on all links; whereas for the passive and active adversaries, the observables additionally include the internal states and computations of all monitored / corrupted parties at all times.

Given an adversary $\mathcal{A}$, the output $O^{\Pi,\mathcal{A}}(\sigma) = (O_1^{\Pi,\mathcal{A}}(\sigma), \ldots, O_N^{\Pi,\mathcal{A}}(\sigma))$ of Π on input σ is a vector of outputs for the N parties.

3.1 Privacy definitions

How do we define security for an anonymous channel? The adversary's view also includes the internal states of corrupted parties. In such case, we may wish to protect the identities of honest senders from the recipients that are in cahoots with the adversary. However, even an ideal anonymous channel cannot prevent the contents of messages (including the volumes of messages) from providing a clue on who sent the messages; thus any "message content" leakage should be outside the purview of an anonymous channel. To that end, we say that an communications protocol is secure if it is difficult for the adversary to learn who is communicating with whom, beyond what leaks from captured messages.

Below, we provide two flavors of this security notion; we will prove that our constructions achieve either statistical privacy or (ϵ, δ)-differential privacy [19, Defn. 2.4] in the idealized encryption setting.

▶ **Definition 2** (Statistical privacy). Let Σ^* be the input set consisting of every input of the form $\sigma = (\{(m_1, \pi(1))\}, \ldots, \{(m_N, \pi(N))\})$, where $\pi : [N] \to [N]$ is any permutation function over the set $[N]$, and $m_i \in \mathcal{M}$ for every $i \in [N]$. A communications protocol Π is *statistically private* from every adversary from the class $\mathbb{A}$ if for all $\mathcal{A} \in \mathbb{A}$ and for all $\sigma_0, \sigma_1 \in \Sigma^*$ that differ only on the honest parties' inputs and outputs, the adversary's views $V^{\Pi,\mathcal{A}}(\sigma_0)$ and $V^{\Pi,\mathcal{A}}(\sigma_1)$ are statistically indistinguishable, i.e., $\Delta(V^{\Pi,\mathcal{A}}(\sigma_0), V^{\Pi,\mathcal{A}}(\sigma_1)) = \mathbf{negl}(\lambda)$, where $\lambda \in \mathbb{N}$ denotes the security parameter, and $\Delta(\cdot,\cdot)$ denotes statistical distance (i.e., total variation distance). Π is perfectly secure if the statistical distance is zero instead.

▶ **Definition 3** (Distance between inputs). The *distance* $d(\sigma_0, \sigma_1)$ between two inputs $\sigma_0 = (\sigma_{0,1}, \ldots, \sigma_{0,N})$ and $\sigma_1 = (\sigma_{1,1}, \ldots, \sigma_{1,N})$ is given by $d(\sigma_0, \sigma_1) \stackrel{\text{def}}{=} \sum_{i=1}^{N} |\sigma_{0,i} \nabla \sigma_{1,i}|$, where $(\cdot \nabla \cdot)$ denotes the symmetric difference.

▶ **Definition 4** (Neighboring inputs). Two inputs σ_0 and σ_1 are *neighboring* if $d(\sigma_0, \sigma_1) \leq 1$.

▶ **Definition 5** ((ϵ, δ)-DP [19, Defn. 2.4]). Let Σ be the set of all valid inputs. A communications protocol is (ϵ, δ)-*DP* from every adversary in the class $\mathbb{A}$ if for all $\mathcal{A} \in \mathbb{A}$, for every neighboring inputs $\sigma_0, \sigma_1 \in \Sigma$ that differ only on an honest party's input and an honest party's output, and any set $\mathcal{V}$ of views, $\Pr[V^{\Pi,\mathcal{A}}(\sigma_0) \in \mathcal{V}] \leq e^{\epsilon} \cdot \Pr[V^{\Pi,\mathcal{A}}(\sigma_1) \in \mathcal{V}] + \delta$.

While differential privacy is defined with respect to neighboring inputs, it also provides (albeit weaker) guarantees for non-neighboring inputs; it is known that the security parameters degrade proportionally in the distance between the inputs [19].

3.2 Other performance metrics

Since message delivery cannot be guaranteed in the presence of an active adversary, we define correctness with respect to *passive* adversaries.

▶ **Definition 6** (Correctness). A communications protocol Π is *correct* on an input $\sigma \in \Sigma$ if for any passive adversary $\mathcal{A}$, and for every recipient $j \in [N]$, the output $O_j^{\Pi,\mathcal{A}}(\sigma)$ corresponds to the multiset of all messages for recipient j in the input vector σ. That is, $O_j^{\Pi,\mathcal{A}}(\sigma) = \{m \,|(m, j) \in M(\sigma)\}$, where $M(\sigma)$ denotes the multiset of all messages in σ.

Efficiency of OR protocols

The communication complexity blow-up of an onion routing (OR) protocol measures how many more onion transmissions are required by the protocol, compared with transmitting the messages in onions directly from the senders to the recipients (without passing through intermediaries). We assume that every message $m \in \mathcal{M}$ in the message space $\mathcal{M}$ "fits" into a single onion. The communication complexity is measured in unit onions, which is appropriate when the parties pass primarily onions to each other.

▶ **Definition 7** (Communication complexity blow-up). The *communication complexity blow-up* of an OR protocol Π is defined with respect to an input vector σ and an adversary $\mathcal{A}$. Denoted $\gamma^{\Pi,\mathcal{A}}(\sigma)$, it is the expected ratio between the total number $\Gamma^{\Pi,\mathcal{A}}(\sigma)$ of onions transmitted in protocol Π and the total number $|M(\sigma)|$ of messages in the input vector. That is, $\gamma^{\Pi,\mathcal{A}}(\sigma) \stackrel{\text{def}}{=} \mathbb{E}\left[\frac{\Gamma^{\Pi,\mathcal{A}}(\sigma)}{|M(\sigma)|}\right]$.

▶ **Definition 8** (Server load). The *server load* of an OR protocol Π is defined with respect to an input vector σ and an adversary $\mathcal{A}$. It is the expected number of onions processed by a single party in a round.

▶ **Definition 9** (Latency). The *latency* of an OR protocol Π is defined with respect to an input vector σ and an adversary $\mathcal{A}$. It is the expected number of rounds in a protocol execution.

In addition to having low (i.e., polylog in the security parameter) communication complexity blow-ups, we will show that our OR protocols have low (i.e., polylog in the security parameter) server load and low (i.e., polylog in the security parameter) latency.

4 The passive adversary

Communication patterns can trivially be hidden by sending every message to every participant in the network, but this solution is not scalable as it requires a communication complexity blow-up that is linear in the number of participants. Here, we prove that an OR protocol can provide anonymity from the passive adversary while being practical with low communication complexity and low server load.

To do this, every user must send and receive the same number of messages as any other user; otherwise, the sender-receiver relation can leak from the differing volumes of messages sent and received by the users. In other words, every user essentially commits to sending a message, be it the empty message $\perp$ to itself. Let Σ^* be the set of all input vectors of the form $\sigma = (\{(m_1, \pi(1))\}, \ldots, \{(m_N, \pi(N))\})$, where $\pi : [N] \to [N]$ is any permutation function over the set $[N]$, and $m_1, \ldots, m_N$ are any messages from the message space $\mathcal{M}$; our solution, Π_p, is presented in the setting where the input vector is constrained to Σ^*.

Let $[N]$ be the set of users, and $\mathcal{S} = \{S_1, \ldots, S_n\} \subset [N]$ the set of servers. Π_p uses a secure onion routing scheme, denoted by $\mathcal{OR} = (\mathsf{Gen}, \mathsf{FormOnion}, \mathsf{ProcOnion})$, as a primitive building block. For every $i \in [N]$, let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$ be the key pair generated for party i, where λ denotes the security parameter.

During a setup phase, each user $i \in [N]$ creates an onion. On input $\sigma_i = \{(m, j)\}$, user i first picks a sequence $T_1, \ldots, T_L$ servers, where each server is chosen independently and uniformly at random, and then forms an onion from the message m, the routing path $(T_1, \ldots, T_L, j)$, the public keys $(\mathsf{pk}_{T_1}, \ldots, \mathsf{pk}_{T_L}, \mathsf{pk}_j)$ associated with the parties on the routing path, and a list of empty nonces. At the first round of the protocol run, user i sends the formed onion to the first hop T_1 on the routing path.

After every round $i \in [L]$ (but before round $i+1$) of the protocol run, each server processes the onions it received at round i. At round $i+1$, the resulting peeled onions are sent to their respective next destinations in random order. At round $L+1$, every user receives an onion and processes it to reveal a message.

Correctness and efficiency

Clearly, Π_p is correct. In Π_p, N messages are transmitted in each of the $L+1$ rounds of the protocol run. Thus, the communication complexity blow-up and the latency are both $L+1$. The server load is $\frac{N}{n}$.

Privacy

To prove that Π_p is statistically private from the passive adversary, we first prove that it is secure from the network adversary.

▶ **Theorem 10.** *Π_p is statistically private from the network adversary when $\frac{N}{n} = \Omega(\log^2 \lambda)$, and $L = \Omega(\log^2 \lambda)$, where $\lambda \in \mathbb{N}$ denotes the security parameter.*

Proof. Let $\frac{N}{n} = \alpha \log^2 \lambda$; so that after each round, every location receives $\alpha \log^2 \lambda$ onions in expectation. We recast our problem as a balls-in-bins problem, where the balls are the onions, and the bins, the locations. At every round of the protocol, all $\alpha n \log^2 \lambda$ balls (i.e., onions) are thrown uniformly at random into n bins (i.e., each onion is routed to one of n locations, chosen independently and uniformly at random).

Fix any target sender U, and let $X^r = (X_1^r, \ldots, X_n^r)$ be a vector of non-negative numbers summing to one, representing $\mathcal{A}$'s best estimate for the location of U's ball after r rounds (and before round $r+1$); for every $i \in [n]$, X_i^r is the likelihood that bin i contains U's ball after r rounds. Let $(X^r_{f_r(1)}, \ldots, X^r_{f_r(n)})$ be the result of sorting $(X_1^r, \ldots, X_n^r)$ in decreasing order, where $f_r : [n] \to [n]$ is a permutation function over the set $[n]$. For every $i \in [n]$, let $b_i^r = f_r(i)$ be the index of the bin with the i-th largest likelihood at round r.

W.l.o.g. we assume that n is divisible by three. We partition the bins into three groups G_1^r, G_2^r, and G_3^r; such that G_1^r contains all the *balls* in the top one-third most likely bins $b_1^r, \ldots, b_{\frac{n}{3}}^r$; G_3^r contains all the balls in the bottom one-third most likely bins $b^r_{\frac{2n}{3}+1}, \ldots, b_n^r$; and G_2^r contains all the balls in the remaining bins $b^r_{\frac{n}{3}+1}, \ldots, b^r_{\frac{2n}{3}}$.

For each $j \in [3]$, let $O_j^r \in G_j^r$ be a ball with the maximum likelihood of being U's onion among the balls in group G_j^r. For any $d \in (0,1)$, let $d' = 1 - \frac{1}{1+d}$. Let c_j^r be the bin containing O_j^r. The bin c_j^r contains at least $(1-d')\alpha \log^2 \lambda$ balls (Chernoff bounds for Poisson trials). It follows that,

$$\Pr[O_j^r \text{ is } U\text{'s onion}] \leq (1+d)\frac{X^r_{c_j^r}}{\alpha \log^2 \lambda} \leq (1+d)\frac{X^r_{\frac{(j-1)n}{3}+1}}{\alpha \log^2 \lambda} \tag{1}$$

with overwhelming probability, where $X^r_{\frac{(j-1)n}{3}+1}$ is the likelihood of the most likely bin in group G_j.

The number of balls contained in each group G_j^r is arbitrarily close to the expected number $\frac{\alpha}{3} n \log^2 \lambda$ of balls in a group (Chernoff bounds). Thus, the most probable bin b_1^{r+1} after the next round receives at most $\frac{(1+d)\alpha}{3} \log^2 \lambda$ balls from each of the three groups: G_1^r, G_2^r, and G_3^r. From (1), this implies that, with overwhelming probability, $X_1^{r+1} \leq \frac{(1+d)^2}{3} \sum_{j=1}^3 X^r_{\frac{(j-1)n}{3}+1}$. Using a symmetric argument, we can conclude that, with overwhelming probability, $X_n^{r+1} \geq \frac{(1-d)^2}{3} \sum_{j=1}^3 X^r_{\frac{jn}{3}}$, where $X^r_{\frac{jn}{3}}$ is the likelihood of the least likely bin in group G_j.

For all $r \in [L]$, define $g^r = X_1^r - X_n^r$ as the difference in likelihoods between the most and least likely bins at round r.

$$g^{r+1} \leq \frac{(1+d)^2 \sum_{j=1}^3 X^r_{\frac{(j-1)n}{3}+1}}{3} - \frac{(1-d)^2 \sum_{j=1}^3 X^r_{\frac{jn}{3}}}{3} \leq \frac{1}{2}\left(X_1^r - X_n^r\right) = \frac{g^r}{2},$$

where the latter inequality follows from telescopic cancelling, since $X^r_{\frac{n}{3}+1} \leq X^r_{\frac{n}{3}}$, and $X^r_{\frac{2n}{3}+1} \leq X^r_{\frac{2n}{3}}$.

The difference g^r is at least halved at every round. By round $\log^2 \lambda$, the difference is negligible in λ. Thus, after traveling L random hops, each onion becomes unlinked from its sender. Since everyone sends the same number of onions, and everyone receives the same number of onions; it follows that the adversary's views from any two inputs are statistically indistinguishable.

In the proof above, the bins were partitioned into three groups at every round. By partitioning the bins into an appropriately large constant number of groups, we can show that Π_p achieves statistical privacy after $L = \Omega(\log^2 \lambda)$ rounds. ◀

We are now ready to prove the main result of this section:

▶ **Theorem 11.** *Π_p is statistically private from the passive adversary capable of monitoring any constant fraction $\kappa \in [0, 1)$ of the servers when $\frac{N}{n} = \Omega(\log^2 \lambda)$, and $L = \Omega(\log^2 \lambda)$, where $\lambda \in \mathbb{N}$ denotes the security parameter.*

Proof. We prove this by cases.

In the first case, σ_1 is the same as σ_0 except that the inputs of two users are swapped, i.e., $d(\sigma_0, \sigma_1) = 2$. Using Chernoff bounds for Poisson trials, there are at least some polylog number of rounds where the swapped onions are both routed to an honest bin (not necessarily the same bin). From Theorem 10, after the polylog number of steps, the locations of these two target onions are statistically indistinguishable from each other.

In the second case, $d(\sigma_0, \sigma_1) > 2$. However, the distance between σ_0 and σ_1 is always polynomially bounded. By a simple hybrid argument, it follows that $V^{\Pi,\mathcal{A}}(\sigma_0) \approx_s V^{\Pi,\mathcal{A}}(\sigma_1)$ from case 1. ◀

Remark: Protocol Π_p is not secure from the active adversary. This is because, with non-negligible probability, any honest user will choose a corrupted party as its first hop on its onion's routing path, in which case the adversary can drop the target user's onion at the first hop and observe who does not receive an onion at the last round.

5 The active adversary

We now present an OR protocol, Π_a, that is secure from the active adversary. The setting for Π_a is different from that of our previous solution in a couple of important ways. Whereas Π_p is *statistically* private from the passive adversary, Π_a is only *differentially* private from the active adversary. The upside is that we are no longer constrained to operate in the simple I/O setting; the input can be any valid input.

We let $[N]$ be the set of N parties participating in a protocol. Every party is both a user and a server. As before, $\mathcal{OR} = (\mathsf{Gen}, \mathsf{FormOnion}, \mathsf{ProcOnion})$ is a secure onion routing scheme; and for every $i \in [N]$, $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\lambda)$ denotes the key pair generated for party i, where λ is the security parameter. Further, we assume that every pair $(i, k) \in [N]^2$ of parties shares a common secret key[3], denoted by $\mathsf{sk}_{i,k}$. F is a pseudorandom function (PRF).

We describe the protocol by the setup and routing algorithms for party $i \in [N]$; each honest party runs the same algorithms.

Setup

Let $L = \beta \log^2 \lambda$ for some constant $\beta > 0$. During the setup phase, party i prepares a set of onions from its input. For every message pair $u = \{m, j\}$ in party i's input, party i picks a sequence $T_1^u, \ldots, T_L^u$ of parties, where each party T_ℓ^u is chosen independently and uniformly at random, and forms an onion from the message m, the routing path $(T_1^u, \ldots, T_L^u, j)$, the public keys $(\mathsf{pk}_{T_1^u}, \ldots, \mathsf{pk}_{T_L^u}, \mathsf{pk}_j)$, and a list of empty nonces.

Additionally, party i forms some dummy onions, where a dummy onion is an onion formed using the empty message $\perp$.

1: **for** every index $(r, k) \in [L] \times [N]$:
2: compute $b \leftarrow F(\mathsf{sk}_{i,k}, \mathsf{session} + r, 0)$, where $\mathsf{session} \in \mathbb{N}$ denotes the protocol instance.
3: **if** $b \equiv 1$ – set to occur with frequency $\frac{\alpha \log^2 \lambda}{N}$ for some constant $\alpha > 0$ – **do**:

[3] In practice, the shared keys do not need to be set up in advance; they can be generated as needed from an existing PKI, e.g., using Diffie-Hellman.

4: choose a list $T^{r,k} = (T_1^{r,k}, \ldots, T_{r-1}^{r,k}, T_{r+1}^{r,k}, \ldots, T_{L+1}^{r,k})$ parties, where each party is chosen independently and uniformly at random;
5: create a list $s^{r,k} = (s_1^{r,k}, \ldots, s_2^{r,k})$ of nonces, where $s_r^{r,k} = (\mathsf{checkpt}, F(\mathsf{sk}_{i,k}, \mathsf{session} + r, 1))$, and all other elements of $s^{r,k}$ are $\perp$; and
6: form a dummy onion using the message $\perp$, the routing path $T^{r,k}$, the public keys associated with $T^{r,k}$, and the list $s^{r,k}$ of nonces. **end if, end for**

The additional information $s_r^{r,k}$ is embedded in only the r-th layer; no additional information is embedded in any other layer. At the first round of the protocol run (after setup), all formed onions are sent to their respective first hops.

Routing

If party i forms a dummy onion with nonce $s_r^{r,k}$ embedded in the r-th layer, then it expects to receive a symmetric dummy onion at the r-th round formed by another party k that, when processed, reveals the same nonce $s_r^{r,k}$. If many checkpoint nonces are missing, then party i knows to abort the protocol run.

After every round $r \in [L]$ (but before round $r+1$), party i peels the onions it received at round r and counts the number of missing checkpoint nonces. If the count exceeds a threshold value t, the party aborts the protocol run; otherwise, at round $r+1$, the peeled onions are sent to their next destinations in random order. After the final round, party i outputs the set of messages revealed from processing its the onions it receives at round $L+1$.

Correctness and efficiency

Recalling that correctness is defined with respect to the passive adversary, Π_a is clearly correct. Moreover, unless an honest party aborts the protocol run, all messages that are not dropped by the adversary are delivered to their final destinations. In Π_a, the communication complexity blow-up is $O(\log^6 \lambda)$, since the latency is $L + 1 = O(\log^2 \lambda)$ rounds, and the server load is $O(\log^4 \lambda)$.

Privacy

To prove that Π_a is secure, we require that the thresholding mechanism does its job:

▶ **Lemma 12.** *In Π_a, if F is a random function, $t = c(1-d)(1-\kappa)^2\alpha \log^2 \lambda$ for some $c, d \in (0,1)$, and an honest party does not abort within the first r rounds of the protocol run, then with overwhelming probability, at least $(1-c)$ of the dummy onions created between honest parties survive at least $(r-1)$ rounds, even in the presence of an active adversary non-adaptively corrupting a constant fraction $\kappa \in [0,1)$ of the parties.*

The proof relies on a known concentration bound for the hypergeometric distribution [23] and can be found in the full version of this paper.

▶ **Theorem 13.** *If, in Π_a, F is a random function, $N \geq \frac{3}{1-\kappa}$, and $t = c(1-d)(1-\kappa)^2\alpha \log^2 \lambda$ for some $c, d \in (0,1)$, then, for $\alpha\beta \geq -\frac{36(1+\epsilon/2)^2 \ln(\delta/4)}{(1-c)(1-\kappa)^2\epsilon^2}$, Π_a is (ϵ, δ)-DP from the active adversary non-adaptively corrupting a constant fraction $\kappa \in [0,1)$ of the parties.*

Proof. The proof is by cases.

Case 1: All honest parties abort within the first half of the protocol run. With overwhelming probability, no onion created by an honest party will be delivered to its final destination (Chernoff bounds for Poisson trials), and so the adversary doesn't learn anything.

Case 2: Some honest party doesn't abort within the first half of the protocol run. Let $\mathcal{A}$ be any adversary that non-adaptively corrupts a constant $\kappa \in [0, 1)$ of the parties. Suppose that for every onion that survive the first half of the protocol run, a dark angel provides the adversary $\mathcal{A}$ with the second half of the onion's routing path. Further suppose that no other onions are dropped in the second half of the protocol run. (If more onions are dropped, then Π_a is secure from the post-processing theorem for differential privacy [19, Proposition 2.1].)

For any two neighboring inputs σ_0 and σ_1, the only difference in the adversary's views, $V^{\Pi_a,\mathcal{A}}(\sigma_0)$ and $V^{\Pi_a,\mathcal{A}}(\sigma_1)$, is the routing of a single onion O. If there is an honest party who does not abort within the first half of the protocol run, then from Lemma 12, some constant fraction of the dummy onions created by the honest parties survive the first half of the protocol run with overwhelming probability. So, from Theorem 11, the onions are no longer linked to their senders by the end of the first half of the protocol run. Thus, the only information that $\mathcal{A}$ could find useful is the volume of onions sent out by the sender P_s of the extra onion O and the volume of onions received by the receiver P_r of O.

Let X denote the number of dummy onions created by P_s. For every $(k, r) \in [L] \times [N]$, an honest sender P_s creates a dummy onion with probability $\frac{\alpha \log^2 \lambda}{N}$; so $X \sim \textbf{Binomial}(H, p)$, where $H = LN$, and $p = \frac{\alpha \log^2 \lambda}{N}$.

Let $Y \sim \textbf{Binomial}(G, q)$ be another binomial random variable with parameters $G = \frac{L(1-\kappa)^2 N^2}{3}$ and $q = \frac{(1-c)\alpha \log^2 \lambda}{N^2}$. For $N \geq \frac{3}{1-\kappa}$ and sufficiently small $d > 0$, $G \leq (1 - d)L\binom{(1-\kappa)N-1}{2}$; thus, with overwhelming probability, Y is less than the number of dummy onions created between honest non-P_s parties and received by P_r in the final round (Chernoff bounds).

Let $\mathcal{O} \stackrel{\text{def}}{=} \mathbb{N} \times \mathbb{N}$ be the sample space for the multivariate random variable (X, Y).

Let $\mathcal{O}_1$ be the event that $|X - \mathbb{E}[X]| \leq d'\mathbb{E}[X]$, and $|Y - \mathbb{E}[Y]| \leq d'\mathbb{E}[Y]$, where $d' = \frac{\epsilon/2}{1+\epsilon/2}$, $\mathbb{E}[X] = Hp$ is the expected value of X, and $\mathbb{E}[Y] = Gq$ is the expected value of Y; and let $\bar{\mathcal{O}_1}$ be the complement of $\mathcal{O}_1$.

For every $(x, y) \in \mathcal{O}_1$, we can show that

$$\max\left(\frac{\Pr[(X,Y) = (x,y)]}{\Pr[(X,Y) = (x+1,y+1)]}, \frac{\Pr[(X,Y) = (x+1,y+1)]}{\Pr[(X,Y) = (x,y)]}\right) \leq e^{\epsilon}. \tag{2}$$

We can also show that the probability of the tail event $\bar{\mathcal{O}_1}$ occurring is negligible in λ and at most δ when $\alpha\beta \geq -\frac{36(1+\epsilon/2)^2 \ln(\delta/4)}{(1-c)(1-\kappa)^2\epsilon^2}$. (See the full version of this paper.)

Any event $\mathcal{E}$ can be decomposed into two subsets $\mathcal{E}_1$ and $\mathcal{E}_2$, such that (1) $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$, (2) $\mathcal{E}_1 \subseteq \mathcal{O}_1$, and (3) $\mathcal{E}_2 \subseteq \bar{\mathcal{O}_1}$. It follows that, for every event $\mathcal{E}$,

$$\Pr[(X,Y) \in \mathcal{E}] \leq e^{\epsilon} \cdot \Pr[(X+1,Y+1) \in \mathcal{E}] + \delta, \text{ and} \tag{3}$$

$$\Pr[(X+1,Y+1) \in \mathcal{E}] \leq e^{\epsilon} \cdot \Pr[(X,Y) \in \mathcal{E}] + \delta. \tag{4}$$

The views $V^{\Pi_a,\mathcal{A}}(\sigma_0)$ and $V^{\Pi_a,\mathcal{A}}(\sigma_1)$ are the same except that O exists in one of the views but not in the other. Thus, (3) and (4) suffice to show that for any set $\mathcal{V}$ of views and for any $b \in \{0, 1\}$, $\Pr[V_b^{\Pi_a,\mathcal{A}} \in \mathcal{V}] \leq e^{\epsilon} \cdot \Pr[V_{\bar{b}}^{\Pi_a,\mathcal{A}} \in \mathcal{V}] + \delta$, where $\bar{b} = b + 1 \mod 2$. ◀

Remark: Our protocols are for a *single-pass* setting, where the users send out messages once. It is clear how our statistical privacy results would compose for the multi-pass case. To prove that Π_a also provides differential privacy in the multi-pass scenario – albeit for degraded security parameters – we can use the k-fold composition theorem [19]; the noise falls at a rate of the square-root of the number of runs.

References

1 Mário S. Alvim, Miguel E. Andrés, Konstantinos Chatzikokolakis, Pierpaolo Degano, and Catuscia Palamidessi. Differential privacy: on the trade-off between utility and information leakage. In *FAST 2011*, pages 39–54. Springer, 2011.

2 Michael Backes, Ian Goldberg, Aniket Kate, and Esfandiar Mohammadi. Provably secure and practical onion routing. In *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*, pages 369–385. IEEE, 2012.

3 Michael Backes, Aniket Kate, Praveen Manoharan, Sebastian Meiser, and Esfandiar Mohammadi. AnoA: A framework for analyzing anonymous communication protocols. Cryptology ePrint Archive, Report 2014/087, 2014. URL: `http://eprint.iacr.org/2014/087`.

4 Ron Berman, Amos Fiat, and Amnon Ta-Shma. Provable unlinkability against traffic analysis. In Ari Juels, editor, *FC 2004*, volume 3110 of *LNCS*, pages 266–280, Key West, USA, feb 9–12, 2004. Springer, Heidelberg, Germany.

5 Matt Blaze, John Ioannidis, Angelos D Keromytis, Tal Malkin, and Aviel D Rubin. Anonymity in wireless broadcast networks. *IJ Network Security*, 8(1):37–51, 2009.

6 Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 169–187, Santa Barbara, CA, USA, aug 14–18, 2005. Springer, Heidelberg, Germany.

7 Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145, Las Vegas, NV, USA, oct 14–17, 2001. IEEE Computer Society Press.

8 Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Prakash Panangaden. Anonymity protocols as noisy channels. *Information and Computation*, 206(2–4):378–401, 2008.

9 David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

10 David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

11 David A. Cooper and Kenneth P. Birman. Preserving privacy in a network of mobile computers. In *1995 IEEE Symposium on Security and Privacy*, pages 26–38. IEEE Computer Society Press, 1995.

12 Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 321–338, San Jose, CA, USA, may 17–21, 2015. IEEE Computer Society Press. `doi:10.1109/SP.2015.27`.

13 George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *2003 IEEE Symposium on Security and Privacy*, pages 2–15, Berkeley, CA, USA, may 11–14, 2003. IEEE Computer Society Press.

14 Roger Dingledine and Nick Mathewson. Tor: An anonymous internet communication system. In *Workshop on Vanishing Anonymity, Proceedings from the Conference on Computers, Freedom, and Privacy*, 2005.

15 Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

16 Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540, Interlaken, Switzerland, may 2–6, 2004. Springer, Heidelberg, Germany.

17 Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

18 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *TCC 2006*,

volume 3876 of *LNCS*, pages 265–284, New York, NY, USA, mar 4–7, 2006. Springer, Heidelberg, Germany.

19 Dwork, Cynthia and Roth, Aaron. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

20 Joan Feigenbaum, Aaron Johnson, and Paul Syverson. Probabilistic analysis of onion routing in a black-box model. *ACM Transactions on Information and System Security (TISSEC)*, 15(3):1–28, Nov 2012.

21 Joan Feigenbaum, Aaron Johnson, and Paul F. Syverson. A model of onion routing with provable anonymity. In Sven Dietrich and Rachna Dhamija, editors, *FC 2007*, volume 4886 of *LNCS*, pages 57–71, Scarborough, Trinidad and Tobago, feb 12–16, 2007. Springer, Heidelberg, Germany.

22 Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001.

23 Don Hush and Clint Scovel. Concentration of the hypergeometric distribution. *Statistics & probability letters*, 75(2):127–132, 2005.

24 Boris Köpf and David A. Basin. An information-theoretic model for adaptive side-channel attacks. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07*, pages 286–296, Alexandria, Virginia, USA, oct 28–31, 2007. ACM Press.

25 Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. Atom: Horizontally scaling strong anonymity. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 406–422, New York, NY, USA, 2017. ACM. doi:10.1145/3132747.3132755.

26 Olga Ohrimenko, Michael T. Goodrich, Roberto Tamassia, and Eli Upfal. The melbourne shuffle: Improving oblivious storage in the cloud. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *LNCS*, pages 556–567, Copenhagen, Denmark, jul 8–11, 2014. Springer, Heidelberg, Germany. doi:10.1007/978-3-662-43951-7_47.

27 Charles Rackoff and Daniel R. Simon. Cryptographic defense against traffic analysis. In *25th ACM STOC*, pages 672–681, San Diego, CA, USA, may 16–18, 1993. ACM Press.

28 Vitaly Shmatikov and Ming-Hsiu Wang. Measuring relationship anonymity in mix networks. In *Proceedings of the 5th ACM workshop on Privacy in electronic society*, pages 59–62. ACM, 2006.

29 Yixin Sun, Anne Edmundson, Nick Feamster, Mung Chiang, and Prateek Mittal. Counter-RAPTOR: Safeguarding tor against active routing attacks. In *2017 IEEE Symposium on Security and Privacy*, pages 977–992, San Jose, CA, USA, may 22–26, 2017. IEEE Computer Society Press.

30 Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 423–440, New York, NY, USA, 2017. ACM. doi:10.1145/3132747.3132783.

31 Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *SOSP 2015*, pages 137–152. ACM Press, 2015.

Resolving SINR Queries in a Dynamic Setting

Boris Aronov[1]
Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY 11201, USA
boris.aronov@nyu.edu

Gali Bar-On[2]
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel
galibar@post.bgu.ac.il

Matthew J. Katz[3]
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel
matya@cs.bgu.ac.il

Abstract

We consider a set of transmitters broadcasting simultaneously on the same frequency under the SINR model. Transmission power may vary from one transmitter to another, and a signal's strength decreases (*path loss* or *path attenuation*) by some constant power α of the distance traveled. Roughly, a receiver at a given location can hear a specific transmitter only if the transmitter's signal is stronger than the signal of all other transmitters, combined. An SINR query is to determine whether a receiver at a given location can hear any transmitter, and if yes, which one.

An approximate answer to an SINR query is such that one gets a definite YES or definite NO, when the ratio between the strongest signal and all other signals combined is well above or well below the reception threshold, while the answer in the intermediate range is allowed to be either YES or NO.

We describe several compact data structures that support approximate SINR queries in the plane in a dynamic context, i.e., where both queries and updates (insertion or deletion of a transmitter) can be performed efficiently.

2012 ACM Subject Classification Theory of computation → Computational geometry, Networks → Network algorithms

Keywords and phrases Wireless networks, SINR, dynamic insertion and deletion, interference cancellation, range searching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.145

Acknowledgements We wish to thank Pankaj K. Agarwal, Timothy Chan, Sariel Har-Peled, and Wolfgung Mulzer for discussions, hints, and outright help with some aspects of this paper. Work on this paper was initiated at the Fifth Workshop on Geometry and Graphs, Bellairs Research Institute, Barbados, 2017.

[1] B. Aronov was supported by NSF grants CCF-11-17336, CCF-12-18791, and CCF-15-40656, and by grant 2014/170 from the US-Israel Binational Science Foundation.
[2] G. Bar-On was supported by the Lynn and William Frankel Center for Computer Science.
[3] M. Katz was supported by grant 1884/16 from the Israel Science Foundation and by grant 2014/170 from the US-Israel Binational Science Foundation.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 145; pp. 145:1–145:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Approximate SINR queries in a static setting – previous results. φ is an upper bound on the fatness parameters of the reception regions and $\varphi' \geq \varphi^2$.

	Preprocessing	**Space**	**Query**
Uniform Power [5]	$O(n^2/\varepsilon)$ $(O(n\varepsilon^{-2.5}\log^4 n \log\log n)$ [3])	$O(n/\varepsilon)$	$O(\log n)$
Non-Uniform Power [10]	$O(\frac{\varphi'}{\varepsilon^2}n^2)$	$O(\frac{\varphi'}{\varepsilon^2}n)$	$O(\frac{\varphi}{\varepsilon}\log n)$

1 Introduction

The *Signal to Interference plus Noise Ratio (SINR) model* attempts to predict whether a wireless transmission is received successfully, in a setting consisting of multiple simultaneous transmitters in the presence of background noise. Let $S = \{s_1, \ldots, s_n\}$ be a set of n transmitters (points in the plane), and let p_i denote the transmission power of s_i, for $i = 1, \ldots, n$. Let q be a receiver (a point in the plane). According to the SINR model, q *receives* s_i if and only if

$$sinr(q, s_i) = \frac{\frac{p_i}{|qs_i|^\alpha}}{\sum_{j \neq i} \frac{p_j}{|qs_j|^\alpha} + N} \geq \beta \ ,$$

where $\alpha \geq 1$ and $\beta > 1$ are constants, N is a constant representing the background noise, and $|ab|$ is the Euclidean distance between points a and b.

Observe that, since $\beta > 1$, q may receive at most one transmitter – the one "closest" to it, namely, the one for which the value $\frac{p_i}{|qs_i|^\alpha}$ is maximum, or, equivalently, $\frac{1}{p_i^{1/\alpha}}|qs_i|$ is minimum. Thus, one can partition the plane into n not necessarily connected reception regions R_i, one per transmitter in S, plus an additional region $R_\emptyset$ consisting of all points where none of the transmitters is received. This partition is called the *SINR diagram* of S [5].

In their seminal paper, Avin et al. [5] studied properties of SINR diagrams, focusing on the *uniform power* version where $p_1 = p_2 = \cdots = p_n$. Their main result is that in this version the reception regions R_i are convex and fat. In the *non-uniform power* version, on the other hand, the reception regions are not necessarily connected, and their connected components are not necessarily convex or fat. In fact, they may contain holes [10].

An *SINR query* is: Given a receiver q, find the sole transmitter s that may be received by q and determine whether it is indeed received by q, i.e., whether or not $sinr(q, s) \geq \beta$. A natural question is how quickly can one answer an SINR query, following a preprocessing stage in which data structures of total size nearly linear in n are constructed. However, it seems unlikely that the answer is significantly sub-linear (as the degree of the polynomials describing region boundaries is high), so the research has focused on preprocessing to facilitate efficient *approximate* SINR queries.

The approach of such research has been to construct a data structure which approximates the underlying SINR diagram, and use it for answering approximate SINR queries, by performing point-location queries in this structure. That is, given a query point q, first find the sole candidate s_i that may be received at q (by searching in the appropriate Voronoi diagram), and then perform a point-location query to approximately determine whether q is in R_i or not. Two different notions of approximation have been used. In the first [5], it is guaranteed that the uncertain answer is only given infrequently, namely, the area of the uncertain region associated with R_i is at most $\varepsilon \cdot area(R_i)$, for a prespecified parameter $\varepsilon > 0$. In the second [10], it is guaranteed that for every point in the uncertain region the

SIN ratio is within an ε-neighborhood of β. See Table 1 for a summary of previous results; see also [11] for related work. In addition, Aronov and Katz [3] obtained several results for *batched* approximate SINR queries, using the latter notion of approximation; for example, one can perform n simultaneous approximate queries in a network with n transmitters at polylogarithmic amortized cost per query.

Given $\varepsilon > 0$,[4] an *approximate SINR query* is: Given a receiver q, find the sole transmitter s that may be received by q and return a value $\widetilde{sinr}(q, s)$, such that $(1 - \varepsilon)sinr(q, s) \leq \widetilde{sinr}(q, s) \leq (1 + \varepsilon)sinr(q, s)$. Thus, unless $(1 - \varepsilon)\beta \leq \widetilde{sinr}(q, s) < (1 + \varepsilon)\beta$, the value $\widetilde{sinr}(q, s)$ enables us to determine definitely whether or not s is received by q.

In this paper, we devise efficient algorithms for handling approximate SINR queries in a dynamic setting. That is, given S, α, β, and N, as above, and $\varepsilon > 0$, we describe algorithms for answering approximate SINR queries after some initial preprocessing, in a setting where transmitters may be added to or deleted from S. We analyze our algorithms by the usual measures, namely, preprocessing time, data structure size, query time, and update time.

To the best of our knowledge, these are the first data structures to support dynamic approximate SINR queries. In contrast with previous work on approximate SINR queries, our algorithms do not compute an approximation of the underlying SINR diagram. We distinguish between two main variants of the problem – the uniform power version and the non-uniform power one. The preprocessing time in both cases is $O(n \operatorname{polylog} n)$, while the update and query time is $O(\operatorname{polylog} n)$ for the uniform version, and $O(\sqrt{n} \operatorname{polylog} n)$ for the non-uniform version. Thus, our solution for the *dynamic* uniform version is comparable to the best known solutions for the *static* uniform version. For the non-uniform version, however, our solution is the first solution with bounds that depend only on n and ε and not on other parameters of the input, both in the static and dynamic settings.

In addition to the obvious motivation for devising algorithms for approximate SINR queries in a dynamic setting, we mention another important application of our results. *Successive Interference Cancellation (SIC)* is a technique that enables (in some circumstances) a receiver q to receive a specific transmitter t, even if t cannot be received at q in SINR sense. Informally, our results support SIC; if t's signal is the kth strongest at q, then, through a sequence of $O(k)$ queries and updates, we can determine whether q can decode t's signal from the combined signal; see the discussion in Section 4. In contrast, Avin et al. [4] construct a uniform-power static data structure of size $O(\varepsilon^{-1}n^{10})$ which enables one to determine in $O(\log n)$ time whether t can be received by q using SIC. Their result is not directly comparable to ours, however: They guarantee logarithmic query regardless of the number of transmitters that need to be canceled before t can be heard, and their approximation model is quite different from ours.

The missing proofs and some extensions can be found in the complete version of this paper [2].

2 Uniform power

Let q be a receiver and let s be the closest transmitter to q. Set $intrf(q) = \sum_{s' \in S \setminus \{s\}} \frac{1}{|qs'|^\alpha}$, then $sinr(q, s) = \frac{\frac{1}{|qs|^\alpha}}{intrf(q)}$.[5] When s is the transmitter closest to q, we will simply write

[4] For simplicity of presentation, we will assume hereafter that $n > 1/\varepsilon$.

[5] For clarity of presentation, we assume hereafter that there is no noise, i.e., $N = 0$. Our algorithms extend to the situation where noise is present in a straightforward manner.

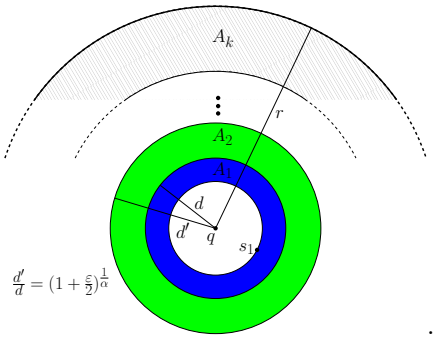


Figure 1 Partitioning $A_q(|qs_1|, r)$ into annuli.

$sinr(q)$ instead of $sinr(q, s)$. Fix $\varepsilon > 0$. We wish to compute a value $\widetilde{sinr}(q)$ such that $(1-\varepsilon)sinr(q) \leq \widetilde{sinr}(q) \leq sinr(q)$. We show below how to compute a value $\widetilde{intrf}(q)$ such that $intrf(q) \leq \widetilde{intrf}(q) \leq (1+\varepsilon)intrf(q)$ and then simply set $\widetilde{sinr}(q) = \frac{\frac{1}{|qs|^\alpha}}{\widetilde{intrf}(q)}$.

▶ **Claim 1.** *Under the above assumption,* $(1-\varepsilon)sinr(q) < \widetilde{sinr}(q) \leq sinr(q)$.

We start with a slower but easier to describe solution and then refine it.

2.1 Annuli

Let $\varepsilon > 0$. Let q be a receiver and let $s \in S$ be the closest transmitter to q. Let $s_1, \ldots, s_{n-1}$ be the transmitters in $S \setminus \{s\}$, and assume without loss of generality that s_1 is the second closest transmitter to q, among all the transmitters in S. Recall that $intrf(q) = \sum_{i=1}^{n-1} \frac{1}{|qs_i|^\alpha}$ and that we wish to compute a value $\widetilde{intrf}(q)$ such that $intrf(q) \leq \widetilde{intrf}(q) \leq (1+\varepsilon)intrf(q)$.

We will need the following simple observation.

▶ **Observation 2.** *$intrf(q)$ is the sum of $n-1$ positive terms of which $\frac{1}{|qs_1|^\alpha}$ is the largest, so we have $\frac{1}{|qs_1|^\alpha} \leq intrf(q) \leq \frac{n-1}{|qs_1|^\alpha}$.*

2.1.1 Query algorithm

Let q be a query point. First, we find s and s_1, the closest and the second closest transmitters to q, respectively. Next, we divide the transmitters in $S \setminus \{s\}$ into two subsets, S_c and S_f, where S_c consists of all transmitters that are 'close' to q and S_f consists of all transmitters that are 'far' from q. More precisely, set $r = (\frac{2n}{\varepsilon})^{1/\alpha} \cdot |qs_1|$, then S_c consists of all the transmitters in $S \setminus \{s\}$ whose distance from q is less than r, and S_f consists of all the remaining transmitters. We now approximate the contribution of each of these subsets to $intrf(q)$.

The contribution of a transmitter s_i in S_f to the sum $intrf(q)$ is

$$\frac{1}{|qs_i|^\alpha} \leq \frac{1}{r^\alpha} = \frac{\varepsilon}{2n|qs_1|^\alpha} \ ,$$

and the combined contribution of the transmitters in S_f is at most $|S_f| \cdot \frac{\varepsilon}{2n|qs_1|^\alpha} \leq \frac{\varepsilon}{2|qs_1|^\alpha}$.

We denote the annulus centered at q with inner radius r_1 and outer radius r_2 by $A_q(r_1, r_2)$. In order to approximate the overall contribution of the transmitters in S_c, we partition the annulus $A_q(|qs_1|, r)$ into k semi-open annuli, $A_1, \ldots, A_k$, such that the ratio of the outer to the inner radius of A_j is $(1+\frac{\varepsilon}{2})^{\frac{1}{\alpha}}$ (except for A_k whose ratio is at most $(1+\frac{\varepsilon}{2})^{\frac{1}{\alpha}}$); see

Figure 1. By semi-open we mean that the inner circle of A_j is contained in A_j, but the outer circle is not. Now, for each A_j, we approximate the contribution of each transmitter $s_i \in S_c \cap A_j$ by $\frac{1}{d^\alpha}$, where d is the inner radius of A_j, that is, we approximate the contribution of s_i by moving it to the inner circle of A_j. We prove below (Corollary 4) that this yields a $(1+\frac{\varepsilon}{2})$-approximation of the overall contribution of the transmitters in S_c to $intrf(q)$.

▶ **Lemma 3.** *Let $s_i \in S_c$ and let $A = A_q(d, d')$ be the annulus to which s_i belongs. Then, by moving s_i to the inner circle of A, one obtains a $(1+\frac{\varepsilon}{2})$-approximation of the contribution of s_i to $intrf(q)$.*

Proof. Since $s_i \in A$, $d \leq s_i < d'$. Moreover, by construction, $d'/d \leq (1+\frac{\varepsilon}{2})^{\frac{1}{\alpha}}$. So, the ratio of our approximation to the real contribution of s_i is

$$\frac{1/d^\alpha}{1/|qs_i|^\alpha} = \frac{|qs_i|^\alpha}{d^\alpha} < (\frac{d'}{d})^\alpha \leq 1 + \frac{\varepsilon}{2} \ . \qquad \blacktriangleleft$$

▶ **Corollary 4.** *By doing this for each transmitter in S_c, one obtains a $(1+\frac{\varepsilon}{2})$-approximation of the overall contribution of the transmitters in S_c to $intrf(q)$.*

It remains to show that $\widetilde{intrf}(q)$, which is the sum of the approximations for S_f and for S_c, satisfies the requirements, i.e., that $intrf(q) \leq \widetilde{intrf}(q) \leq (1+\varepsilon)intrf(q)$. From the description above it is clear that $\widetilde{intrf}(q) \geq intrf(q)$, so we only need to show that $\widetilde{intrf}(q) \leq (1+\varepsilon)intrf(q)$. Indeed,

$$\widetilde{intrf}(q) \leq \frac{\varepsilon}{2|qs_1|^\alpha} + (1+\frac{\varepsilon}{2}) \sum_{s_i \in S_C} \frac{1}{|qs_i|^\alpha} \leq \frac{\varepsilon}{2} intrf(q) + (1+\frac{\varepsilon}{2}) intrf(q) = (1+\varepsilon) intrf(q) \ ,$$

where the second inequality is based on Observation 2.

2.1.2 Implementation

We first show that k, the number of annuli into which the annulus $A_q(|qs_1|, r)$ is partitioned, is small; for the proof, see the complete version [2].

▶ **Lemma 5.** $k = O(\frac{1}{\varepsilon} \log n)$.

In the preprocessing stage we compute the following data structures for the set of transmitters S.

Dynamic nearest neighbor: A data structure due to Kaplan *et al.* [12] can be used for dynamic 2D nearest-neighbor queries. A set of points in the plane can be maintained dynamically in a linear-size data structure, so as to support insertions, deletions, and nearest-neighbor queries. Each insertion takes $O(\log^3 n)$ amortized deterministic time, each deletion takes $O(\log^5 n)$ amortized deterministic time, and each query takes $O(\log^2 n)$ worst-case deterministic time, where n is the size of the set of points at the time the operation is performed; see [12] and also the randomized data structure of Chan [6] with slightly worse (expected) performance.

Dynamic disk range counting: We start with the construction of Matoušek [13]: In linear space and $O(n \log n)$ time one can preprocess a set of n points in $\mathbb{R}^d$ to support semi-group halfspace range queries in $O(n^{1-1/d} \operatorname{polylog} n)$ time. A point can be deleted in $O(\log n)$ amortized time and inserted in $O(\log^2 n)$ amortized time. Lifting circles to points in $\mathbb{R}^3$ in

the standard manner, we obtain a linear-space, $O(n \log n)$ time, $O(n^{2/3} \operatorname{polylog} n)$ disk range counting query, $O(\log n)$ amortized delete, $O(\log^2 n)$ amortized insert data structure. We have not attempted to optimize this ingredient, as we replace this infrastructure with a more efficient one in the following section.

Given a query point q, we find s and s_1 (the closest and second closest transmitters) using the data structure for dynamic nearest neighbor; s_1 is found by deleting and reinserting s. Next, we compute the distance r and partition the annulus $A_q(|qs_1|, r)$ into k annuli, as described above. Now, we calculate $\widetilde{intrf}(q)$ as follows. We first compute the size of the set S_f by performing a disk counting query with the circle of radius r centered at q and subtracting the answer from $n-1$; we initialize $\widetilde{intrf}(q)$ to $|S_f| \cdot \frac{\varepsilon}{2n|qs_1|^\alpha}$. Next, for each of the k annuli, we compute the number x of points of S lying in it, as the difference in the numbers of points in the two disks defined by its bounding circles, obtained by counting queries. We then increment $\widetilde{intrf}(q)$ by $\frac{x}{d^\alpha}$, where d is the radius of the inner circle of the current annulus.

An update is performed by updating the two underlying data structures.

We omit the detailed performance analysis of this version, as a better data structure is described next.

2.2 Polygonal rings

We now present a more efficient solution similar to the previous one, except that we replace the circular annuli by polygonal rings. Set $x = (1 + \frac{\varepsilon}{2})^{\frac{1}{\alpha}}$, and consider any three circles C_0, C_1, C_2 centered at q, such that $r_1/r_0 = r_2/r_1 = \sqrt{x}$, where r_i is the radius of C_i. Set $l = \lceil \frac{\pi}{\sqrt{2 - \frac{2}{\sqrt{x}}}} \rceil$, and let B_i be the regular l-gon inscribed in C_i, so that one its vertices lies on the upward vertical ray through q, for $i = 1, 2$. We now show that C_{i-1} is contained in B_i, for $i = 1, 2$, and therefore, the polygonal ring defined by B_1 and B_2 is contained in the annulus $A_q(r_0, r_2)$. The elementary proofs of the following claims can be found in the complete version [2].

▶ **Claim 6.** *$l = O(1/\sqrt{\varepsilon})$.*

▶ **Claim 7.** *C_{i-1} is contained in B_i, for $i = 1, 2$.*

▶ **Corollary 8.** *The polygonal ring defined by B_1 and B_2 is contained in an annulus centered at q with radii ratio $x = (1 + \frac{\varepsilon}{2})^{\frac{1}{\alpha}}$.*

2.2.1 Query algorithm

We highlight the differences with the query algorithm from Section 2.1.1. Recall that we divided the transmitters into two subsets according to whether they were closer or farther than r from the query point q. We adjust the definitions slightly by setting $r' = |qs_1|x^{m/2}$, where m is the smallest integer for which $|qs_1|x^{m/2} \geq \sqrt{x}r$, and considering a transmitter close to q whenever it lies in the interior of the regular l-gon inscribed in the circle of radius r' centered at q, see Figure 2. The set of such transmitters is the new S_c; the remaining transmitters constitute S_f. The contribution of $s_i \in S_f$ to the sum $intrf(q)$ is, by Claim 7,

$$\frac{1}{|qs_i|^\alpha} \leq \frac{1}{r^\alpha} = \frac{\varepsilon}{2n|qs_1|^\alpha} \ .$$

Thus the overall contribution of the transmitters in S_f is again at most $\frac{\varepsilon}{2|qs|^\alpha}$.

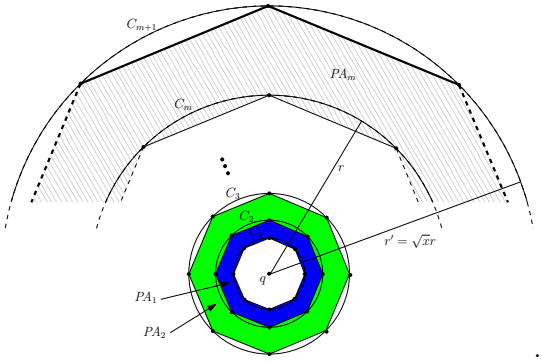


Figure 2 The polygonal rings $PA_1, \ldots, PA_m$. The ring PA_j lies between the circles C_{j-1} and C_{j+1}, for $j > 1$.

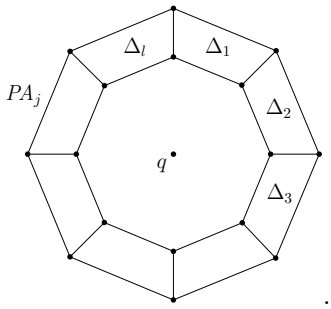


Figure 3 PA_j is the union of l trapezoids.

We now partition $A_q(|qs_1|, r')$ into m annuli, each with outer-to-inner radius ratio $\sqrt{x}$. For each of the $m+1$ circles defining these annuli, draw the regular l-gon inscribed in it. Let $PA_1, \ldots, PA_m$ be the resulting sequence of polygonal rings, numbered from the innermost outwards, see Figure 2; each ring is semi-open: it includes its inner, but not its outer boundary. By Claim 7, each PA_j, $j > 1$, is contained in the union of two consecutive annuli, which in turn is an annulus of ratio x; $S_c \cap PA_1$ is contained in the innermost annulus. Also notice that $m = O(k)$, where k is the number of annuli in the circular annulus version, so, by Lemma 5, $m = O(\frac{1}{\varepsilon} \log n)$.

For each ring PA_j, we bound from above the contribution of each $s_i \in S_c \cap PA_j$ by $1/d^\alpha$, where d is the inner radius of the annulus of ratio x containing PA_j. By Lemma 3 and the subsequent corollary, we obtain a $(1 + \frac{\varepsilon}{2})$-approximation of the overall contribution of the transmitters in S_c to $intrf(q)$; $\widetilde{intrf}(q)$ is obtained by combining the two estimates, one from S_c and one from S_f.

2.2.2 Implementation

Each polygonal ring PA_j is the union of l isosceles trapezoids; moreover the ith trapezoids of all rings are homothets of each other (refer to Figure 3) and therefore are delimited by lines of exactly three different orientations. In the preprocessing stage we compute the following data structures for the set of transmitters S.

Dynamic nearest neighbor: The data structure of Kaplan *et al.* [12] (see Section 2.1.2).

Dynamic trapezoid range counting: We use l instances of the data structure, one for each family of trapezoids. For the ith family we build a three-level orthogonal range counting structure, one for each of the three edge orientations of the trapezoids in the family. The answer to a trapezoid range counting query is the number of points of S lying in the trapezoid.

A standard three-level orthogonal range counting structure requires $O(n \log^2 n)$ space, is constructed in $O(n \log^2 n)$ time, and supports $O(\log^3 n)$-time range queries [8]. It can be modified to support insertions and deletions in $O(\log^3 n)$ amortized time using the standard partial-rebuilding technique [1, 14]. (One can use any of several different optimized variants of these structures [7, 15]. For example, He and Munro [9] describe one with linear space and $O((\log n/\log\log n)^2)$ worst-case query and amortized update time; we stay with comparison-based algorithms and do not attempt to optimize the polylogarithmic factors.)

Now given a query point q, we find its closest and second closest transmitter using the data structure for dynamic nearest neighbor in $O(\log^5 n)$ time, compute the distance r', and construct the (polygonal) rings $PA_1, \ldots, PA_m$, where $m = O(\frac{1}{\varepsilon}\log n)$. For each ring PA_j we proceed as follows. For each of the l trapezoids Δ_i forming PA_j, we perform an orthogonal range counting query in the ith data structure. Let n_j be the sum of the l results. Unless $j = 1$, we add to the value being computed the term $n_j \frac{1}{r_{j-1}^{\alpha}}$, where r_{j-1} is the radius of C_{j-1} (the inner circle of the annulus containing PA_j). If $j = 1$, we simply add the term $n_1 \frac{1}{r_1^{\alpha}}$. Finally, we add to the value being computed the term $|S_f| \cdot \frac{\varepsilon}{2n|qs_1|^{\alpha}} = (n - 1 - \sum_{j=1}^{m} n_j) \cdot \frac{\varepsilon}{2n|qs_1|^{\alpha}}$.

In summary, to implement an SINR query, we need to perform one search for the nearest and second-nearest neighbor, followed by $O(\frac{l}{\varepsilon}\log n) = O(\frac{1}{\varepsilon^{3/2}}\log n)$ range searches.

An update is applied to all the underlying data structures. The following theorem summarizes the main result of this section, while extensions can be found in the complete version [2].

▶ **Theorem 9.** *Given the locations of n uniform-power transmitters, one can preprocess them in $O((n/\sqrt{\varepsilon})\log^2 n)$ time and space into a data structure that can answer approximate SINR queries in $O(\log^5 n + (1/\varepsilon^{3/2})\log^4 n)$ time. Transmitters can be inserted in $O((1/\sqrt{\varepsilon})\log^3 n)$ and deleted in $O(\log^5 n + (1/\sqrt{\varepsilon})\log^3 n)$ amortized time.*

3 Non-uniform power

Let q be a receiver. For a transmitter $s \in S$, the *strength* of its signal at q is $nrg(s,q) = \frac{p(s)}{|qs|^{\alpha}}$ and the (multiplicatively-weighted) distance between q and s is $dist(q,s) = nrg(s,q)^{-1/\alpha} = \frac{1}{p(s)^{1/\alpha}} \cdot |qs|$. Let s be the closest transmitter to q according to $dist$. Set $intrf(q) = \sum_{s' \in S \setminus \{s\}} nrg(s', q)$, then $sinr(q,s) = \frac{nrg(s,q)}{intrf(q)}$, where we once again assume for clarity of presentation that there is no background noise, i.e., $N = 0$. When s is the closest transmitter to q, we will write $sinr(q)$ instead of $sinr(q,s)$.

Fix $\varepsilon > 0$. Again, we wish to approximate $sinr(q)$ by computing $\widetilde{intrf}(q)$ such that $intrf(q) \le \widetilde{intrf}(q) \le (1+\varepsilon)intrf(q)$ and setting $\widetilde{sinr}(q) = \frac{nrg(s,q)}{\widetilde{intrf}(q)}$. As in the uniform case, we start with a more straightforward but less efficient solution and then improve it.

3.1 Conical shells

Let q be a receiver and let $s \in S$ be the closest transmitter to q according to $dist$, i.e., the one whose signal strength at q is the highest. Let $s_1, \ldots, s_{n-1}$ be the transmitters in $S \setminus \{s\}$, and assume without loss of generality that s_1 is the second closest transmitter to q among

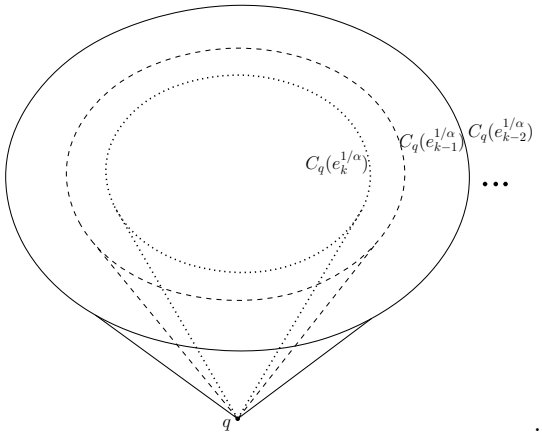


Figure 4 Partitioning $D_q(e_0^{1/\alpha}, e_k^{1/\alpha})$ into sub-shells.

the transmitters in S. Recall that $intrf(q) = \sum_{i=1}^{n-1} nrg(s_i, q)$ and that we wish to compute a value $\widetilde{intrf}(q)$ such that $intrf(q) \leq \widetilde{intrf}(q) \leq (1+\varepsilon) intrf(q)$. We will need the following simple observation.

▶ **Observation 10.** *$intrf(q)$ is the sum of $n-1$ positive terms of which $nrg(s_1, q)$ is the largest, so we have $nrg(s_1, q) \leq intrf(q) \leq (n-1) nrg(s_1, q) \leq n \cdot nrg(s_1, q)$.*

Query algorithm. Let q be a query point. First, we find s and s_1, as defined above. Next, we set $e_0 = \frac{\varepsilon}{2n} nrg(s_1, q)$, and divide the transmitters in $S \setminus \{s\}$ into two subsets, S_c and S_f, where S_c consists of the transmitters with signal strength at q greater than e_0, and S_f of the remaining ones. We now approximate the overall contribution to $intrf$ of the transmitters in S_f and in S_c separately and let $\widetilde{intrf}$ be the sum of the two approximations.

The contribution of a single transmitter $s_i \in S_f$ to the sum $intrf(q)$ is $nrg(s_i, q) \leq e_0 = \frac{\varepsilon}{2n} nrg(s_1, q)$, for a total of at most $|S_f| \cdot \frac{\varepsilon}{2n} nrg(s_1, q) \leq \frac{\varepsilon}{2} nrg(s_1, q)$ over all of S_f.

We identify the plane containing the transmitters and receivers with the xy-plane in $\mathbb{R}^3$. Let $C_q(\rho)$ denote (the surface of) the vertical cone with apex q whose z-coordinate at $t = (t_x, t_y)$ is $\rho|qt|$, where $\rho > 0$ is a constant. Let $D_q(\rho_1, \rho_2)$, $\rho_2 > \rho_1 > 0$, be the set of all points in 3-space lying above (i.e., in the interior of) the cone $C_q(\rho_1)$ and below or on (i.e., not in the interior of) the cone $C_q(\rho_2)$. Informally, $D_q(\rho_1, \rho_2)$ is the region between $C_q(\rho_1)$ and $C_q(\rho_2)$; we call it a *(conical) shell*.

Recall that $e_0 = \frac{\varepsilon}{2n} nrg(s_1, q)$. Let $e_i = (1 + \frac{\varepsilon}{2}) e_{i-1}$, for $i = 1, \ldots, k-1$, where $k-1$ is the largest integer for which $e_i < nrg(s_1, q)$, and set $e_k = nrg(s_1, q)$. We partition the range $I = (e_0, e_k]$ of signal strengths at q into k sub-ranges, $I_1 = (e_0, e_1], I_2 = (e_1, e_2], \ldots, I_k = (e_{k-1}, e_k]$, and count, for each sub-range I_j, the number of transmitters whose signal strength at q lies in I_j.

Consider a sub-range $I_j = (e_{j-1}, e_j]$; we want to count the number transmitters whose signal strength at q lies in I_j. This occurs whenever $e_{j-1}^{1/\alpha} |qs_i| < p_i^{1/\alpha} \leq e_j^{1/\alpha} |qs_i|$, or whenever the point $(s_i, p_i^{1/\alpha})$ in $\mathbb{R}^3$ lies in the shell $D_q(e_{j-1}^{1/\alpha}, e_j^{1/\alpha})$. Thus, we have reduced the problem to the difference of two conical range-counting queries.

We raise each of the transmitters $s_i \in S \setminus \{s\}$ to height $p_i^{1/\alpha}$, and preprocess the resulting set of points for conical range counting queries. If the number of points in the shell corresponding to I_j is x_j, then we add the term $e_j x_j$ to our approximation of $intrf(q)$, that is, we approximate the contribution of each transmitter s_i whose corresponding point lies in

the shell by e_j. (This corresponds to vertically projecting the point $(s_i, p_i^{1/\alpha})$ onto the cone $C_q(e_j^{1/\alpha})$.) We prove in the complete version [2] that this yields a $(1+\frac{\varepsilon}{2})$-approximation of the overall contribution of the transmitters in S_c to $intrf(q)$.

It remains to show that $\widetilde{intrf}(q)$, which is the sum of the approximations for S_f and for S_c, satisfies the requirements, i.e., that $intrf(q) \leq \widetilde{intrf}(q) \leq (1+\varepsilon)intrf(q)$. From the description above it is clear that $\widetilde{intrf}(q) \geq intrf(q)$, so we only need to establish the upper bound. Indeed, using Observation 10, we conclude that

$$\widetilde{intrf}(q) \leq \frac{\varepsilon}{2|qs_1|^\alpha} + (1+\frac{\varepsilon}{2}) \sum_{s_i \in S_C} nrg(s_i, q) \leq \frac{\varepsilon}{2} intrf(q) + (1+\frac{\varepsilon}{2}) intrf(q) = (1+\varepsilon) intrf(q) \ .$$

Implementation. A straightforward calculation shows (analogously to Lemma 5) that k, the number of shells into which $D_q(e_0^{1/\alpha}, e_k^{1/\alpha})$ is partitioned, is $O(\frac{1}{\varepsilon}\log n)$.

We preprocess the set of raised transmitters for cone range reporting/counting queries. Then, given a query point q, we find s and s_1 as follows. Pick a random sample T of $\sqrt{n}\log n$ transmitters and let $t_1 \in T$ be the transmitter whose signal strength at q is the strongest. With high probability, the number of transmitters in S that are closer to q than t_1 in terms of signal strength at q is $O(\sqrt{n})$, and we perform a range reporting query with the cone C_1 corresponding to t_1 in order to find them. The closest and second-closest points among the reported points are clearly s and s_1.

As for shell range counting queries, for each such query we issue two cone range counting queries – with the outer cone and the inner cone – and return the difference of the answers.

We omit the analysis of this version, since we describe below a more efficient variant, in which cones are replaced by pyramids.

3.2 Pyramidal shells

We now replace the conical shells by pyramidal ones to obtain an improved solution. Set $x = (1+\frac{\varepsilon}{2})^{\frac{1}{\alpha}}$, and consider any three cones $C_q(\rho_0)$, $C_q(\rho_1)$ and $C_q(\rho_2)$ with apex at q, such that $\rho_0/\rho_1 = \rho_1/\rho_2 = \sqrt{x}$. Let $P_q(\rho_i)$ be a regular l-pyramid inscribed in $C_q(\rho_i)$, where $l = \lceil \frac{\pi}{\sqrt{2-\frac{2}{\sqrt{x}}}} \rceil$. That is, $P_q(\rho_i)$'s apex is at q, its edges emanating from q are contained in (the surface of) $C_q(\rho_i)$, and the cross section of $P_q(\rho_i)$ and $C_q(\rho_i)$, using any horizontal cutting plane above q, is a regular l-gon and its circumcircle, respectively. The *pyramidal shell* defined by $P_q(\rho_2)$ and $P_q(\rho_1)$ and denoted $PS_q(\rho_2, \rho_1)$ is the semi-open region consisting of all points in the interior of $P_q(\rho_2)$ but not in the interior of $P_q(\rho_1)$. From Claim 7 and the observation above, it follows that $C_q(\rho_{i-1})$ is contained in $P_q(\rho_i)$, for $i = 1, 2$, and therefore, $PS_q(\rho_2, \rho_1)$ is contained in $D_q(\rho_2, \rho_0)$.

Query algorithm. We highlight the differences with the conical-shell based approach. First, we find s and s_1, the closest and the second-closest transmitters to q, respectively, as described in detail below. Previously, the transmitters were divided into two subsets lying close to q and lying far from it, with the threshold $e_0 = \frac{\varepsilon}{2n} \cdot nrg(s_1, q)$. Here, we set $e_0' = \frac{nrg(s_1,q)}{x^{m/2}}$, where m is the smallest integer for which $\frac{nrg(s_1,q)}{x^{m/2}} \leq \frac{e_0}{\sqrt{x}}$, and consider a transmitter close to q whenever it lies in the interior of the pyramid $P_q(e_0'^{\frac{1}{\alpha}})$, i.e., the pyramid inscribed in $C_q(e_0'^{\frac{1}{\alpha}})$. The contribution of a single transmitter $s_i \in S_f$ to the sum $intrf(q)$ is $nrg(s_i, q) \leq e_0 = \frac{\varepsilon}{2n} \cdot nrg(s_1, q)$, for a total of at most $\frac{\varepsilon}{2|qs|^\alpha}$, as before.

Consider the conical shell $D_q(e_0'^{\frac{1}{\alpha}}, nrg(s_1, q)^{\frac{1}{\alpha}})$ and partition it into m conical shells, such that the ratio between the parameters of the inner and outer cone of a shell is $\sqrt{x}$. For each

of the $m+1$ cones defining these conical shells, draw its inscribed regular l-pyramid. Let $P_q(\rho_1), ..., P_q(\rho_{m+1})$ be the resulting sequence of pyramids, where $P_q(\rho_1)$ is the innermost one, and consider the corresponding sequence of m nested pyramidal shells. Notice that $m = O(k)$, where k is the number of cones in the conical shells version, so once again $m = O(\frac{1}{\varepsilon}\log n)$. Moreover, each of the pyramidal shells, except for $PS_q(\rho_2, \rho_1)$, is contained in the union of two consecutive conical shells, which is a conical shell of ratio x. For $PS_q(\rho_2, \rho_1)$, we observe that $S_c \cap PS_q(\rho_2, \rho_1)$ is contained in the innermost conical shell.

We assign a transmitter s_i to a shell $PS_q(\rho_{j+1}, \rho_j)$ if s_i lands in the shell after being raised to height $p_i^{1/\alpha}$. Now, for each shell $PS_q(\rho_{j+1}, \rho_j)$ and each $s_i \in S_c$ assigned to it, we estimate the contribution of s_i from above by $1/\rho_{j-1}^{\alpha}$, i.e., by projecting s_i onto the inner cone of the conical shell $D_q(\rho_{j+1}, \rho_{j-1})$ containing $PS_q(\rho_{j+1}, \rho_j)$. We obtain a $(1+\frac{\varepsilon}{2})$-approximation of the overall contribution of the transmitters in S_c to $intrf(q)$, where the missing details can be found in the complete version [2]. Adding our previous estimate for those in S_f yields the promised $\widetilde{intrf}(q)$.

Implementation. Observe that each regular l-pyramid $P_q(\rho_j)$ is the union of l 3-sided wedges, where the ith wedge is defined by two planes of fixed orientation (perpendicular to the xy-plane) and a third plane containing the ith face of the pyramid.

In the preprocessing stage we construct l data structures over the set S, one for each family of wedges, supporting dynamic 3-dimensional 3-sided wedge range counting queries (a restricted form of simplex range counting in three dimensions). Each data structure handles wedges of the same "type"; the orientations of the two vertical bounding planes are fixed, while the orientation of the third plane varies (but remains perpendicular to the vertical plane bisecting the first two). The data structure for the ith family is a three-level search structure, where the first two levels allow us to represent the points of S that lie in the 2-sided wedge formed by the two vertical planes delimiting our 3-sided wedge, as a small collection of canonical subsets. For each canonical subset of the second level of the structure, we raise each of its points s_i to height $p_i^{1/\alpha}$ and then project it onto a vertical plane which is parallel to the bisector of the two vertical wedge boundaries. Finally, we construct for the resulting set of points a data structure for two-dimensional halfplane range counting queries. We will also need the corresponding reporting structure, see below.

Using standard tools for dynamic multilevel structures and, for example, Matoušek's data structure for halfplane range counting at the bottom level, we obtain a structure of $O(n \operatorname{polylog} n)$ size that supports wedge counting (and reporting) queries in $O(n^{1/2} \operatorname{polylog} n)$ time and updates in $O(\operatorname{polylog} n)$ amortized time.

Now, given a pyramidal shell $PS_q(\rho_{j+1}, \rho_j)$, we can count the number of raised points that lie in it as follows. We first perform l queries for the pyramid $P_q(\rho_{j+1})$, one in each of the l data structures, to obtain the total number of points that lie in it. We repeat the process for the pyramid $P_q(\rho_j)$ and finally subtract the latter number from the former one.

Below we describe how to find s and s_1, the closest and second-closest points to q, in randomized $O(\sqrt{n}\log n)$ time w.h.p. plus l wedge reporting queries. Once again, an update is performed by modifying the underlying data structures. We summarize the main result of this section.

▶ **Theorem 11.** *One can preprocess n arbitrary-power transmitters, in $O(\frac{n}{\sqrt{\varepsilon}} \operatorname{polylog} n)$ time and space, into a data structure that can answer approximate SINR queries in randomized $O(\frac{\sqrt{n}}{\sqrt{\varepsilon}} \operatorname{polylog} n)$ time w.h.p. and perform updates in $O(\frac{1}{\sqrt{\varepsilon}} \operatorname{polylog} n)$ amortized time.*

Finding the closest and second-closest transmitters to q. We have assumed that given a query point q, we can find the closest s and second-closest s_1 transmitters to q, efficiently. This section deals with this initial stage.

We begin by observing that we do not really need to find s_1, provided that we can obtain a sufficiently good approximation of $nrg_1 = nrg(s_1, q)$. Let $\widetilde{nrg}_1 \in \mathbb{R}$ such that $\widetilde{nrg}_1 \leq nrg_1 \leq (1+\delta)\widetilde{nrg}_1$, where $\delta > 0$ is a sufficiently small constant. Then, it is easy to modify our query algorithm so that it uses $\widetilde{nrg}_1$ instead of nrg_1. For simplicity of presentation, we refer in this paragraph to the algorithm using conical shells. Set $\tilde{e}_0 = \frac{\varepsilon}{2n}\widetilde{nrg}_1$. A transmitter will be considered close to q if and only if its signal strength at q is greater than $\tilde{e}_0$. If $s_i \in S$ is far from q, then $nrg(s_i, q) \leq \tilde{e}_0 \leq \frac{\varepsilon}{2n} nrg_1 = \frac{\varepsilon}{2n} nrg(s_1, q)$, so the overall contribution of the transmitters in S_f is bounded by $\frac{\varepsilon}{2} nrg(s_1, q)$, as before. Next, we partition the range $I = (\tilde{e}_0, (1+\delta)\widetilde{nrg}_1]$ into $k = O(\frac{1}{\varepsilon}\log n)$ sub-ranges, such that the ratio between the extreme values of a sub-range is at most $1 + \varepsilon/2$ and proceed exactly as before.

We now describe how to find s. Our algorithm may or may not find s_1. However, if it does not find s_1, it returns a transmitter $t_1 \in S$ such that $nrg(t_1, q) \leq nrg(s_1, q) \leq (1+\delta) nrg(t_1, q)$, where $\delta > 0$ is a sufficiently small constant, so we can set $\widetilde{nrg}_1 = nrg(t_1, q)$ and apply the above modified query algorithm.

Pick a random sample T of $\sqrt{n}\log n$ transmitters and let $t_1 \in T$ be the transmitter whose signal strength at q is the strongest. This can be done in $O(\sqrt{n}\log n)$ time. With high probability the number of transmitters in S that are closer to q than t_1, in terms of signal strength at q, is $O(\sqrt{n})$.

We first lift each transmitter $s = (s_x, s_y) \in S$ to the point $\hat{s} = (s_x, s_y, p(s)^{1/\alpha})$. Draw the cone C_1 corresponding to t_1, i.e., the cone whose z-coordinate above point s is $nrg(t_1, q)^{1/\alpha}|qs| = (p(t_1)^{1/\alpha}/|qt_1|)|qs|$. Let l be as above and consider the l-pyramid P_1 inscribed in C_1. Let C_0 be the cone inscribed in P_1, so that P_1 lies between C_0 and C_1. Notice that C_0 is the cone whose z-coordinate above point s is $(1+\delta) nrg(t_1, q)^{1/\alpha}|qs|$ (where we set $\delta = (1+\frac{\varepsilon}{2})^{\frac{1}{2\alpha}} - 1$).

Perform a range reporting query with P_1 (i.e., find all lifted points that lie in the interior of P_1 or on P_1). Since P_1 is inside C_1, with high probability the number of points in P_1 is $O(\sqrt{n})$. If the resulting set is non-empty, then in randomized $O(\sqrt{n})$ time w.h.p. we can find s and also s_1 (provided the number of returned points is greater than 1).

Otherwise, if P_1 is empty, we claim that the answer to the SINR query must be NO, i.e., q cannot receive any transmitter. Indeed, in the best scenario $\hat{s}$ lies on C_0, where $s \in S$ is the closest transmitter to q, and the rest of the $O(\sqrt{n})$ transmitters, lifted to 3-space, lie on the cone C_1. But this will imply that $sinr(s) < 1$. Indeed $nrg(s, q) = (1+\delta)^\alpha nrg(t_1, q)$ and, for any other of the $\sqrt{n}$ transmitters s', $nrg(s', q) = nrg(t_1, q)$, implying $sinr(s) < (1+\delta)^\alpha/\sqrt{n} \ll 1$.

If only one point lies in P_1, then we use t_1 as an approximation of s_1 as described above.

4 Successive interference cancellation (SIC)

Fix a receiver location q. SIC is a technique that enables q to receive a specific transmitter t, even when $sinr(q, t) < \beta$. More specifically, order the transmitters $s_1, \ldots, s_n$ in S by increasing signal strength at q, assume $t = s_k$, and let $sinr_i(q)$ denote the SIN ratio for the signal of s_i at q, while ignoring transmitters $s_1, \ldots, s_{i-1}$. If $sinr_1(q) = sinr(q, s_1) \geq \beta$, q can subtract s_1's signal from the combined signal. If, in addition, $sinr_2(q) \geq \beta$, q can also subtract s_2 from the combined signal of the transmitters $s_2, \ldots, s_n$, and so on. If $sinr_i(q) \geq \beta$, for $i = 1, \ldots, k$, we say that *SIC succeeds for s_k at q, in k rounds*. We can simulate this process using our data structures for approximate SINR queries via a sequence

of k queries and $k-1$ deletions, and determine (approximately) whether SIC succeeds for s_k at q. Observe that we need t only to terminate the query, while Avin et al. [4] need t to identify the part of the data structure in which to initiate the search; in particular, we can generate all the transmitters accessible via SIC given a location q in polylogarithmic time per transmitter, while they need to consult each of the n parts of the data structure. We obtain the following theorem.

▶ **Theorem 12.** *Assuming $t = s_k$, the simulation above can be performed in $O((1/\varepsilon^{3/2})k \operatorname{polylog} n)$ time in the uniform-power version, and in $O((1/\sqrt{\varepsilon})k\sqrt{n} \operatorname{polylog} n)$ time in the non-uniform version.*

References

1. Pankaj K. Agarwal. Range searching. In *Handbook of Discrete and Computational Geometry, Second Edition*, pages 809–837. CRC Press LLC, 2004. doi:10.1201/9781420035315.ch36.
2. Boris Aronov, Gali Bar-On, and Matthew J. Katz. Resolving SINR queries in a dynamic setting. arXiv:1804.10654.
3. Boris Aronov and Matthew J. Katz. Batched point location in SINR diagrams via algebraic tools. *ACM Transactions on Algorithms*, 2018. Accepted for publication; see also arXiv:1412.0962.
4. Chen Avin, Asaf Cohen, Yoram Haddad, Erez Kantor, Zvi Lotker, Merav Parter, and David Peleg. SINR diagram with interference cancellation. *Ad Hoc Networks*, 54:1–16, 2017. doi:10.1016/j.adhoc.2016.08.003.
5. Chen Avin, Yuval Emek, Erez Kantor, Zvi Lotker, David Peleg, and Liam Roditty. SINR diagrams: Convexity and its applications in wireless networks. *J. ACM*, 59(4):18:1–18:34, 2012. doi:10.1145/2339123.2339125.
6. Timothy M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *J. ACM*, 57(3):16:1–16:15, 2010. doi:10.1145/1706591.1706596.
7. B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988.
8. M. de Berg, O. Cheong, M. van Kreveld, and M. H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 3rd edition, 2008. URL: http://www.cs.ruu.nl/geobook/.
9. Meng He and J. Ian Munro. Space efficient data structures for dynamic orthogonal range counting. *Comput. Geom.*, 47(2):268–281, 2014. doi:10.1016/j.comgeo.2013.08.007.
10. Erez Kantor, Zvi Lotker, Merav Parter, and David Peleg. The topology of wireless communication. In *Proceedings 43rd ACM Symposium on Theory of Computing, STOC 2011*, pages 383–392, 2011. doi:10.1145/1993636.1993688.
11. Erez Kantor, Zvi Lotker, Merav Parter, and David Peleg. Nonuniform SINR+Voroni diagrams are effectively uniform. In *Proceedings 29th International Symposium on Distributed Computing, DISC 2015*, pages 588–601, 2015. doi:10.1007/978-3-662-48653-5_39.
12. Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. In *Proceedings 28th ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 2495–2504, 2017. See also arXiv:1604.03654. doi:10.1137/1.9781611974782.165.
13. J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.
14. M. H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes in Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1983.
15. D. E. Willard and G. S. Lueker. Adding range restriction capability to dynamic data structures. *J. ACM*, 32:597–617, 1985. doi:10.1145/3828.3839.

Uniform Mixed Equilibria in Network Congestion Games with Link Failures

Vittorio Bilò
Department of Mathematics and Physics, University of Salento, Lecce, Italy
vittorio.bilo@unisalento.it

Luca Moscardelli
Department of Economic Studies, University of Chieti-Pescara, Pescara, Italy
luca.moscardelli@unich.it

Cosimo Vinci
Department of Information Engineering Computer Science and Mathematics, University of L'Aquila, L'Aquila, Italy - Gran Sasso Science Institute, L'Aquila, Italy
cosimo.vinci@univaq.it

Abstract

Motivated by possible applications in fault-tolerant routing, we introduce the notion of uniform mixed equilibria in network congestion games with adversarial link failures, where players need to route traffic from a source to a destination node. Given an integer $\rho \geq 1$, a ρ-uniform mixed strategy is a mixed strategy in which a player plays exactly ρ edge disjoint paths with uniform probabilities, so that a ρ-uniform mixed equilibrium is a tuple of ρ-uniform mixed strategies, one for each player, in which no player can lower her cost by deviating to another ρ-uniform mixed strategy. For games with weighted players and affine latency functions, we show existence of ρ-uniform mixed equilibria and provide a tight characterization of their price of anarchy. For games with unweighted players, instead, we extend the existential guarantee to any class of latency functions and, restricted to games with affine latencies, we derive a tight characterization of both the prices of anarchy and stability.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory, Theory of computation → Quality of equilibria, Theory of computation → Network games

Keywords and phrases Network Congestion Games, Fault-Tolerant Routing, Nash Equilibria, Price of Anarchy, Price of Stability

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.146

Related Version A full version of the paper is available at `http://dx.doi.org/10.13140/RG.2.2.26479.74403`.

1 Introduction

Consider n users who need to send an unsplittable amount of traffic from a source to a destination in a network which is subject to adversarial link failures. In particular, each user u_i is coupled with an adversary a_i who, upon knowledge of the mixed strategy adopted by u_i to route her traffic, chooses which links to corrupt. Thus, every pair (u_i, a_i) is engaged in a two-player Stackelberg game in which u_i is the leader, a_i is the follower, and both are interested in the probability that the path selected by u_i as a realization of her mixed strategy fails: u_i wants to minimize it, while a_i aims at its maximization[1]. To make things more

[1] We stress that a_i is only aware of the mixed strategy chosen by u_i and not of its final realization.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 146; pp. 146:1–146:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

interesting, all users also play an atomic congestion game among themselves, in which each of them wants to minimize the expected latency of the chosen mixed strategy. We assume that a user's priority is to route her traffic at any cost, so that she will be interested in selecting the mixed strategy of minimum expected latency among those minimizing the failure probability (of its realization). It is not difficult to see that u_i minimizes this probability if and only if she assigns uniform probabilities to the maximum number of edge disjoint paths connecting her source-destination pair.

Motivated by the above scenario, we introduce the notion of uniform mixed equilibrium for (network) congestion games (with adversarial link failures). Formally, given an integer $\rho \geq 1$, a ρ-uniform mixed strategy is a mixed strategy in which exactly ρ edge disjoint paths are chosen with uniform probabilities. Thus, given an n-tuple of positive integers $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_n)$, a $\boldsymbol{\rho}$-uniform mixed profile is a mixed profile in which each user u_i adopts a ρ_i-uniform mixed strategy and a $\boldsymbol{\rho}$-uniform mixed equilibrium is a $\boldsymbol{\rho}$-uniform mixed profile in which no user u_i can lower her cost by deviating to another ρ_i-uniform mixed strategy.

As a first step in the understanding of the properties of these equilibria, we assume that $\rho_i = \rho_j := \rho$ for every pair of users u_i and u_j, and we denote $\boldsymbol{\rho} = (\rho, \ldots, \rho)$ simply as ρ. Besides defining a simple, yet interesting case, this assumption has at least two practical applications/justifications. First, it models the case of symmetric games, in which all users share the same source-destination pair; this setting has been widely studied with respect to the analysis of efficiency of Nash equilibria [21] and to the (hardness of) computation of equilibria [17, 16]. From a theoretical point of view, it is worth noting that, in this case, the value of ρ, i.e. the maximum number of edge disjoint paths connecting the common source-destination pair, can be efficiently computed by a reduction to the max-flow problem (see [1] for further details). To illustrate the second application, observe that the desire to minimize the failure probability induces each user to add even extremely costly paths to the set of her possible alternatives. It is reasonable to assume that, in some contexts, users' priorities can be restated as follows: each user wants to select the mixed strategy of minimum expected latency among those keeping the failure probability within a certain threshold θ. For the ease of exposition, assume that the adversary can corrupt just one link and that $\theta = 1/3$. By simple calculations, it is not difficult to establish that the best strategy for each user is to play $\rho = 3$ edge disjoint paths with probability $1/3$ each.

We stress that, although the notions of 1-uniform mixed equilibria and that of pure Nash equilibria [26] coincide, there are no correlations between the set of ρ-uniform mixed equilibria and that of mixed Nash equilibria of a given game when $\rho > 1$. Moreover, for $\rho > 1$, ρ-uniform mixed strategies can be interpreted as an hybridization between the notions of pure and mixed strategies. In fact, although the cost incurred by a player needs to be evaluated in expectation (as it happens when adopting mixed strategies), the fact that probabilities are superimposed by the model limits the players' choices to deciding which strategies to play (as it happens when adopting pure strategies). To the best of our knowledge, this is the first attempt towards this direction.

1.1 Our Contribution

We study the existence and efficiency of ρ-uniform mixed equilibria in (network) congestion games by distinguish between the case in which all players need to route the same amount of traffic (*unweighted congestion games*) and the general case of different traffic rates (*weighted congestion games*). In particular, we focus on networks in which the link latency functions are affine (*affine congestion games*).

Table 1 The prices of anarchy and stability of ρ-uniform mixed equilibria in affine unweighted congestion games and the price of anarchy of ρ-uniform mixed equilibria in affine weighted congestion games, for each value of $\rho \geq 1$. Bounds labeled as $*$ holds also for parallel link networks with restricted strategies, while bounds labeled as † applies to even parallel link networks with unrestricted strategies.

ρ	unweighted games		weighted games
	price of stability	price of anarchy	price of anarchy
1	$1+1/\sqrt{3}$ [11, 13]	$5/2^*$ [3, 11, 14]	$(3+\sqrt{5})/2^\dagger$[3, 5, 11, 14]
2	$1+1/\sqrt{5}$	$5/3^*$	$2^\dagger$
3	$1+1/\sqrt{7}$	$(2\sqrt{7}-1)/3^*$	$(7+\sqrt{13})/6^\dagger$
4	$4/3^\dagger$	$4/3^\dagger$	$(9+\sqrt{17})/8^\dagger$
≥ 5	$4/3^\dagger$	$4/3^\dagger$	$4\rho^2/(3\rho^2-2\rho-1)^\dagger$

We first prove that ρ-uniform mixed equilibria do exist in affine weighted congestion games, for each $\rho \geq 1$. This is done by showing that any affine weighted congestion game in which players adopt ρ-uniform mixed strategies admits a potential function. This generalizes to every value of ρ the results by [21, 22, 27] which were proved for the classical setting in which players adopt pure strategies, i.e., for $\rho = 1$. For the case of unweighted players, existential guarantees are extended to any class of latency functions. This generalizes Rosenthal's Theorem [30] which shows existence of pure Nash equilibria, i.e., existence of ρ-uniform mixed equilibria for the basic case of $\rho = 1$.

Then, for each $\rho \geq 1$, by exploiting the primal-dual method [6], we derive tight bounds on the price of anarchy of ρ-uniform mixed equilibria in affine weighted congestion games and tight bounds on both the prices of anarchy and stability of ρ-uniform mixed equilibria in affine unweighted congestion games (see the values reported in Table 1, where many lower bounds hold even for parallel link networks). It is worth noticing that our results nicely extend the ones obtained for pure Nash equilibria, i.e., the case of $\rho = 1$. In particular, for unweighted congestion games with affine latency functions, [13] proved that the price of stability is lower bounded by $1+1/\sqrt{3}$, while [11] showed that this bound is tight; [3, 14] proved that the price of anarchy is $5/2$ and [11] showed that the same (lower) bound extends to the special case of parallel link networks with restricted strategies (i.e., every player can only select a link from an allowable set of alternatives). For weighted congestion games with linear latency functions, [3, 14] proved that the price of anarchy is $(3+\sqrt{5})/2$, [11] showed that the same (lower) bound extends to the special case of parallel link networks with restricted strategies, and finally [5] proved that even parallel link networks with unrestricted strategies are enough to obtain a matching lower bound. The existential guarantee, as well as the bounds for unweighted games, are obtained by exploiting the fact that, for each $\rho \geq 1$, any unweighted congestion game in which players adopt ρ-uniform mixed strategies is isomorphic to an unweighted congestion game in which players adopt pure strategies and whose latency functions are slightly different.

Our results show that, as ρ increases, the prices of anarchy and stability of ρ-uniform mixed equilibria of affine congestion games approach the value $4/3$, that is, the price of anarchy of affine non-atomic congestion games [32]. This is in accordance with the intuition that, by arbitrarily splitting an atomic request over disjoint strategies, atomic congestion games tend to their non-atomic counterparts. The striking evidence of our findings, however, is that, for unweighted players, when such a splitting is restricted to be uniform (i.e., the same amount of traffic must be routed on each selected path), this happens for $\rho = 4$ already.

1.2 Related Work

Penn, Polukarov and Tennenholtz [28, 29] introduced congestion games with failures. In their model, each player has a task that can be executed on any resource, i.e. players only adopt singleton strategies, and each resource may fail with a certain probability, hence, for reliability reasons, a player may choose to simultaneously use multiple resources. The cost of a player is given by the minimum of the costs payable on all the selected resources that do not fail. In this setting, the existence, properties and efficiency of pure Nash equilibria are investigated.

The setting of adversarial behavior in congestion games was introduced by Karakostas and Viglas [23] for network congestion games. Babaioff, Kleinberg and Papadimitriou [4] studied the impact of malicious players on the quality of Nash equilibria for non-atomic games. In particular, [4, 23] considered two classes of players, i.e., rational players and malicious players; while rational players act aiming at minimizing their own cost, malicious ones aim at maximizing the average delay experienced by the rational players. Roth [31] applied this adversarial setting to the class of linear congestion games. Also Moscibroda et al. [25] analyzed an adversarial behavior in a different game.

1.3 Paper Organization

The paper is organized as follows. In the next section we provide the notation and definitions, together with some basic results. Section 3 is devoted to the study of affine weighted congestion games, while Section 4 to the analysis of the unweighted case. Finally, Section 5 gives some conclusive remarks and lists some interesting open problems. Due to space limitations, some proofs are omitted (see the full version of the paper).

2 Definitions and Notation

Given two integers $0 \leq k_1 \leq k_2$, let $[k_2]_{k_1}$ denote the set $\{k_1, k_1+1, \ldots, k_2-1, k_2\}$ and let $[k_1]$ denote the set $[k_1]_1$. Moreover, let $\vec{1}^n$ denote the vector $(1, \ldots, 1) \in \mathbb{R}^n_{\geq 0}$.

A *weighted congestion model* is defined by a tuple $\mathsf{CM} = (\mathsf{N}, E, (\ell_e)_{e\in E}, (w_i)_{i\in\mathsf{N}}, (\Sigma_i)_{i\in\mathsf{N}})$, where N is a set of $n \geq 2$ players, E is a set of resources, $\ell_e : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is the latency function of resource $e \in E$, and, for each $i \in \mathsf{N}$, $w_i \geq 0$ is the weight of player i and $\Sigma_i \subseteq 2^E \setminus \emptyset$ is her set of strategies. A weighted congestion model is *symmetric* if $\Sigma_i = \Sigma$ for each $i \in \mathsf{N}$, i.e., if all players share the same strategy space. A *weighted load balancing model* is a weighted congestion model in which for each $i \in \mathsf{N}$ and $S \in \Sigma_i$, $|S| = 1$, that is, all players' strategies are singleton sets. Observe that a weighted load balancing model corresponds to a parallel link network. A weighted congestion model is *affine* if its latency functions are of the form $\ell_e(x) := \alpha_e x + \beta_e$, with $\alpha_e, \beta_e \geq 0$. An *unweighted congestion model* is a weighted congestion model such that $w_i = 1$ for each $i \in \mathsf{N}$.

Depending on the types of strategies adopted by the players, a congestion model CM may induce different classes of congestion games.

A *strategy profile* is an n-tuple of strategies $\boldsymbol{s} = (s_1, s_2, \ldots, s_n)$, that is a state in which each player $i \in \mathsf{N}$ adopts pure strategy $s_i \in \Sigma_i$. When players adopt pure strategies, CM induces a congestion game $\mathsf{CG}(\mathsf{CM})$ (usually, when CM is clear from the context, we shall drop it from the notation). For a strategy profile $\boldsymbol{s}$, the *congestion* of resource $e \in E$ in $\boldsymbol{s}$, denoted as $k_e(\boldsymbol{s}) := \sum_{i\in\mathsf{N}: e\in s_i} w_i$, is the total weight of the players using resource e in $\boldsymbol{s}$, (observe that, for unweighted games, $k_e(\boldsymbol{s})$ coincides with the number of users selecting resource e in $\boldsymbol{s}$). The

cost of player i in $\boldsymbol{s}$ is defined as $cost_i^{\mathsf{CG}}(\boldsymbol{s}) = \sum_{e \in s_i} \ell_e(k_e(\boldsymbol{s}))$ (usually, when CG is clear from the context, we shall drop it from the notation). The quality of a strategy profile in $\mathsf{CG}(\mathsf{CM})$ is measured by using the *social function* $\mathsf{SUM}(\boldsymbol{s}) = \sum_{i \in \mathsf{N}} w_i cost_i(\boldsymbol{s}) = \sum_{e \in E} k_e(\boldsymbol{s})\ell_e(k_e(\boldsymbol{s}))$, that is, the weighted sum of the players' costs. A *pure Nash equilibrium* for $\mathsf{CG}(\mathsf{CM})$ is a strategy profile $\boldsymbol{s}$ such that, for any player $i \in \mathsf{N}$ and strategy $s_i' \in \Sigma_i$, $cost_i(\boldsymbol{s}) \leq cost_i(\boldsymbol{s}_{-i}, s_i')$. We denote by $\mathsf{Eq}(\mathsf{CG}(\mathsf{CM}))$ the set of pure Nash equilibria of a weighted congestion game $\mathsf{CG}(\mathsf{CM})$. The *price of anarchy* (resp. *price of stability*) of a weighted congestion game $\mathsf{CG}(\mathsf{CM})$ is defined as $\mathsf{PoA}(\mathsf{CG}(\mathsf{CM})) = \max_{\boldsymbol{s} \in \mathsf{Eq}(\mathsf{CG}(\mathsf{CM}))} \left\{ \frac{\mathsf{SUM}(\boldsymbol{s})}{\mathsf{SUM}(\boldsymbol{s}^*)} \right\}$ (resp. $\mathsf{PoS}(\mathsf{CG}(\mathsf{CM})) = \min_{\boldsymbol{s} \in \mathsf{Eq}(\mathsf{CG}(\mathsf{CM}))} \left\{ \frac{\mathsf{SUM}(\boldsymbol{s})}{\mathsf{SUM}(\boldsymbol{s}^*)} \right\}$), where $\boldsymbol{s}^*$ is a *social optimum* for $\mathsf{CG}(\mathsf{CM})$, that is a strategy profile minimizing the social function.

A mixed strategy for player i is a probability distribution σ_i defined over Σ_i, so that $\sigma_i(s)$ is the probability that player i plays strategy $s \in \Sigma_i$. We denote by $\mathsf{supp}(\sigma_i) = \{s \in \Sigma_i : \sigma_i(s) > 0\}$ the set of strategies played with positive probability in σ_i. A *mixed profile* $\boldsymbol{\sigma}$ is an n-tuple of mixed strategies, i.e., $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$. Informally, $\boldsymbol{\sigma}$ is a state in which each player $i \in \mathsf{N}$ picks her strategy according to probability distribution σ_i, independently from the choices of other players. If σ_i is such that a pure strategy s_i is picked with probability one by player i, we write s_i instead of σ_i.

Given an integer $\rho \geq 1$ and a weighted congestion model CM in which for each player $i \in N$ there exist at least ρ pairwise disjoint strategies in Σ_i, a *ρ-uniform mixed strategy* for player i is a mixed strategy σ_i such that $|\mathsf{supp}(\sigma_i)| = \rho$, $s_1 \cap s_2 = \emptyset$ for any $s_1, s_2 \in \mathsf{supp}(\sigma_i)$ with $s_1 \neq s_2$, and $\sigma_i(s) = 1/\rho$ for each $s \in \mathsf{supp}(\sigma_i)$, i.e., a mixed strategy in which player i plays exactly ρ pairwise disjoint strategies with uniform probability. Denote by $\Delta_i^{\rho}(\mathsf{CM})$ the set of ρ-uniform mixed strategies for player i. A *ρ-uniform mixed profile* $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ is an n-tuple of ρ-uniform mixed strategies, one for each player. When players adopt *ρ-uniform mixed strategies*, CM induces a ρ-uniform congestion game ρ-$\mathsf{CG}(\mathsf{CM})$ (again, when CM is clear from the context, we shall drop it from the notation). For a ρ-uniform mixed profile $\boldsymbol{\sigma}$, the *expected congestion* of resource $e \in E$ in $\boldsymbol{\sigma}$, denoted as $k_e(\boldsymbol{\sigma}) := \mathbb{E}_{\boldsymbol{s} \sim \boldsymbol{\sigma}} \left(\sum_{i \in \mathsf{N} : e \in s_i} w_i \right)$, is the expected total weight of the players using resource e in $\boldsymbol{\sigma}$. The cost of player i in $\boldsymbol{\sigma}$ is defined as $cost_i^{\rho\text{-}\mathsf{CG}}(\boldsymbol{\sigma}) = \mathbb{E}_{\boldsymbol{s} \sim \boldsymbol{\sigma}} \left(\sum_{e \in s_i} \ell_e(k_e(\boldsymbol{s})) \right)$ (again, when ρ-CG is clear from the context, we shall drop it from the notation). The quality of a ρ-uniform mixed profile in ρ-$\mathsf{CG}(\mathsf{CM})$ becomes $\mathsf{SUM}(\boldsymbol{\sigma}) = \mathbb{E}_{\boldsymbol{s} \sim \boldsymbol{\sigma}} \left(\sum_{i \in \mathsf{N}} w_i cost_i(\boldsymbol{s}) \right) = \sum_{e \in E} \mathbb{E}_{\boldsymbol{s} \sim \boldsymbol{\sigma}} \left(k_e(\boldsymbol{s})\ell_e(k_e(\boldsymbol{s})) \right)$, that is, the expected weighted sum of the players' costs. A *ρ-uniform mixed equilibrium* for ρ-$\mathsf{CG}(\mathsf{CM})$ is a ρ-uniform mixed profile $\boldsymbol{\sigma}$ such that, for any player $i \in \mathsf{N}$ and ρ-uniform mixed strategy $\sigma_i' \in \Delta_i^{\rho}(\mathsf{CM})$, $cost_i(\boldsymbol{\sigma}) \leq cost_i(\boldsymbol{\sigma}_{-i}, \sigma_i')$. We denote by $\mathsf{Eq}(\rho\text{-}\mathsf{CG}(\mathsf{CM}))$ the set of ρ-uniform mixed equilibria of a weighted congestion game ρ-$\mathsf{CG}(\mathsf{CM})$. The price of anarchy (resp. price of stability) of a ρ-uniform weighted congestion game ρ-$\mathsf{CG}(\mathsf{CM})$ is defined as $\mathsf{PoA}_{\rho}(\rho\text{-}\mathsf{CG}(\mathsf{CM})) = \max_{\boldsymbol{\sigma} \in \mathsf{Eq}(\rho\text{-}\mathsf{CG}(\mathsf{CM}))} \left\{ \frac{\mathsf{SUM}(\boldsymbol{\sigma})}{\mathsf{SUM}(\boldsymbol{\sigma}^*)} \right\}$ (resp. $\mathsf{PoS}_{\rho}(\rho\text{-}\mathsf{CG}(\mathsf{CM})) = \min_{\boldsymbol{\sigma} \in \mathsf{Eq}(\rho\text{-}\mathsf{CG}(\mathsf{CM})} \left\{ \frac{\mathsf{SUM}(\boldsymbol{\sigma})}{\mathsf{SUM}(\boldsymbol{\sigma}^*)} \right\}$), where $\boldsymbol{\sigma}^*$ is a *ρ-uniform social optimum* for ρ-$\mathsf{CG}(\mathsf{CM})$, that is a ρ-uniform mixed profile minimizing the social function.

Given a ρ-uniform mixed strategy σ_i, let $E(\sigma_i) = \bigcup_{s \in \mathsf{supp}(\sigma_i)} s$ denote the set of resources contained by all strategies belonging to $\mathsf{supp}(\sigma_i)$[2]. For a ρ-uniform mixed profile $\boldsymbol{\sigma}$, the *ρ-maximum congestion* of resource e in $\boldsymbol{\sigma}$, denoted as $k_{\rho,e}(\boldsymbol{\sigma}) = \sum_{i : e \in E(\sigma_i)} w_i$, is the congestion of e obtained if all players assigning non-null probability to a strategy s containing e pick s.

[2] Given $e \in E(\sigma_i)$, there exists a unique strategy of σ_i containing e, since strategies selected with non-null probability by each player are pairwise disjoint.

▶ Remark. According to the first application described in Section 1, given a symmetric congestion model CM such that the maximum number of disjoint strategies is $\rho* > \rho$, we can consider a congestion model CM' such that $\mathsf{CG}(\mathsf{CM}')$ is equivalent to $\mathsf{CG}(\mathsf{CM})$, and such that the maximum number of disjoint strategies of $\mathsf{CG}(\mathsf{CM}')$ is ρ. To this aim, it suffices considering a congestion model CM' in which $E' := E \cup \{e'_1, e'_2, \ldots, e'_\rho\}$, where e'_j is a dummy resource with $\ell'(e'_j) = 0$ for any $j \in [\rho]$, $\Sigma_i{}' := \{s \cup \{e'_j\} : s \in \Sigma_i, j \in [\rho]\}$ for any $i \in \mathsf{N}$, and all the other quantities are defined as in CM. Observe that, given ρ-disjoint strategies $s_1, s_2, \ldots, s_\rho$ in Σ, we have that $s_1 \cup \{e'_1\}, s_2 \cup \{e'_2\}, \ldots, s_\rho \cup \{e'_\rho\}$ are disjoint strategies of $\Sigma_i{}'$. Furthermore, there are no $\rho + 1$ disjoint strategies in $\Sigma_i{}'$, since, given $\rho + 1$ strategies of $\Sigma_i{}'$, there are necessarily at least two strategies $s'_1, s'_2 \in \Sigma_i{}'$ such that $e'_j \in s'_1 \cap s'_2$ for some $j \in [\rho]$. Thus, ρ is the maximum number of disjoint strategies in $\mathsf{CG}(\mathsf{CM}')$. Finally, since each strategy of $\Sigma_i{}'$ is defined as union of some strategy of Σ_i and some dummy resource having null cost, games $\mathsf{CG}(\mathsf{CM}')$ and $\mathsf{CG}(\mathsf{CM})$ are completely equivalent.

We conclude the section by providing useful equations to express the players' costs in ρ-uniform congestion games as a function of the ρ-maximum congestions only, thus getting rid of expected values. Towards this end, as shown in [6], we can assume without loss of generality that the latency functions of the games we consider are restricted to be linear, that is, of the form $\ell(x) = \alpha_e x$ for some $\alpha_e \geq 0$.

▶ **Lemma 1.** *Given an affine weighted congestion model* CM *and a* ρ*-uniform strategy profile* $\boldsymbol{\sigma}$ *for* ρ-$\mathsf{CG}(\mathsf{CM})$*, we have*

$$cost_i(\boldsymbol{\sigma}) = \sum_{e \in E(\sigma_i)} \alpha_e \left(\frac{k_{\rho,e}(\boldsymbol{\sigma})}{\rho^2} + \left(\frac{\rho - 1}{\rho^2} \right) w_i \right) \tag{1}$$

and, given $\sigma'_i \in \Delta_i^\rho(\mathsf{CM})$*, we have*

$$\begin{aligned} cost_i(\boldsymbol{\sigma}_{-i}, \sigma'_i) &= \sum_{e \in E(\sigma'_i) \setminus E(\sigma_i)} \alpha_e \left(\frac{k_{\rho,e}(\boldsymbol{\sigma})}{\rho^2} + \frac{w_i}{\rho} \right) + \sum_{e \in E(\sigma'_i) \cap E(\sigma_i)} \alpha_e \left(\frac{k_{\rho,e}(\boldsymbol{\sigma})}{\rho^2} + \left(\frac{\rho - 1}{\rho^2} \right) w_i \right) \\ &\leq \sum_{e \in E(\sigma'_i)} \alpha_e \left(\frac{k_{\rho,e}(\boldsymbol{\sigma})}{\rho^2} + \frac{w_i}{\rho} \right). \end{aligned} \tag{2}$$

3 Weighted Games

In this section, we consider the general case of ρ-uniform congestion games induced by affine weighted congestion models. We start by showing that ρ-uniform mixed equilibria are always guaranteed to exist, for each $\rho \geq 1$. In particular, by resorting to a potential function argument, we prove that, for each affine weighted congestion model CM, any better-response dynamics in ρ-$\mathsf{CG}(\mathsf{CM})$ converges to a ρ-uniform mixed equilibrium after a finite number of steps.

▶ **Theorem 2.** *For each affine weighted congestion model* CM *and* $\rho \geq 1$*,* ρ-$\mathsf{CG}(\mathsf{CM})$ *admits a potential function.*

Proof. Given an affine weighted congestion model CM and an integer $\rho \geq 1$, consider the function Φ_ρ defined on the set of ρ-uniform mixed profiles for ρ-$\mathsf{CG}(\mathsf{CM})$:

$$\Phi_\rho(\boldsymbol{\sigma}) := \sum_{e \in E} \alpha_e \left(\frac{k_{\rho,e}(\boldsymbol{\sigma})^2}{2\rho^2} + \frac{2\rho - 1}{2\rho^2} \sum_{i : e \in E(\sigma_i)} w_i^2 \right). \tag{3}$$

We prove that Φ_ρ is a weighted potential function for ρ-CG. Fix a ρ-uniform mixed profile $\boldsymbol{\sigma}$, a player $i \in \mathsf{N}$, and a ρ-uniform mixed strategy $\sigma'_i \in \Delta_i^\rho(\mathsf{CM})$. Let $\mathbb{I}_e = 1$ if $e \in \sigma'_i \setminus \sigma_i$, $\mathbb{I}_e = -1$ if $e \in \sigma_i \setminus \sigma'_i$, $\mathbb{I}_e = 0$ if $e \in \sigma'_i \cap \sigma_i$. We have

$$
\begin{aligned}
&\Phi_\rho(\boldsymbol{\sigma}_{-i}, \sigma'_i) - \Phi_\rho(\boldsymbol{\sigma}) \\
&= \sum_{e \in E} \alpha_e \left(\frac{(k_{\rho,e}(\boldsymbol{\sigma}) + \mathbb{I}_e w_i)^2}{2\rho^2} + \frac{2\rho - 1}{2\rho^2} \left(\sum_{j : e \in E(\sigma_j)} w_j^2 + \mathbb{I}_e w_i^2 \right) \right) \\
&\quad - \sum_{e \in E} \alpha_e \left(\frac{k_{\rho,e}(\boldsymbol{\sigma})^2}{2\rho^2} + \frac{2\rho - 1}{2\rho^2} \sum_{j : e \in E(\sigma_j)} w_j^2 \right) \\
&= \sum_{e \in E(\sigma'_i) \setminus E(\sigma_i)} \alpha_e \left(\frac{(k_{\rho,e}(\boldsymbol{\sigma}) + w_i)^2 - k_{\rho,e}(\boldsymbol{\sigma})^2}{2\rho^2} + \frac{2\rho - 1}{2\rho^2} w_i^2 \right) \\
&\quad - \sum_{e \in E(\sigma_i) \setminus E(\sigma'_i)} \alpha_e \left(\frac{k_{\rho,e}(\boldsymbol{\sigma})^2 - (k_{\rho,e}(\boldsymbol{\sigma}) - w_i)^2}{2\rho^2} + \frac{2\rho - 1}{2\rho^2} w_i^2 \right) \\
&= \sum_{e \in E(\sigma'_i) \setminus E(\sigma_i)} \alpha_e w_i \left(\frac{k_{\rho,e}(\boldsymbol{\sigma})}{\rho^2} + \frac{1}{\rho} w_i \right) - \sum_{e \in E(\sigma_i) \setminus E(\sigma'_i)} \alpha_e w_i \left(\frac{k_{\rho,e}(\boldsymbol{\sigma})}{\rho^2} + \frac{\rho - 1}{\rho^2} w_i \right) \\
&= w_i(cost_i(\boldsymbol{\sigma}_{-i}, \sigma'_i) - cost_i(\boldsymbol{\sigma})), \qquad (4)
\end{aligned}
$$

where (4) comes from Lemma 1. Thus, Φ_ρ is a weighted potential function for ρ-CG(CM). ◀

3.1 Price of Anarchy

In this subsection, we derive exact bounds on the price of anarchy of ρ-uniform congestion games induced by affine weighted congestion models.

▶ **Theorem 3.** *Fix an affine weighted congestion model* CM. *For any $\rho \geq 1$, we have*

$$
\mathsf{PoA}_\rho(\rho\text{-CG}(\mathsf{CM})) \leq \begin{cases} \frac{\sqrt{4\rho+1}+2\rho+1}{2\rho} & \text{if } \rho < 5, \\ \frac{4\rho^2}{3\rho^2-2\rho-1} & \text{if } \rho \geq 5. \end{cases}
$$

Proof. Given an integer $\rho \geq 1$, let CM be an arbitrary affine weighted congestion model. Let $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}^*$ be a ρ-uniform mixed equilibrium and a ρ-uniform social optimum for ρ-CG(CM), respectively. By exploiting the primal-dual technique we get the following linear program:

$$
\begin{aligned}
\max \quad & \mathsf{SUM}(\boldsymbol{\sigma}) = \sum_{i \in \mathsf{N}} w_i cost_i(\boldsymbol{\sigma}) \\
\text{s.t.} \quad & \sum_{i \in \mathsf{N}} w_i cost_i(\boldsymbol{\sigma}) \leq \sum_{i \in \mathsf{N}} w_i cost_i(\boldsymbol{\sigma}_{-i}, \sigma_i^*) \qquad (5) \\
& \mathsf{SUM}(\boldsymbol{\sigma}^*) = \sum_{i \in \mathsf{N}} w_i cost_i(\boldsymbol{\sigma}^*) = 1 \qquad (6) \\
& \alpha_e \geq 0, \quad \forall e \in E
\end{aligned}
$$

where:

- (5) has been obtained by multiplying each inequality $cost_i(\boldsymbol{\sigma}) \leq cost_i(\boldsymbol{\sigma}_{-i}, \sigma_i^*)$ by w_i, and then summing them up for each $i \in \mathsf{N}$;
- the linear coefficients α_e's are the variables of the linear program, and the other quantities are fixed parameters;

- (6) normalizes the optimal social function, so that the maximum value of the objective function (i.e. the social function of the ρ-uniform mixed equilibrium) is an upper bound on the price of anarchy.

By using Lemma 1 in the previous linear program, we get the following relaxation (relaxation comes from inequality (2), that may not be tight):

$$\begin{aligned}
\mathsf{LP}: \quad \max \quad & \sum_{e\in E}\alpha_e\left(\frac{k_{\rho,e}(\boldsymbol{\sigma})^2}{\rho^2}+\left(\frac{\rho-1}{\rho^2}\right)\sum_{i:e\in E(\sigma_i)}w_i^2\right) && (7)\\
\text{s.t.} \quad & \sum_{e\in E}\alpha_e\left(\frac{k_{\rho,e}(\boldsymbol{\sigma})^2}{\rho^2}+\left(\frac{\rho-1}{\rho^2}\right)\sum_{i:e\in E(\sigma_i)}w_i^2\right)\leq\\
& \leq\sum_{e\in E}\alpha_e\left(\frac{k_{\rho,e}(\boldsymbol{\sigma}^*)k_{\rho,e}(\boldsymbol{\sigma})}{\rho^2}+\sum_{i:e\in E(\sigma_i^*)}\frac{w_i^2}{\rho}\right) && (8)\\
& \sum_{e\in E}\alpha_e\left(\frac{k_{\rho,e}(\boldsymbol{\sigma}^*)^2}{\rho^2}+\left(\frac{\rho-1}{\rho^2}\right)\sum_{i:e\in E(\sigma_i^*)}w_i^2\right)=1 && (9)\\
& \alpha_e\geq 0, \quad \forall e\in E
\end{aligned}$$

where (7) comes from (1), as $\sum_{i\in\mathsf{N}} w_i cost_i(\boldsymbol{\sigma}) = \sum_{i\in\mathsf{N}} w_i \sum_{e\in E(\sigma_i)}\alpha_e\left(\frac{k_{\rho,e}(\boldsymbol{\sigma})}{\rho^2}+\left(\frac{\rho-1}{\rho^2}\right)w_i\right) = \sum_{e\in E}\alpha_e\left(\frac{k_{\rho,e}(\boldsymbol{\sigma})^2}{\rho^2}+\left(\frac{\rho-1}{\rho^2}\right)\sum_{i:e\in E(\sigma_i)}w_i^2\right)$, and (8) comes from (2), as $\sum_{i\in\mathsf{N}} w_i cost_i(\boldsymbol{\sigma}_{-i},\sigma_i^*) \leq \sum_{i\in\mathsf{N}} w_i \sum_{e\in E(\sigma_i^*)}\alpha_e\left(\frac{k_{\rho,e}(\boldsymbol{\sigma})}{\rho^2}+\frac{w_i}{\rho}\right) = \sum_{e\in E}\alpha_e\left(\frac{k_{\rho,e}(\boldsymbol{\sigma}^*)k_{\rho,e}(\boldsymbol{\sigma})}{\rho^2}+\sum_{i:e\in E(\sigma_i^*)}\frac{w_i^2}{\rho}\right)$. By taking the dual of LP, where we associate the dual variable x to the primal constraint (8) and the dual variable γ to the primal constraint (9), we get

$$\begin{aligned}
\mathsf{DLP}: \quad \min \quad & \gamma\\
\text{s.t.} \quad & \gamma\left(\frac{k_{\rho,e}(\boldsymbol{\sigma}^*)^2}{\rho^2}+\left(\frac{\rho-1}{\rho^2}\right)\sum_{i:e\in E(\sigma_i^*)}w_i^2\right)\geq\\
& \geq -(x-1)\left(\frac{k_{\rho,e}(\boldsymbol{\sigma})^2}{\rho^2}+\left(\frac{\rho-1}{\rho^2}\right)\sum_{i:e\in E(\sigma_i)}w_i^2\right)+\\
& \quad + x\left(\frac{k_{\rho,e}(\boldsymbol{\sigma}^*)k_{\rho,e}(\boldsymbol{\sigma})}{\rho^2}+\sum_{i:e\in E(\sigma_i^*)}\frac{w_i^2}{\rho}\right), \quad \forall e\in E && (10)\\
& x\geq 0
\end{aligned}$$

By choosing $x>1$, we have that, if $k_{\rho,e}(\boldsymbol{\sigma}^*)=0$, (10) is always satisfied. Thus, assume that $k_{\rho,e}(\boldsymbol{\sigma}^*)>0$. Let us now manipulate (10) as follows: we divide both sides by $k_{\rho,e}(\boldsymbol{\sigma}^*)^2$, so that we can rewrite it as a function of a variable $t := k_{\rho,e}(\boldsymbol{\sigma})/k_{\rho,e}(\boldsymbol{\sigma}^*)$ and of some new player's weights $u_i = w_i/k_{\rho,e}(\boldsymbol{\sigma}^*)$. Furthermore, by setting $\sum_{i:e\in E(\sigma_i)} w_i^2 = 0$ we have stronger constraints. By observing that $\sum_{i:e\in E(\sigma_i^*)} w_i = k_{\rho,e}(\boldsymbol{\sigma}^*)$, and then $\sum_{i:e\in E(\sigma_i^*)} u_i = 1$,

we obtain that the following value $\gamma(x)$ is a feasible solution of DLP for any $x > 1$:[3]

$$\gamma(x) = \sup_{\left\{n\in\mathbb{N}, t\geq 0, u_i\geq 0, \sum_{i=1}^n u_i = 1\right\}} \frac{\frac{t^2}{\rho^2} + x\left(-\frac{t^2}{\rho^2} + \frac{t}{\rho^2} + \frac{1}{\rho}\sum_{i=1}^n u_i^2\right)}{\frac{1}{\rho^2} + \frac{\rho-1}{\rho^2}\sum_{i=1}^n u_i^2}.$$

Since $\sum_{i=1}^n u_i = 1$ and $u_i \geq 0$ for each $i \in [n]$, one can replace $\sum_{i=1}^n u_i^2$ with a variable $u \in [0,1]$, so that we can set $\gamma(x) = \sup_{t\geq 0, u\in[0,1]} \gamma(x,t,u)$, where

$$\gamma(x,t,u) = \frac{\frac{t^2}{\rho^2} + x\left(-\frac{t^2}{\rho^2} + \frac{t}{\rho^2} + \frac{u}{\rho}\right)}{\frac{1}{\rho^2} + \frac{\rho-1}{\rho^2}u}.$$

We have that the monotonicity of $\gamma(x,t,u)$ with respect to u does not depend on u, thus $\gamma(x,t,u)$ is maximized either by $u = 0$ or by $u = 1$. So, we get $\gamma(x) = \sup_{t\geq 0, u\in\{0,1\}} \gamma(x,t,u)$. Note that $t = \frac{x}{2x-2}$ is the unique maximum point of $\gamma(x,t,u)$ for $u \in \{0,1\}$. Thus, we can conclude

$$\gamma(x) = \max\left\{\gamma\left(x, \frac{x}{2x-2}, 0\right), \gamma\left(x, \frac{x}{2x-2}, 1\right)\right\}.$$

Now, if $\rho < 5$, set $x = 1 + \frac{1}{\sqrt{4\rho+1}}$, otherwise set $x = \frac{4\rho}{3\rho+1}$. If $\rho < 5$, we get

$$\gamma\left(1 + \frac{1}{\sqrt{4\rho+1}}\right) = \frac{\sqrt{4\rho+1} + 2\rho + 1}{2\rho} \geq \mathsf{PoA}_\rho(\rho\text{-CG}),$$

otherwise, for $\rho \geq 5$, we get

$$\gamma\left(\frac{4\rho}{3\rho+1}\right) = \frac{4\rho^2}{3\rho^2 - 2\rho - 1} \geq \mathsf{PoA}_\rho(\rho\text{-CG}),$$

thus showing the claim. ◀

We show that the derived upper bounds are tight, even when restricting to games induced by symmetric load balancing models.

▶ **Theorem 4.** *For any $\rho \geq 1$ and $\epsilon > 0$, there exists an affine weighted symmetric load balancing model $\mathsf{CM} := \mathsf{CM}(\rho,\epsilon)$ such that*

$$\mathsf{PoA}_\rho(\rho\text{-CG}(\mathsf{CM})) \geq \begin{cases} \frac{\sqrt{4\rho+1}+2\rho+1}{2\rho} - \epsilon & \text{if } \rho < 5, \\ \frac{4\rho^2}{3\rho^2-2\rho-1} - \epsilon & \text{if } \rho \geq 5. \end{cases}$$

4 Unweighted Games

In this section, we consider the case of ρ-uniform congestion games induced by unweighted congestion models. First, we show that uniform mixed equilibria are always guaranteed to exist for any class of latency functions.

Given an unweighted congestion model CM and an integer $\rho \geq 1$, let f be a function mapping CM and ρ to another congestion model $f(\mathsf{CM},\rho)$, according to the following definition.

[3] To simplify the notation, we have written $\sum_{i=1}^n u_i$, instead of $\sum_{i:e\in E(\sigma_i^*)} u_i$.

▶ **Definition 5.** Given an unweighted congestion model

$$\mathsf{CM} = \left(\mathsf{N}, E, (\ell_e)_{e\in E}, \vec{1}^n, (\Sigma_i)_{i\in\mathsf{N}}\right),$$

define $\mathsf{CM}' = f(\mathsf{CM},\rho) = \left(\mathsf{N}', E', (\ell'_e)_{e\in E}, \vec{1}^n, (\Sigma'_i)_{i\in\mathsf{N}}\right)$ as the unweighted congestion model such that $\mathsf{N}' = \mathsf{N}$, $E' = E$, $\Sigma'_i = \{E(\sigma_i) : \sigma_i \text{ is a } \rho\text{-uniform mixed strategy for player } i \text{ in } \rho\text{-}\mathsf{CG}(\mathsf{CM})\}$ for each $i \in \mathsf{N}$, and

$$\ell'_e(x) := \frac{1}{\rho}\left(\sum_{j=0}^{x-1}\binom{x-1}{j}\left(\frac{1}{\rho}\right)^j\left(\frac{\rho-1}{\rho}\right)^{x-1-j}\ell_e(j+1)\right) \tag{11}$$

for each $e \in E'$. Moreover, given a latency function ℓ_e, let $\ell_e^{f(\rho)}$ denote the latency function defined in (11), and let $\ell_e^{-f(\rho)}$ denote the function $\overline{\ell}_e$ such that $\overline{\ell}_e^{f(\rho)} = \ell_e$.

For instance, if $\mathsf{CG}(\mathsf{CM})$ is a symmetric load balancing game, then $\mathsf{CG}(f(\mathsf{CM},\rho))$ is a ρ-uniform matroid congestion game [15], i.e. the strategies of each player are arbitrary subsets of ρ resources.

We show that ρ-$\mathsf{CG}(\mathsf{CM})$ is equivalent to $\mathsf{CG}(f(\mathsf{CM},\rho))$ for each $\rho \geq 1$. For a ρ-uniform mixed profile $\boldsymbol{\sigma}$ for ρ-$\mathsf{CG}(\mathsf{CM})$, define $\mathbf{s}'(\boldsymbol{\sigma})$ as the strategy profile for $\mathsf{CG}(f(\mathsf{CM},\rho))$ such that $\mathbf{s}'(\boldsymbol{\sigma}) := (E(\sigma_1), E(\sigma_2), \ldots, E(\sigma_n))$.

▶ **Theorem 6.** *Given $\rho \geq 1$ and an unweighted congestion model CM, we have that, for each ρ-uniform mixed profile $\boldsymbol{\sigma}$ for ρ-$\mathsf{CG}(\mathsf{CM})$ and $i \in \mathsf{N}$, $cost_i^{\rho\text{-}\mathsf{CG}(\mathsf{CM})}(\boldsymbol{\sigma}) = cost_i^{\mathsf{CG}(f(\mathsf{CM},\rho))}(\mathbf{s}'(\boldsymbol{\sigma}))$.*

As a corollary, we obtain existence of uniform mixed equilibria for each uniform congestion games induced by unweighted congestion models, regardless of which are their latency functions. In particular, we extend Rosenthal's Theorem [30], by showing that, for each $\rho \geq 1$, any ρ-uniform unweighted congestion game admits an exact potential.

▶ **Corollary 7.** *For each $\rho \geq 1$ and unweighted congestion model CM, ρ-$\mathsf{CG}(\mathsf{CM})$ admits an exact potential.*

Proof. By Rosenthal's Theorem [30], $\mathsf{CG}(f(\mathsf{CM},\rho))$ admits an exact potential function Φ. Because of Theorem 6, we have that $\Phi\circ\boldsymbol{s}'$ is an exact potential function for ρ-$\mathsf{CG}(\mathsf{CM})$. Indeed, given $i \in \mathsf{N}$, a strategy profile $\boldsymbol{\sigma}$ of ρ-$\mathsf{CG}(\mathsf{CM})$, and $\sigma'_i \in \Delta_i^\rho(\mathsf{CM})$, we get $cost_i^{\rho\text{-}\mathsf{CG}(\mathsf{CM})}(\boldsymbol{\sigma}) - cost_i^{\rho\text{-}\mathsf{CG}(\mathsf{CM})}(\boldsymbol{\sigma}_{-i},\sigma'_i) = cost_i^{\mathsf{CG}(f(\mathsf{CM},\rho))}(\boldsymbol{s}'(\boldsymbol{\sigma})) - cost_i^{\mathsf{CG}(f(\mathsf{CM},\rho))}(\boldsymbol{s}'(\boldsymbol{\sigma}_{-i},\sigma'_i)) = \Phi(\boldsymbol{s}'(\boldsymbol{\sigma})) - \Phi(\boldsymbol{s}'(\boldsymbol{\sigma}_{-i},\sigma'_i)) = (\Phi\circ\boldsymbol{s}')(\boldsymbol{\sigma}) - (\Phi\circ\boldsymbol{s}')(\boldsymbol{\sigma}_{-i},\sigma'_i)$. ◀

4.1 Price of Anarchy

In this subsection, we derive exact bounds on the price of anarchy of ρ-uniform congestion games induced by affine unweighted congestion models.

▶ **Theorem 8.** *Fix an affine unweighted congestion model CM. For any $\rho \geq 1$, we have*

$$\mathsf{PoA}_\rho(\rho\text{-}\mathsf{CG}(\mathsf{CM})) \leq \begin{cases} \frac{5}{\rho+1} & \text{if } \rho \leq 2, \\ \frac{2\sqrt{7}-1}{3} & \text{if } \rho = 3, \\ \frac{4}{3} & \text{if } \rho \geq 4. \end{cases}$$

Proof. Given an integer $\rho \geq 1$, let CM be an arbitrary affine unweighted congestion model. Let $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}^*$ be a ρ-uniform mixed equilibrium and a ρ-uniform social optimum for

ρ-CG(CM), respectively. By exploiting (1) and (2), we have that, for each $i \in \mathsf{N}$, the inequality $cost_i(\boldsymbol{\sigma}) \leq cost_i(\boldsymbol{\sigma}_{-i}, \sigma_i^*)$ becomes

$$\sum_{e \in E(\sigma_i)} \alpha_e \left(\frac{k_{\rho,e}(\boldsymbol{\sigma}) + \rho - 1}{\rho^2} \right) - \sum_{e \in E(\sigma_i^*)} \alpha_e \left(\frac{k_{\rho,e}(\boldsymbol{\sigma}) + \rho}{\rho^2} \right) \leq 0.$$

By also using (1) within SUM($\boldsymbol{\sigma}$) and SUM($\boldsymbol{\sigma}^*$), we get the following linear program:

$$\begin{aligned}
\mathsf{LP}: \quad \max \quad & \sum_{e \in E} \alpha_e \left(\frac{k_{\rho,e}(\boldsymbol{\sigma})(k_{\rho,e}(\boldsymbol{\sigma}) + \rho - 1)}{\rho^2} \right) \\
\text{s.t.} \quad & \sum_{e \in E(\sigma_i)} \alpha_e \left(\frac{k_{\rho,e}(\boldsymbol{\sigma}) + \rho - 1}{\rho^2} \right) - \sum_{e \in E(\sigma_i^*)} \alpha_e \left(\frac{k_{\rho,e}(\boldsymbol{\sigma}) + \rho}{\rho^2} \right) \leq 0, \quad \forall i \in \mathsf{N} \qquad (12) \\
& \sum_{e \in E} \alpha_e \left(\frac{k_{\rho,e}(\boldsymbol{\sigma}^*)(k_{\rho,e}(\boldsymbol{\sigma}^*) + \rho - 1)}{\rho^2} \right) = 1 \qquad (13) \\
& \alpha_e \geq 0, \quad \forall e \in E
\end{aligned}$$

By taking the dual of LP, where we associate the dual variable x_i to the ith primal constraint in (12) and the dual variable γ to the primal constraint (13), we get:

$$\begin{aligned}
\mathsf{DLP}: \quad \min \quad & \gamma \\
\text{s.t.} \quad & \sum_{i : e \in E(\sigma_i)} \left(x_i \frac{k_{\rho,e}(\boldsymbol{\sigma}) + \rho - 1}{\rho^2} \right) - \sum_{i : e \in E(\sigma_i^*)} \left(x_i \frac{k_{\rho,e}(\boldsymbol{\sigma}) + \rho}{\rho^2} \right) \\
& + \gamma \frac{k_{\rho,e}(\boldsymbol{\sigma}^*)(k_{\rho,e}(\boldsymbol{\sigma}^*) + \rho - 1)}{\rho^2} \\
& \geq \frac{k_{\rho,e}(\boldsymbol{\sigma})(k_{\rho,e}(\boldsymbol{\sigma}) + \rho - 1)}{\rho^2}, \quad \forall e \in E \qquad (14) \\
& x_i \geq 0, \quad \forall i \in \mathsf{N}
\end{aligned}$$

By using $x_i = x$ for each $i \in \mathsf{N}$, $k := k_{\rho,e}(\boldsymbol{\sigma})$ and $o := k_{\rho,e}(\boldsymbol{\sigma}^*)$ in (14), and multiplying both sides by ρ^2, we obtain the following relaxed dual constraint:

$$xk(k + \rho - 1) - xo(k + \rho) + \gamma o(o + \rho - 1) \geq k(k + \rho - 1). \qquad (15)$$

To complete the proof, we are left to provide, for each $\rho \geq 1$, a suitable value $x \geq 0$ satisfying inequality (15) where γ is set to be equal to the claimed upper bound on the ρ-uniform price of anarchy. We now proceed by case analysis.

For $\rho \leq 2$, for which we have $\gamma = \frac{5}{\rho+1}$, set $x = \frac{\rho+2}{\rho+1}$. By substituting these values in (15), we get the inequality $k^2 - k\left(o(\rho+2) + \rho + 1\right) + o(5o - \rho^2 + 3\rho - 5) \geq 0$ which can be easily shown to be satisfied for any pair of non-negative integers k, o when $\rho = 1, 2$. In fact, the discriminant of the associated equality is negative for each integer $o \geq 2$, while the cases of $o \in \{0, 1\}$ can be checked by inspection.

For $\rho = 3$, for which we have $\gamma = \frac{2\sqrt{7}-1}{3}$, set $x = 2\sqrt{7} - 4$. By substituting these values in (15), we get the inequality

$$(6\sqrt{7} - 15)k^2 - 6k\left(o(\sqrt{7} - 2) - 2\sqrt{7} + 5\right) + o\left(o(2\sqrt{7} - 1) - 14\sqrt{7} + 34\right) \geq 0,$$

which can be easily shown to be satisfied for any pair of non-negative integers k, o. In fact, the discriminant of the associated equality is negative for each integer $o \geq 2$, while the cases of $o \in \{0, 1\}$ can be checked by inspection.

For $\rho \geq 4$, for which we have $\gamma = \frac{4}{3}$, set $x = \frac{4}{3}$. By substituting these values in (15), we get the inequality $k^2 - k(4o - \rho + 1) + 4o(o-1) \geq 0$ whose left-hand side is increasing in ρ. Hence, we only need to prove that it gets satisfied for the case of $\rho = 4$, by which we get the inequality $k^2 - k(4o-3) + 4o(o-1) \geq 0$ which can be easily shown to be satisfied for any pair of non-negative integers k, o. Again, the discriminant of the associated equality is negative for each integer $o \geq 2$, while the cases of $o \in \{0, 1\}$ can be checked by inspection. ◀

We show matching lower bounds for each $\rho \leq 3$. For $\rho \geq 4$, we show in the next subsection a matching lower bound holding even for the price of stability.

▶ **Theorem 9.** *For any $\rho \leq 3$ and $\epsilon > 0$, there exists an affine unweighted load balancing model $\mathsf{CM} := \mathsf{CM}(\rho, \epsilon)$ such that*

$$\mathsf{PoA}_\rho(\rho\text{-}\mathsf{CG}(\mathsf{CM})) \geq \begin{cases} \frac{5}{\rho+1} - \epsilon & \text{if } \rho \leq 2, \\ \frac{2\sqrt{7}-1}{3} - \epsilon & \text{if } \rho = 3. \end{cases}$$

4.2 Price of Stability

In this subsection, we exhibit exact bounds on the price of stability of ρ-uniform congestion games induced by affine unweighted congestion models.

▶ **Theorem 10.** *Fix an affine unweighted congestion model CM. For any $\rho \geq 1$, we have*

$$\mathsf{PoS}_\rho(\rho\text{-}\mathsf{CG}(\mathsf{CM})) \leq \begin{cases} 1 + \frac{1}{\sqrt{2\rho+1}} & \text{if } \rho \leq 3, \\ \frac{4}{3} & \text{if } \rho \geq 4. \end{cases}$$

We also have matching lower bounds. We first consider the case of $\rho \leq 3$.

▶ **Theorem 11.** *For each $\rho \leq 3$ and $\epsilon > 0$, there exists an affine unweighted congestion model $\mathsf{CM} := \mathsf{CM}(\rho, \epsilon)$ such that $\mathsf{PoS}_\rho(\rho\text{-}\mathsf{CG}(\mathsf{CM})) \geq 1 + \frac{1}{\sqrt{2\rho+1}} - \epsilon$.*

For $\rho \geq 4$, the upper bounds are tight even when restricting to games induced by symmetric load balancing models.

▶ **Theorem 12.** *For each $\rho \geq 1$ and $\epsilon > 0$, there exists an affine unweighted symmetric load balancing model $\mathsf{CM} := \mathsf{CM}(\rho, \epsilon)$ such that $\mathsf{PoS}_\rho(\rho\text{-}\mathsf{CG}(\mathsf{CM})) \geq \frac{4}{3} - \epsilon$.*

5 Open Problems

In this paper, motivated by possible applications in fault-tolerant routing, we have introduced the notion of uniform mixed equilibria, and we have applied it to the well-studied class of (network) congestion games with affine latency functions, by providing existential results of these equilibria and by deriving tight bounds to the prices of anarchy and stability.

The main left open problem is to consider the more general definition of $\boldsymbol{\rho}$-uniform mixed equilibria, where players can use uniform mixed strategies of different support size. Another important question is the determination of lower bounds for the price of stability of ρ-unform mixed equilibria, in the setting of weighted congestion games. However, this question is open even for the price of stability of pure Nash equilibria (i.e., $\rho = 1$), for which only an upper bound equal to 2 is known, as a direct consequence of the potential function given in [20]. Following the approach used in [12, 10, 19, 9] for $\rho = 1$, it could be interesting investigating resource taxation or other strategies to improve the performance of ρ-unform mixed equilibria. Another interesting research direction is that of extending the results to other latency functions, e.g., polynomial functions, or decreasing functions as the ones inducing the Shapley cost sharing game [2, 24, 8, 7, 18].

References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.
2. E. Anshelevich, A. Dasgupta, J. M. Kleinberg, É. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. *SIAM J. Comput.*, 38(4):1602–1623, 2008.
3. B. Awerbuch, Y. Azar, and A. Epstein. The price of routing unsplittable flow. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC, pages 57–66, 2005.
4. M. Babaioff, R. Kleinberg, and C. H. Papadimitriou. Congestion games with malicious players. *Games and Economic Behavior*, 67(1):22–35, 2009.
5. K. Bhawalkar, M. Gairing, and T. Roughgarden. Weighted congestion games: price of anarchy, universal worst-case examples, and tightness. *ACM Transactions on Economics and Computation*, 2(4):1–23, 2014.
6. V. Bilò. A unifying tool for bounding the quality of non-cooperative solutions in weighted congestion games. In *Proceedings of the 10th Workshop on Approximation and Online Algorithms (WAOA)*, volume 7846 of *LNCS*, pages 229–241, 2012.
7. V. Bilò, I. Caragiannis, A. Fanelli, and G. Monaco. Improved lower bounds on the price of stability of undirected network design games. *Theory Comput. Syst.*, 52(4):668–686, 2013.
8. V. Bilò, M. Flammini, and L. Moscardelli. The price of stability for undirected broadcast network design with fair cost allocation is constant. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*, FOCS, pages 638–647, 2013.
9. V. Bilò and C. Vinci. On Stackelberg strategies in affine congestion games. In *Proceedings of the 11th Conference on Web and Internet Economics (WINE)*, LNCS" volume = "9470, pages 132–145, 2015.
10. V. Bilò and C. Vinci. Dynamic taxes for polynomial congestion games. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, EC '16, pages 839–856, 2016.
11. I. Caragiannis, M. Flammini, C. Kaklamanis, P. Kanellopoulos, and L. Moscardelli. Tight bounds for selfish and greedy load balancing. *Algorithmica*, 61(3):606–637, 2011.
12. I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. Taxes for linear atomic congestion games. *ACM Trans. Algorithms*, 7(1):13:1–13:31, 2010.
13. G. Christodoulou and E. Koutsoupias. On the price of anarchy and stability of correlated equilibria of linear congestion games. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA)*, volume 3669 of *LNCS*, pages 59–70, 2005.
14. G. Christodoulou and E. Koutsoupias. The price of anarchy of finite congestion games. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 67–73, 2005.
15. J. de Jong, M. Klimm, and M. Uetz. Efficiency of equilibria in uniform matroid congestion games. In *Proceedings of the 9th International Symposium on Algorithmic Game Theory (SAGT)*, volume 9928 of *LNCS*, pages 105–116, 2016.
16. C. Deeparnab, M. Aranyak, and N. Viswanath. Fairness and optimality in congestion games. In *Proceedings of the 6th ACM Conference on Electronic Commerce*, EC '05, pages 52–57, 2005.
17. A. Fabrikant, C. H. Papadimitriou, and K. Talwar. The complexity of pure Nash equilibria. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 604–612, 2004.
18. A. Fiat, K. Kaplan, M. Levy, S. Olonetsky, and R. Shabo. On the price of stability for designing undirected networks with fair cost allocations. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I*, pages 608–618, 2006.

19 D. Fotakis. Stackelberg strategies for atomic congestion games. *Theor. Comp. Sys.*, 47(1):218–249, 2010.

20 D. Fotakis, S. C. Kontogiannis, and P. G. Spirakis. Selfish unsplittable flows. *Theoretical Computer Science*, 348:226–239, 2005.

21 D. Fotakis, S. C. Kontogiannis, and P. G. Spirakis. Symmetry in network congestion games: pure equilibria and anarchy cost. In *Proceedings of the 3rd Workshop on Approximation and Online Algorithms (WAOA)*, volume 3879 of *LNCS*, pages 161–175, 2005.

22 T. Harks, M. Klimm, and R.H. Möhring. Characterizing the existence of potential functions in weighted congestion games. *Theory of Computing Systems*, 49(1):46–70, 2011.

23 G. Karakostas and A. Viglas. Equilibria for networks with malicious users. *Math. Program.*, 110(3):591–613, 2007.

24 J. Li. An O(log(n)/log(log(n))) upper bound on the price of stability for undirected Shapley network design games. *Information Processing Letters*, 109(15):876–878, 2009.

25 T. Moscibroda, S. Schmid, and R. Wattenhofer. When selfish meets evil: Byzantine players in a virus inoculation game. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 35–44, 2006.

26 J. F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Science*, 36(1):48–49, 1950.

27 P. N. Panagopoulou and P. G. Spirakis. Algorithms for pure Nash equilibria in weighted congestion games. *Journal of Experimental Algorithmics*, 11(2.7), 2006.

28 M. Penn, M. Polukarov, and M. Tennenholtz. Congestion games with load-dependent failures: Identical resources. *Games and Economic Behavior*, 67(1):156–173, 2009.

29 M. Penn, M. Polukarov, and M. Tennenholtz. Congestion games with failures. *Discrete Applied Mathematics*, 159(15):1508–1525, 2011.

30 R. W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

31 A. Roth. The price of malice in linear congestion games. In *Internet and Network Economics, 4th International Workshop, WINE 2008. Proceedings*, pages 118–125, 2008.

32 T. Roughgarden and É. Tardos. How bad is selfish routing? *J. ACM*, 49(2):236–259, 2002.

Byzantine Gathering in Polynomial Time

Sébastien Bouchard
Sorbonne Universités, UPMC Univ Paris 06, CNRS, INRIA, LIP6 UMR 7606, Paris, France
sebastien.bouchard@lip6.fr

Yoann Dieudonné
Laboratoire MIS & Université de Picardie Jules Verne, Amiens, France
yoann.dieudonne@u-picardie.fr

Anissa Lamani
Laboratoire MIS & Université de Picardie Jules Verne, Amiens, France
anissa.lamani@u-picardie.fr

Abstract

Gathering a group of mobile agents is a fundamental task in the field of distributed and mobile systems. This can be made drastically more difficult to achieve when some agents are subject to faults, especially the Byzantine ones that are known as being the worst faults to handle. In this paper we study, from a deterministic point of view, the task of Byzantine gathering in a network modeled as a graph. In other words, despite the presence of Byzantine agents, all the other (good) agents, starting from possibly different nodes and applying the same deterministic algorithm, have to meet at the same node in finite time and stop moving. An adversary chooses the initial nodes of the agents (the number of agents may be larger than the number of nodes) and assigns a different positive integer (called label) to each of them. Initially, each agent knows its label. The agents move in synchronous rounds and can communicate with each other only when located at the same node. Within the team, f of the agents are Byzantine. A Byzantine agent acts in an unpredictable and arbitrary way. For example, it can choose an arbitrary port when it moves, can convey arbitrary information to other agents and can change its label in every round, in particular by forging the label of another agent or by creating a completely new one.

Besides its label, which corresponds to a local knowledge, an agent is assigned some global knowledge denoted by $\mathcal{GK}$ that is common to all agents. In literature, the Byzantine gathering problem has been analyzed in arbitrary n-node graphs by considering the scenario when $\mathcal{GK} = (n, f)$ and the scenario when $\mathcal{GK} = f$. In the first (resp. second) scenario, it has been shown that the minimum number of good agents guaranteeing deterministic gathering of all of them is $f+1$ (resp. $f+2$). However, for both these scenarios, all the existing deterministic algorithms, whether or not they are optimal in terms of required number of good agents, have the major disadvantage of having a time complexity that is exponential in n and L, where L is the value of the largest label belonging to a good agent. In this paper, we seek to design a deterministic solution for Byzantine gathering that makes a concession on the proportion of Byzantine agents within the team, but that offers a significantly lower complexity. We also seek to use a global knowledge whose the length of the binary representation (that we also call size) is small. In this respect, assuming that the agents are in a *strong team* i.e., a team in which the number of good agents is at least some prescribed value that is quadratic in f, we give positive and negative results. On the positive side, we show an algorithm that solves Byzantine gathering with all strong teams in all graphs of size at most n, for any integers n and f, in a time polynomial in n and the length $|l_{min}|$ of the binary representation of the smallest label of a good agent. The algorithm works using a global knowledge of size $\mathcal{O}(\log\log\log n)$, which is of optimal order of magnitude in our context to reach a time complexity that is polynomial in n and $|l_{min}|$. Indeed, on the negative side, we show that there is no deterministic algorithm solving Byzantine gathering with all strong teams, in all graphs of size at most n, in a time polynomial in n and $|l_{min}|$ and using a global knowledge of size $o(\log\log\log n)$.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 147; pp. 147:1–147:15


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Theory of computation → Distributed algorithms, Computing methodologies → Mobile agents

Keywords and phrases gathering, deterministic algorithm, mobile agent, Byzantine fault, polynomial time

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.147

Related Version A full version of the paper is available at `https://arxiv.org/abs/1801.07656`.

Funding This work was performed within Project ESTATE (Ref. ANR-16-CE25-0009-03) and Project TOREDY. The first project is supported by French state funds managed by the ANR (Agence Nationale de la Recherche), while the second project is supported by the European Regional Development Fund (ERDF) and the Hauts-de-France region.

1 Introduction

1.1 Context

Gathering a group of mobile agents is a basic problem that has been widely studied in literature dedicated to mobile and distributed systems. One of the main reasons for this popularity stems from the fact that this task turns out to be an essential prerequisite to achieve more complex cooperative works. In other words, getting fundamental results on the problem of gathering implies *de facto* getting fundamental results on a large set of problems whose resolution needs to use gathering as a building block.

The scale-up when considering numerous agents is inevitably tied to the occurrence of faults among them, the most emblematic of which is the Byzantine one. Byzantine faults are very interesting under multiple aspects, especially because the Byzantine case is the most general one, as it subsumes all the others kind of faults. In fact, in the field of fault tolerance they are considered as the worst faults that can occur.

In this paper, we consider the problem of gathering in a deterministic way in a network modeled as a graph, wherein some agents are Byzantine. A Byzantine agent acts in an unpredictable and arbitrary manner. For instance it may choose to never stop or to never move. It may also convey arbitrary information to the other agents, impersonate the identity of another agent, and so on. In such a context, gathering is very challenging, and so far the power of such Byzantine agents has been offset by a huge complexity when solving this problem. In what follows, we seek a solution allowing to withstand Byzantine agents while keeping a "reasonable" complexity.

1.2 Model and problem

A team of mobile agents are initially placed by an adversary at arbitrary nodes of a network modeled as a finite, connected, undirected graph $G = (V, E)$. We assume that $|V| \leq n$. Several agents may initially share the same node and the size of the team may be larger than n. Two assumptions are made about the labelling of the two main components of the graph that are nodes and edges. The first assumption is that nodes are anonymous i.e., they do not have any kind of labels or identifiers allowing them to be distinguished from one another. The second assumption is that edges incident to a node v are locally ordered with a fixed port numbering ranging from 0 to $deg(v) - 1$ where $deg(v)$ is the degree of v. Therefore, each edge

has exactly two port numbers, one for each of both nodes it links. The port numbering is not supposed to be consistent: a given edge $(u, v) \in E$ may be the i-th edge of u but the j-th edge of v, where $i \neq j$. These two assumptions are not fortuitous. The primary motivation of the first one is that if each node could be identified by a label, gathering would become quite easy to solve as it would be tantamount to explore the graph (via e.g. a breadth-first search) and then meet in the node having the smallest label. While the first assumption is made so as to avoid making the problem trivial, the second assumption is made in order to avoid making the problem impossible to solve. Indeed, in the absence of a way allowing an agent to distinguish locally the edges incident to a node, gathering could be proven as impossible to solve deterministically in view of the fact that some agents could be precluded from traversing some edges and visit some parts of the graph.

Time is discretized into an infinite sequence of rounds. In each round, every agent, which has been previously woken up (this notion is detailed in the next paragraph), is allowed to stay in place at its current node or to traverse an edge according to a deterministic algorithm. The algorithm is the same for all agents: only the input, whose nature is specified further in the subsection, varies among agents.

Before being woken up, an agent is said to be dormant. A dormant agent may be woken up only in two different ways: either by the adversary that wakes some of the agents at possibly different rounds, or as soon as a non-dormant agent is at the starting node of the dormant agent. We assume that the adversary wakes up at least one agent. Note that, when the adversary chooses to wake up in round r a dormant agent located at a node v, all the dormant agents that are at node v wake up in round r.

When an agent is woken up in a round r, it is told the degree of its starting node. As mentioned above, in each round $r' \geq r$, the executed algorithm can ask the agent to stay idle or to traverse an edge. In the latter case, this takes the following form: the algorithm asks the agent, located at node u, to traverse the edge having port number i, where $0 \leq i \leq deg(u) - 1$. Let us denote by $(u, v) \in E$ this traversed edge. In round $r' + 1$, the agent enters node v: it then learns the degree $deg(v)$ as well as the local port number j of (u, v) at node v (recall that in general $i \neq j$). An agent cannot leave any kind of tokens or markers at the nodes it visits or the edges it traverses.

In the beginning, the adversary also assigns a different positive integer (called label) to each agent. Each agent knows its label but does not know *a priori* the labels of the other agents (except if some or all of them are inserted in the global knowledge $\mathcal{GK}$ that is introduced below). When several agents are at the same node v in the same round t, they see, for each agent x at node v, the label of agent x and all information it wants to share with the others in round t. This transmission of information is done in a "shouting" mode in one round: all the transmitted information by all agents at node v in round t becomes common knowledge for agents that are currently at node v in round t. On the other hand when two agents are not at the same node in the same round they cannot see or talk to each other: in particular, two agents traversing simultaneously the same edge but in opposite directions, and thus crossing each other on the same edge, do not notice this fact. In every round, the input of the algorithm executed by an agent a is made up of the label of agent a, the up-to-date memory of what agent a has seen and learnt since its waking up and some global knowledge denoted by $\mathcal{GK}$. Parameter $\mathcal{GK}$ is a piece of information that is initially given to all agents and common to all of them (i.e., $\mathcal{GK}$ is the same for all agents): its nature is precised at the end of this subsection. Note that in the absence of a way of distinguishing the agents, the gathering problem would have no deterministic solution in some graphs, regardless of the nature of $\mathcal{GK}$. This is especially the case in a ring in which at each node the

edge going clockwise has port number 0 and the edge going anti-clockwise has port 1: if all agents are woken up in the same round and start from different nodes, they will always have the same input and will always follow the same deterministic rules leading to a situation where the agents will always be at distinct nodes no matter what they do.

Within the team, it is assumed that f of the agents are Byzantine. A Byzantine agent has a high capacity of nuisance: it can choose an arbitrary port when it moves, can convey arbitrary information to other agents and can change its label in every round, in particular by forging the label of another agent or by creating a completely new one. All the agents that are not Byzantine are called good. We consider the task of f-Byzantine gathering which is stated as follows. The adversary wakes up at least one good agent and all good agents must eventually be in the same node in the same round, simultaneously declare termination and stop, despite the fact there are f Byzantine agents. Regarding this task, it is worth mentioning that we cannot require the Byzantine agents to cooperate as they may always refuse to be with some agents. Thus, gathering all good agents with termination is the strongest requirement we can make in such a context. The time complexity of an algorithm solving f-Byzantine gathering is the number of rounds counted from the start of the earliest good agent until the task is accomplished.

We end this subsection by explaining what we mean by global knowledge, that can be viewed as a kind of advice given to all agents. Following the paradigm of algorithms with advice [1, 25, 38, 9, 20, 19, 33], $\mathcal{GK}$ is actually a piece of information that is initially provided to the agents at the start, by an oracle knowing the initial instance of the problem. By instance, we precisely mean: the entire graph with its port numbering, the initial positions of the agents with their labels, the f agents that are Byzantine, and for each agent the round, if any, when the adversary wakes it up in case it has not been woken up before by another agent. So, for example, $\mathcal{GK}$ might correspond to the size of the network, the number of Byzantine agents, or a complete map of the network, etc. As mentionned earlier, we assume that $\mathcal{GK}$ is the same for all agents. The size of $\mathcal{GK}$ is the length of its binary representation.

1.3 Related works

When reviewing the chronology of the works that are related to the gathering problem, it can be seen that this problem has been first studied in the particular case in which the team is made of exactly two agents. Under such a limitation, gathering is generally referred to as *rendezvous*. From the first mention of the rendezvous problem in [36], this problem and its generalization, gathering, have been extensively studied in a great variety of ways. Indeed, there is a lot of alternatives for the combinations we can make when addressing the problem, e.g., by playing on the environment in which the agents are supposed to evolve, the way of applying the sequences of instructions (i.e., deterministic or randomized) or the ability to leave some traces in the visited locations, etc. In this paper, we are naturally closer to the research works that are related to deterministic gathering in networks modeled as graphs. Hence, we will mostly dwell on this scenario in the rest of this subsection. However, for the curious reader wishing to consider the matter in greater depth, we invite him to consult [8, 2, 24] that address the problem in the plane via various scenarios, especially in a system affected by the occurrence of faults or inaccuracies for the last two references. Regarding randomized rendezvous, a good starting point is to go through [3, 4, 28].

Now, let us focus on the area that concerns the present paper most directly, namely deterministic rendezvous and/or gathering in graphs. In most papers on rendezvous in networks, a synchronous scenario was assumed, in which agents navigate in the network in synchronous rounds. Under this context, a lot of effort has been dedicated to the study of

the feasibility and to the time (i.e., number of rounds) required to achieve the task, when feasible. For instance, in [16] the authors show a rendezvous algorithm polynomial in the size of the graph, in the length of the shorter label and in the delay between the starting time of the agents. In [26] and [37] solutions are given for rendezvous, which are polynomial in the first two of these parameters and independent of the delay. While these algorithms ensure rendezvous in polynomial time (i.e., a polynomial number of rounds), they also ensure it at polynomial cost where the cost corresponds here to the total number of edge traversals made by both agents until meeting. Indeed, since each agent can make at most one edge traversal per round, a polynomial time always implies a polynomial cost. However, the reciprocal may be not true, for instance when using an algorithm relying on a technique similar to "coding by silence" in the time-slice algorithm for leader election [29]: "most of the time" both agents stay idle, in order to guarantee that agents rarely move simultaneously. Thus these parameters of cost and time are not always linked to each other. This was recently highlighted in [32] where the authors studied the tradeoffs between cost and time for the deterministic rendezvous problem. Some other efforts have been also dedicated to analyse the impact on time complexity of rendezvous when in every round the agents are brought with some pieces of information by making a query to some device or some oracle, see, e.g., [14, 31]. Along with the works aiming at optimizing the parameters of time and/or cost of rendezvous, some other works have examined the amount of memory that is required to achieve deterministic rendezvous e.g., in [21, 22] for tree networks and in [12] for general networks.

Apart from the synchronous scenario, the academic literature also contains several studies focusing on a scenario in which the agents move at constant speed, which are different from each other, or even move asynchronously: in this latter case the speed of agents may then vary and is controlled by the adversary. For more details about rendezvous under such a context, the reader is referred to [30, 13, 23, 18, 27] for rendezvous in finite graphs and [5, 10] for rendezvous in infinite grids.

As stated in the previous subsection, our paper is also related to the field of fault tolerance since some agents may be prone to Byzantine faults. First introduced in [34], a Byzantine fault is an arbitrary fault occurring in an unpredictable way during the execution of a protocol. Due to its arbitrary nature, such a fault is considered as the worst fault that can occur. Byzantine faults have been extensively studied for "classical" networks i.e., in which the entities are fixed nodes of the graph (cf., e.g., the book [29] or the survey [6]). To a lesser extend, the occurrence of Byzantine faults has been also studied in the context of mobile entities evolving on a one-dimensional or two-dimensional space, cf. [2, 15, 11].

Gathering in arbitrary graphs in presence of many Byzantine agents was considered in [17, 7]. Actually, our model is borrowed from both these papers, and thus they are naturally the closest works to ours. In [17], the problem is introduced via the following question: *what is the minimum number $\mathcal{M}$ of good agents that guarantees f-Byzantine gathering in all graphs of size n?* In [17], the authors provided several answers to this problem by firstly considering a relaxed variant, in which the Byzantine agents cannot lie about their labels, and then by considering a harsher form (the same as in our present paper) in which Byzantine agents can lie about their identities. For the relaxed variant, it has been proven that the minimum number $\mathcal{M}$ of good agents that guarantees f-Byzantine gathering is precisely 1 when $\mathcal{GK} = (n, f)$ and $f + 2$ when $\mathcal{GK}$ is reduced to f only. The proof that both these values are enough, relies on polynomial algorithms using a mechanism of blacklists that are, informally speaking, lists of labels corresponding to agents having exhibited an "inconsistent" behavior. Of course, such blacklists cannot be used when the Byzantine agents can change

their labels and in particular steal the identities of good agents. Still in [17], the authors give for the harsher form of f-byzantine gathering a lower bound of $f+1$ (resp. $f+2$) on $\mathcal{M}$ and a deterministic gathering algorithm requiring at least $2f+1$ (resp. $4f+2$) good agents, when $\mathcal{GK}=(n,f)$ (resp. $\mathcal{GK}=f$). Both these algorithms have a huge complexity as they are exponential in n and L, where L is the largest label of a good agent evolving in the graph. Some advances are made in [7], via the design of an algorithm for the case $\mathcal{GK}=(n,f)$ (resp. $\mathcal{GK}=f$) that works with a number of good agents that perfectly matches the lower bound of $f+1$ (resp. $f+2$) shown in [17]. However, these algorithms also suffer from a complexity that is exponential in n and L.

1.4 Our results

As mentioned just above, the existing deterministic algorithms dedicated to f-Byzantine gathering all have the major disadvantage of having a time complexity that is exponential in n and L, when Byzantine agents are allowed to change their labels. Actually, these solutions are all based on a common strategy that consists in enumerating the possible initial configurations, and successively testing them one by one. Once the testing reaches the correct initial configuration, the gathering can be achieved. However, in order to get a significantly more efficient algorithm, such a costly strategy must be abandoned in favor of a completely new one.

In this paper, we seek to design a deterministic solution for Byzantine gathering that makes a concession on the proportion of Byzantine agents within the team, but that offers a significantly lower complexity. We also seek to use a global knowledge whose the length of the binary representation (that we also call size) is small. In this respect, assuming that the agents are in a *strong team* i.e., a team in which the number of good agents is at least the quadratic value $5f^2+6f+2$, we give positive and negative results. On the positive side, we show an algorithm that solves f-Byzantine gathering with all strong teams in all graphs of size at most n, for any integers n and f, in a time polynomial in n and $|l_{min}|$. The algorithm works using a global knowledge of size $\mathcal{O}(\log\log\log n)$, which is of optimal order of magnitude in our context to reach a time complexity that is polynomial in n and $|l_{min}|$. Indeed, on the negative side, we show that there is no deterministic algorithm solving f-Byzantine gathering with all strong teams, in all graphs of size at most n, in a time polynomial in n and $|l_{min}|$ and using a global knowledge of size $o(\log\log\log n)$.

1.5 Roadmap

The next section is dedicated to the presentation of some definitions and routines that we need in the rest of this paper. In Section 3, we give a description of our main algorithm corresponding to our positive result. In Section 4, we prove our negative result. Finally we make some concluding remarks in Section 5. Note that due to the lack of space, several details and proofs are omitted but will appear in the journal version of the paper.

2 Preliminaries

Throughout the paper, log denotes the binary logarithm. The length of the binary representation of the smallest label of a good agent in a given team will be denoted by $|l_{min}|$.

In this paper, we will use a procedure whose aim is graph exploration, i.e., visiting all nodes of the graph. This procedure, based on universal exploration sequences (UXS), is a corollary of the result of Reingold [35]. Given any positive integer n, this procedure, called

EXPLO(n), allows the executing agent to traverse all nodes of any graph of size at most n, starting from any node of this graph, using a number of edge traversals that is polynomial in n. We denote by X_n the execution time of procedure EXPLO with parameter n.

Besides this exploration procedure, we will use a label transformation derived from [16]. Let ℓ_B be the label of an agent B and $b_1 \dots b_c$ its binary representation with c its length. The binary representation of the corresponding transformed label ℓ_B^* is $10b_1b_1 \dots b_cb_c0110b_1b_1 \dots b_cb_c01$. This transformation is made to ensure the following property on which the algorithm of Section 3 partly relies.

▶ **Proposition 1.** *Let ℓ_B and ℓ_X be two labels such that $\ell_B < \ell_X$. Let $b_1^*b_2^* \dots b_{4c+8}^*$ and $x_1^*x_2^* \dots x_{4y+8}^*$ be the respective binary representations of ℓ_B^* and ℓ_X^*, with c and y the lengths of the binary representations of ℓ_B and ℓ_X respectively. There exist two positive integers $i \leq 2c+4$ and $2c+4 < j \leq 4c+8$ such that $b_i^* \neq x_i^*$ and $b_j^* \neq x_j^*$.*

To establish our main result that is presented in Section 3, we needed to design two building blocks, namely procedures GROUP and MERGE. These procedures are not trivial, but due to space restrictions we only give here the two associated theorems.

The first building block called GROUP takes as input three integers $\mathcal{T}$, n and *bin* such that $bin \in \{0; 1\}$. Let x be an integer that is at least $f+2$. Roughly speaking, subroutine GROUP($\mathcal{T}, n, bin$) ensures that $(x-f)$ good agents finish the execution of the subroutine at the same round and in the same node in a graph of size at most n provided some conditions given in the next theorem are satisfied.

▶ **Theorem 1.** *Consider a team made of at least $(x-1)(f+1)+1$ good agents in a graph of size at most n, where $x \geq f+2$. Let Δ be the first round when a good agent starts executing* GROUP($\mathcal{T}, n, bin$). *If all good agents start executing* GROUP($\mathcal{T}, n, bin$) *by round $\Delta + \mathcal{T} - 1$, and parameter bin is 0 (resp. 1) for at least one good agent, then we have the following property. After at most a time polynomial in n and $\mathcal{T}$ from Δ, at least $(x-f)$ good agents finish the execution of* GROUP *at the same round and in the same node.*

The second building block called MERGE takes as input two integers n and $\mathcal{T}$. Subroutine MERGE($\mathcal{T}, n$) allows all the good agents to finish their executions of the subroutine in the same node and at the same round, provided some conditions given in the next theorem are satisfied.

▶ **Theorem 2.** *Consider a team of agents in a graph of size at most n. Let Δ be the first round when a good agent starts executing* MERGE($\mathcal{T}, n$). *If every good agent starts executing* MERGE($\mathcal{T}, n$) *by round $\Delta + \mathcal{T} - 1$ and among them at least $4f+2$ start the execution in the same node and at the same round, then all good agents finish their executions of procedure* MERGE *in the same node and at the same round $r < \Delta + 4\mathcal{T} + 6X_n - 1$.*

3 The positive result

In this section we present an algorithm, called GATHER, that solves f-Byzantine gathering with strong teams in all graphs of size at most n, assuming that $\mathcal{GK} = \lceil \log \log n \rceil$: note that such a global knowledge can be coded using $\mathcal{O}(\log \log \log n)$ bits. The algorithm works in a time polynomial in n and $|l_{min}|$, and it makes use of the building blocks introduced in the section Preliminaries.

In the sequel, we denote by G_n the maximal time complexity of procedure $\texttt{GROUP}(X_n, n, \rho)$ with $\rho \in \{0;1\}$ in all graphs of size at most n. We also denote by M_n the maximal time complexity of procedure $\texttt{MERGE}(X_n + G_n, n)$ in all graphs of size at most n. Note that according to Theorems 1 and 2, G_n and M_n exist and are polynomials in n.

In order to better describe the high level idea of our solution, let us first consider a situation that would be ideal to solve Byzantine gathering with a strong team and that would be as follows. Instead of assigning distinct labels to all agents, the adversary assigns to each of them just one bit $\rho \in \{0;1\}$, so that there are at least one good agent for which $\rho = 0$ and at least one good agent for which $\rho = 1$. Such a situation would clearly constitute an infringement of our model, but would allow the simple protocol described in Algorithm 1 to solve the problem in a time that is polynomial in n when $\mathcal{GK} = \lceil \log \log n \rceil$. Let us briefly explain why.

Algorithm 1 Algorithm executed by every good agent in the ideal situation.

1: Let ρ be the bit assigned to me by the adversary
2: Execute $\mathcal{A}(\rho)$
3: Declare that gathering is achieved

Algorithm 2 $\mathcal{A}(\rho)$ executed by a good agent.

1: $N \leftarrow 2^{(2^{\mathcal{GK}})}$
2: Execute $\texttt{EXPLO}(N)$
3: Execute $\texttt{GROUP}(X_N, N, \rho)$
4: Execute $\texttt{MERGE}(X_N + G_N, N)$

Algorithm 1 consists mainly of a call to $\mathcal{A}(\rho)$ that is given by Algorithm 2. Since $\mathcal{GK} = \lceil \log \log n \rceil$, we know that at line 1 of Algorithm 2, N is a polynomial upperbound on n, and the execution of $\texttt{EXPLO}(N)$ in a call to $\mathcal{A}(\rho)$ by the first woken-up good agent permits to visit every node of the graph and to wake up all dormant agents. As a result, the delay between the starting times of $\texttt{GROUP}(X_N, N, \rho)$ by any two good agents of the strong team is at most X_N. According to the properties of procedure GROUP (cf. Theorem 1) and the fact that the number of good agents is at least $5f^2 + 6f + 2 = ((5f + 2) - 1)(f + 1) + 1$, this guarantees in turn that the delay between the starting times of $\texttt{MERGE}(X_N + G_N, N)$ by any two good agents is at most $X_N + G_N$, and at least $4f + 2$ good agents start this procedure at the same time in the same node. Hence, in view of the properties of procedure MERGE (cf. Theorem 2), all good agents declare gathering is achieved at the same time in the same node after a polynomial number of rounds (w.r.t n) since the wake-up time of the earliest good agent.

Unfortunately, we are not in such an ideal situation. At first glance, one might argue that it is not really a problem because all agents are assigned distinct labels that are, after all, distinct binary strings. Thus, by ensuring that each good agent applies on its label the transformation given in Section 2, and then processes one by one each bit b_i of its transformed label by executing $\mathcal{A}(b_i)$, we can guarantee (with some minor technical adjustments) that the gathering of all good agents is done in time polynomial in n and $|l_{min}|$. Indeed, in view of Proposition 1 the conditions of the ideal situation are recreated when the agents process their j-th bits for some $j \leq 2|l_{min}| + 4$. Unfortunately it is not enough for our purpose. In fact, in the ideal situation, there is just one bit to process: thus, *de facto* every good agent knows that every good agent knows that gathering will be done at the end of this single process. However, it is no longer the case when the agents have to deal with sequences of bit processes: the good agents have a priori no mean to detect collectively and simultaneously

when they are gathered. It should be noted that if the agents knew f, we could use an existing algorithmic component (cf. [17]) allowing to solve f-Byzantine gathering if at some point some good agents detect the presence of a group of at least $2f+1$ agents in the network. Such a group is necessarily constructed during the sequence of bit processes given above, but again, it cannot be a priori detected as the agents do not know f or an upperbound on it. Hence, in our goal to optimize the amount of global knowledge, we need to implement a new strategy to allow the good agents to declare gathering achieved jointly and simultaneously. It is the purpose of the rest of this subsection.

To get all good agents declare simultaneously the gathering achieved, we want to reach a round in which every good agent knows that every good agent knows that gathering is done. So, let us return to our sequence of bit processes. As mentioned above, when a good agent has finished to read the first half of its transformed label – call such an agent *experienced* – it has the guarantee that the gathering of all good agents has been done at least once. Hence, when an experienced agent starts to process the second half of its transformed label, it actually knows an approximation of the number of good agents with a margin of error of f at the most. For the sake of convenience, let us consider that an experienced agent knows the exact number μ of good agents: the general case adds a slight level of complexity that is unnecessary to understand the intuition. So, each time an experienced agent completes the process of a bit in the second half of its transformed label, it is in a node containing less than μ agents or at least μ agents. In the first case, the experienced agent is sure that the gathering is not achieved. In the second case, the experienced agent is in doubt. In our solution, we build on this doubt. How do we do that? So far, each bit process was just made of one call to procedure $\mathcal{A}$: now at the end of each bit process, we add a waiting period of some prescribed length, followed by an extra step that consists in applying $\mathcal{A}$ again, but this time according to the following rule. If during the waiting period it has just done, an agent X was in a node containing, for a sufficiently long period, an agent pretending to be experienced and in doubt (this agent may be X itself), then agent X is said to be *optimistic* and the second step corresponds to the execution of $\mathcal{A}(0)$. Otherwise, agent X is said to be *pessimistic* and the second step corresponds to the execution of $\mathcal{A}(1)$.

If at least one good agent is optimistic within a given second step, then the gathering of all good agents is done at the end of this step. Indeed, through similar arguments of partition to those used for the ideal situation, we can show it is the case when at least another agent is pessimistic. However, it is also, more curiously, the case when there is no pessimistic agents at all. This is due in part to the fact that two good experienced agents cannot have been in doubt in two distinct nodes during the previous waiting period (otherwise, we would get a contradiction with the definition of μ). Thus, all good agents start $\mathcal{A}(0)$ from at most $f+1$ distinct nodes (as the Byzantine agents can mislead the good agents in at most f distinct nodes during the waiting period), which implies by the pigeonhole principle that at least $4f+2$ good agents start it from the same node. Combined with some other technical arguments, we can show that the conditions of Theorem 2 are fulfilled when the agents execute `MERGE` at the end of $\mathcal{A}(0)$, thereby guaranteeing again gathering of all good agents.

As a result, the addition of an extra step to each bit process gives us the following interesting property: when a good agent is optimistic at the beginning of a second step, at its end the gathering is done and, more importantly, the optimistic agent knows it because its existence ensures it. Note that, it is a great progress, but unfortunately it is not yet sufficient, particularly because the pessimistic agents do not have the same kind of guarantee. The way of remedying this is to repeat once more the same kind of algorithmic ingredient as above. More precisely, at the end of each second step, we add again a waiting period of

some prescribed length, followed by a third step that consists in applying $\mathcal{A}$ in the following manner. If during the waiting period it has just done, an agent X was in a node containing, for a sufficiently long period, an agent pretending to be optimistic, then the third step of agent X corresponds to the execution of $\mathcal{A}(0)$ and it becomes optimistic if it was not. Otherwise, the third step of agent X corresponds to the execution of $\mathcal{A}(1)$ and the agent stays pessimistic.

By doing so, we made a significant move forward. To understand why, we want to invite the reader to reconsider the case when there is at least one good agent that is optimistic at the beginning of a second step. As we have seen earlier, at the end of this second step, all good agents are necessarily gathered and every optimistic agent knows it. In view of the last changes made to our solution, when starting the third step, every good agent is then optimistic. As explained above the absence of pessimistic good agent is very helpful, and using here the same arguments, we are sure that when finishing the third step, all good agents are gathered and every good agent knows it because all of them are optimistic. Actually, it is even a little more subtle: the optimistic agents of the first generation (i.e., those that were already optimistic when starting the second step) know that the gathering is done and know that every good agent knows it. Concerning the optimistic agents of the second generation (i.e., those that became optimistic only when starting the third step), they just know that the gathering is done, but do not know whether the other agents know it or not. Recall that to get all good agents declare simultaneously the gathering achieved, we want to reach a round in which every good agent knows that every good agent knows that gathering is done. We are very close to such a consensus. To reach it, at the end of a third step, the optimistic agents of the first generation make themselves known to all agents. Note that if there were at least $f+1$ agents declaring to be optimistic agents of the first generation and if f was part of $\mathcal{GK}$, the consensus would be reached. Indeed, among the agents declaring to be optimistic of the first generation, at least one is necessarily good and every agent can notice it: at this point we can show that every good agent knows that every good agent knows that gathering is done.

However, the agents do not know f. That being said, at the end of a third step, note that an optimistic agent knowing that the gathering is done can compute an approximation $\tilde{f}$ of the number of Byzantine agents. More precisely, if the number of agents gathered in its node is p, the optimistic agent knows than the number of Byzantine agents cannot exceed $\tilde{f} = max\{y|(5y+1)(y+1)+1 \leq p\}$ according to the definition of a strong team. Based on this fact, we are saved. Indeed, our algorithm is designed in such a way that all good agents correctly declare the gathering is achieved in the same round after having computed the same approximation $\tilde{f}$ and noticed at least $\tilde{f}+1$ agents that claim being optimistic of the first generation during a third step. We show that such an event necessarily occurs before any agent finishes the $(4|l_{min}|+8)$-th bit process of its transformed label, which permits to obtain the promised polynomial complexity. This is where our feat of strength is: obtaining such a complexity with a small amount of global knowledge, while ensuring that the Byzantine agents cannot confuse the good agents in any way. Actually, our algorithm is judiciously orchestrated so that the only thing Byzantine agents can really do is just to accelerate the resolution of the problem.

▶ **Theorem 3.** *Assuming that* $\mathcal{GK} = \lceil \log \log n \rceil$*, Algorithm* `GATHER` *solves* f*-Byzantine gathering with every strong team in all graph of size at most* n*, and has a time complexity that is polynomial in* n *and* $|l_{min}|$*.*

4 The negative result

Algorithm GATHER introduced in the previous section uses the value $\lceil \log \log n \rceil$ as global knowledge, which can be coded with a binary string of size $\mathcal{O}(\log \log \log n)$. In this section, we show that, to solve Byzantine gathering with all strong teams, in all graph of size at most n, in a time polynomial in n and $|l_{min}|$, the order of magnitude of the size of knowledge used by our algorithm GATHER is optimal. More precisely, we have the following theorem.

▶ **Theorem 4.** *There is no algorithm solving f-Byzantine gathering with all strong teams for all f and in all graphs of size at most n, which is polynomial in n and $|l_{min}|$ and which uses a global knowledge of size $o(\log \log \log n)$.*

Proof. Suppose by contradiction that the theorem is false. Hence, there exists an algorithm Alg that solves f-Byzantine gathering with all strong teams for all f in all graphs of size at most n, which is polynomial in n and $|l_{min}|$ and which uses a global knowledge of size $o(\log \log \log n)$. The proof relies on the construction of a family $\mathcal{F}_n$ (for any $n \geq 4$) of initial instances with strong teams such that for each of them the graph size is at most n. Our goal is to prove that there is an instance from $\mathcal{F}_n$ for which algorithm Alg needs a global knowledge whose size does not belong to $o(\log \log \log n)$, which would be a contradiction with the definition of Alg. Let us first present the construction of an infinite sequence of instances $\mathcal{I} = I_0, I_1, I_2, \ldots, I_i, \ldots$ by induction on i. Instance I_0 consists of an oriented ring of 4 nodes (i.e., a ring in which at each node the edge going clockwise has port number 0 and the edge going anti-clockwise has port 1). In this ring, there is no Byzantine agent but there are two good agents labeled 0 and 1 that are placed in diametrically opposed nodes. All the agents in I_0 wake up at the same time.

Now let us describe the construction of instance I_i with $i \geq 1$ using some features of instance I_{i-1}. Let c be the smallest constant integer such that the time complexity of algorithm Alg is at most n^c from every instance made of a graph of size at most n with a strong team in which $|l_{min}| = 1$. Let μ_{i-1} and n_{i-1} be respectively the total number of agents in I_{i-1} and the number of nodes in the graph of I_{i-1}. Instance I_i consists of an oriented ring of $(n_{i-1})^{4c}$ nodes. In this ring an agent labeled 0 is placed on a node denoted by v_0. In each of both nodes that are adjacent to v_0, $(n_{i-1})^c \cdot \mu_{i-1}$ Byzantine agents are placed (which gives a total of $2(n_{i-1})^c \cdot \mu_{i-1}$ Byzantine agents). On the node that is diametrically opposed to v_0, enough good agents are placed in order to have a strong team. The way of assigning labels to all agents that are not at v_0 is arbitrary but respects the condition that initially no two agents share the same label. Finally, all the agents in I_i wake up at the same time. This closes the description of the construction of $\mathcal{I}$, for which we have the following claim.

Claim 1. For any two instances I_j and $I_{j'}$ of $\mathcal{I}$, algorithm Alg requires a distinct global knowledge.

Proof of Claim 1. Assume by contradiction that the claim does not hold for two instances I_j and $I_{j'}$ such that $j < j'$. Consider any execution EX_j of algorithm Alg from I_j. According to the construction of $\mathcal{I}$, we know that every agent is woken up at the first round of EX_j. We denote by $r_1, r_2, \ldots, r_k$ the sequence of consecutive rounds from the first round of EX_j to the round when all good agents declare that gathering is done. We also denote by G_i the group of agents (possibly empty) that are with the good agent labeled 0 at round r_i of EX_j. Now, using execution EX_j, let us describe a possible execution $EX_{j'}$ of algorithm Alg

from $I_{j'}$: this execution is designed in such a way that it will fool the good agent labeled 0 and will induce it into premature termination. According to the construction of $\mathcal{I}$, all the agents of $I_{j'}$ are woken up in the first round of $I_{j'}$ and all the good ones are executing algorithm `Alg`. In the first round of $EX_{j'}$ the agent labeled 0 is alone (as in the first round of EX_j). Then, for each $i \in 2, \ldots, k$, the good agent labeled 0 in $EX_{j'}$ meets a group of $|G_i|$ Byzantine agents whose the multiset of labels is exactly the same as the multiset of labels belonging to the agents of G_i in the ith round of EX_j. This is always possible in view of the fact that for each $i \in 1, \ldots, k$, $|G_i| \leq \mu_j$ and the Byzantine agents of $I_{j'}$ can choose to move by ensuring that in the ith round of $EX_{j'}$ it remains at least $(k-i) \cdot \mu_j$ Byzantine agents in the node adjacent to the one occupied by the agent labeled 0 in the clockwise direction (resp. anti-clockwise direction): indeed according to the construction of $I_{j'}$, in each of both nodes adjacent to the starting node of the good agent labeled 0, there are initially $(n_{j'-1})^c \cdot \mu_{j'-1} \geq k \cdot \mu_{j'-1}$ Byzantine agents, as $k \leq (n_j)^c \leq (n_{j'-1})^c$. Finally, if algorithm `Alg` prescribes some message exchange between agents during their meetings, then the Byzantine agents in execution $EX_{j'}$ give exactly the same information to 0, as the agents with respective labels in execution EX_j. Hence, from the point of view of agent 0, the first k rounds of EX_j look exactly identical to the first k rounds of $EX_{j'}$. This is due to the actions of Byzantine agents, the fact that all nodes in I_j and $I_{j'}$ look identical, and also because $k \leq (n_j)^c$ which implies that, regardless of the algorithm `Alg`, the agent labeled 0 cannot meet any good agent in the first k rounds of $EX_{j'}$ as the distance between agent 0 and any other good agent is initially at least $\frac{(n_{j'-1})^{4c}}{2} \geq \frac{(n_j)^{4c}}{2}$. Therefore, in the kth round of execution $EX_{j'}$, the good agent labeled 0 declares having met all good agents and stops, which is incorrect, since it has not met any good agent. This contradicts the definition of algorithm `Alg` and closes the proof of this claim.

Now, consider the largest x such that in each of the $x+1$ first instances $I_0, I_1, \ldots, I_x$ of $\mathcal{I}$, the graph size is at most n: these $x+1$ instances constitute family $\mathcal{F}_n$. In view of the construction of sequence $\mathcal{I}$ and the definition of x, we have $4^{((4c)^x)} \leq n < 4^{((4c)^{x+1})}$. Hence, x belongs to $\Omega(\log\log n)$. However, according to Claim 1, the global knowledge given to distinct instances in this family must be different. Hence, there is at least one instance of $\mathcal{F}_n$ for which algorithm `Alg` uses a global knowledge of size $\Omega(\log x)$: since $x \in \Omega(\log\log n)$, we have $\Omega(\log x) \in \Omega(\log\log\log n)$. This contradicts the fact that `Alg` uses a global knowledge of size $o(\log\log\log n)$ and proves the theorem. ◀

5 Conclusion

In this paper, we designed the first polynomial algorithm w.r.t n and $|l_{min}|$ allowing to gather all good agents in presence of Byzantine ones that can act in an unpredictable way and lie about their labels. Our algorithm works under the assumption that the team evolving in the network is strong i.e., the number of good agents is roughly at least quadratic in the number f of Byzantine agents. The required global knowledge $\mathcal{GK}$ is of size $\mathcal{O}(\log\log\log n)$, which is of optimal order of magnitude to get a time complexity that is polynomial in n and $|l_{min}|$ even with strong teams.

A natural open question that immediately comes to mind is to ask if we can do the same by reducing the ratio between the good agents and the Byzantine agents. For example, could it be still possible to solve the problem in polynomial time with a global knowledge of size $\mathcal{O}(\log\log\log n)$ if the number of good agents is at most $o(f^2)$? Note that the answer to this question may be negative but then may become positive with a little bit more global knowledge. Actually, we can even easily show that the answer is true if the agents are initially

given a complete map of the graph with all port numbers, and in which each node v is associated to the list of all labels of the good agents initially occupying node v. However, the size of $\mathcal{GK}$ is then huge as it belongs to $\Omega(n^2)$. In fact, in this case what is really interesting is to find the optimal size for $\mathcal{GK}$. This observation allows us to conclude with the following open problem that is more general and appealing.

What are the trade-offs among the ratio good/Byzantine agents, the time complexity and the amount of global knowledge to solve f-Byzantine gathering?

Bringing an exhaustive and complete answer to this question appears to be really challenging but would turn out to be a major step in our understanding of the problem.

References

1. Serge Abiteboul, Haim Kaplan, and Tova Milo. Compact labeling schemes for ancestor queries. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA.*, pages 547–556, 2001.
2. Noa Agmon and David Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM J. Comput.*, 36(1):56–82, 2006.
3. Steve Alpern. Rendezvous search: A personal perspective. *Operations Research*, 50(5):772–795, 2002.
4. Steve Alpern. *The theory of search games and rendezvous.* International Series in Operations Research and Management Science, Kluwer Academic Publishers, 2003.
5. Evangelos Bampas, Jurek Czyzowicz, Leszek Gasieniec, David Ilcinkas, and Arnaud Labourel. Almost optimal asynchronous rendezvous in infinite multidimensional grids. In *Distributed Computing, 24th International Symposium, DISC 2010, Cambridge, MA, USA, September 13-15, 2010. Proceedings*, pages 297–311, 2010.
6. Michael Barborak and Miroslaw Malek. The consensus problem in fault-tolerant computing. *ACM Comput. Surv.*, 25(2):171–220, 1993.
7. Sébastien Bouchard, Yoann Dieudonné, and Bertrand Ducourthial. Byzantine gathering in networks. *Distributed Computing*, 29(6):435–457, 2016.
8. Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. *SIAM J. Comput.*, 41(4):829–879, 2012.
9. Reuven Cohen, Pierre Fraigniaud, David Ilcinkas, Amos Korman, and David Peleg. Label-guided graph exploration by a finite automaton. *ACM Trans. Algorithms*, 4(4):42:1–42:18, 2008.
10. Andrew Collins, Jurek Czyzowicz, Leszek Gasieniec, and Arnaud Labourel. Tell me where I am so I can meet you sooner. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, pages 502–514, 2010.
11. Jurek Czyzowicz, Konstantinos Georgiou, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, Jaroslav Opatrny, and Sunil M. Shende. Search on a line by byzantine robots. In *27th International Symposium on Algorithms and Computation, ISAAC 2016, December 12-14, 2016, Sydney, Australia*, pages 27:1–27:12, 2016.
12. Jurek Czyzowicz, Adrian Kosowski, and Andrzej Pelc. How to meet when you forget: log-space rendezvous in arbitrary graphs. *Distributed Computing*, 25(2):165–178, 2012.
13. Jurek Czyzowicz, Andrzej Pelc, and Arnaud Labourel. How to meet asynchronously (almost) everywhere. *ACM Transactions on Algorithms*, 8(4):37, 2012.
14. Shantanu Das, Dariusz Dereniowski, Adrian Kosowski, and Przemyslaw Uznanski. Rendezvous of distance-aware mobile agents in unknown graphs. In *Structural Information and Communication Complexity - 21st International Colloquium, SIROCCO 2014, Takayama, Japan, July 23-25, 2014. Proceedings*, pages 295–310, 2014.

15 Xavier Défago, Maria Gradinariu, Stéphane Messika, and Philippe Raipin Parvédy. Fault-tolerant and self-stabilizing mobile robots gathering. In *Distributed Computing, 20th International Symposium, DISC 2006, Stockholm, Sweden, September 18-20, 2006, Proceedings*, pages 46–60, 2006.

16 Anders Dessmark, Pierre Fraigniaud, Dariusz R. Kowalski, and Andrzej Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, 2006.

17 Yoann Dieudonné, Andrzej Pelc, and David Peleg. Gathering despite mischief. *ACM Transactions on Algorithms*, 11(1):1, 2014.

18 Yoann Dieudonné, Andrzej Pelc, and Vincent Villain. How to meet asynchronously at polynomial cost. *SIAM J. Comput.*, 44(3):844–867, 2015.

19 Pierre Fraigniaud, Cyril Gavoille, David Ilcinkas, and Andrzej Pelc. Distributed computing with advice: information sensitivity of graph coloring. *Distributed Computing*, 21(6):395–403, 2009.

20 Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Tree exploration with advice. *Inf. Comput.*, 206(11):1276–1287, 2008.

21 Pierre Fraigniaud and Andrzej Pelc. Deterministic rendezvous in trees with little memory. In *Distributed Computing, 22nd International Symposium, DISC 2008, Arcachon, France, September 22-24, 2008. Proceedings*, pages 242–256, 2008.

22 Pierre Fraigniaud and Andrzej Pelc. Delays induce an exponential memory gap for rendezvous in trees. *ACM Transactions on Algorithms*, 9(2):17, 2013.

23 Samuel Guilbault and Andrzej Pelc. Gathering asynchronous oblivious agents with local vision in regular bipartite graphs. *Theor. Comput. Sci.*, 509:86–96, 2013.

24 Taisuke Izumi, Samia Souissi, Yoshiaki Katayama, Nobuhiro Inuzuka, Xavier Défago, Koichi Wada, and Masafumi Yamashita. The gathering problem for two oblivious robots with unreliable compasses. *SIAM J. Comput.*, 41(1):26–46, 2012.

25 Michal Katz, Nir A. Katz, Amos Korman, and David Peleg. Labeling schemes for flow and connectivity. *SIAM J. Comput.*, 34(1):23–40, 2004.

26 Dariusz R. Kowalski and Adam Malinowski. How to meet in anonymous network. *Theor. Comput. Sci.*, 399(1-2):141–156, 2008.

27 Evangelos Kranakis, Danny Krizanc, Euripides Markou, Aris Pagourtzis, and Felipe Ramírez. Different speeds suffice for rendezvous of two agents on arbitrary graphs. In *SOFSEM 2017: Theory and Practice of Computer Science - 43rd International Conference on Current Trends in Theory and Practice of Computer Science, Limerick, Ireland, January 16-20, 2017, Proceedings*, pages 79–90, 2017.

28 Evangelos Kranakis, Danny Krizanc, and Sergio Rajsbaum. Mobile agent rendezvous: A survey. In *Structural Information and Communication Complexity, 13th International Colloquium, SIROCCO 2006, Chester, UK, July 2-5, 2006, Proceedings*, pages 1–9, 2006.

29 Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

30 Gianluca De Marco, Luisa Gargano, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, and Ugo Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theor. Comput. Sci.*, 355(3):315–326, 2006.

31 Avery Miller and Andrzej Pelc. Fast rendezvous with advice. *Theor. Comput. Sci.*, 608:190–198, 2015.

32 Avery Miller and Andrzej Pelc. Time versus cost tradeoffs for deterministic rendezvous in networks. *Distributed Computing*, 29(1):51–64, 2016.

33 Nicolas Nisse and David Soguet. Graph searching with advice. *Theor. Comput. Sci.*, 410(14):1307–1318, 2009.

34 Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

35 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.

36 Thomas Schelling. *The Strategy of Conflict*. Oxford University Press, Oxford, 1960.

37 Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Transactions on Algorithms*, 10(3):12, 2014.

38 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.

Temporal Vertex Cover with a Sliding Time Window

Eleni C. Akrida
Department of Computer Science, University of Liverpool, UK
Eleni.Akrida@liverpool.ac.uk
https://orcid.org/0000-0002-1126-1623

George B. Mertzios
Department of Computer Science, Durham University, UK
George.Mertzios@durham.ac.uk
https://orcid.org/0000-0001-7182-585X

Paul G. Spirakis
Department of Computer Science, University of Liverpool, UK
Department of Computer Engineering & Informatics, University of Patras, Greece
P.Spirakis@liverpool.ac.uk
https://orcid.org/0000-0001-5396-3749

Viktor Zamaraev
Department of Computer Science, Durham University, UK
Viktor.Zamaraev@durham.ac.uk
https://orcid.org/0000-0001-5755-4141

Abstract

Modern, inherently dynamic systems are usually characterized by a network structure, i.e. an underlying graph topology, which is subject to discrete changes over time. Given a static underlying graph G, a temporal graph can be represented via an assignment of a set of integer time-labels to every edge of G, indicating the discrete time steps when this edge is active. While most of the recent theoretical research on temporal graphs has focused on the notion of a temporal path and other "path-related" temporal notions, only few attempts have been made to investigate "non-path" temporal graph problems. In this paper, motivated by applications in sensor and in transportation networks, we introduce and study two natural temporal extensions of the classical problem Vertex Cover. In our first problem, Temporal Vertex Cover, the aim is to cover every edge at least once during the lifetime of the temporal graph, where an edge can only be covered by one of its endpoints at a time step when it is active. In our second, more pragmatic variation Sliding Window Temporal Vertex Cover, we are also given a natural number Δ, and our aim is to cover every edge at *least once* at *every* Δ *consecutive* time steps. In both cases we wish to minimize the total number of "vertex appearances" that are needed to cover the whole graph. We present a thorough investigation of the computational complexity and approximability of these two temporal covering problems. In particular, we provide strong hardness results, complemented by various approximation and exact algorithms. Some of our algorithms are polynomial-time, while others are asymptotically almost optimal under the Exponential Time Hypothesis (ETH) and other plausible complexity assumptions.

2012 ACM Subject Classification Mathematics of computing → Graph theory, Mathematics of computing → Graph algorithms

Keywords and phrases Temporal networks, temporal vertex cover, APX-hard, approximation algorithm, Exponential Time Hypothesis

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.148

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 148; pp. 148:1–148:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version https://arxiv.org/abs/1802.07103

Funding Partially supported by NeST initiative of the School of EEE and CS at the University of Liverpool and by the EPSRC Grants EP/P020372/1 and EP/P02002X/1.

1 Introduction and Motivation

A great variety of both modern and traditional networks are inherently dynamic, in the sense that their link availability varies over time. Information and communication networks, social networks, transportation networks, and several physical systems are only a few examples of networks that change over time [18, 27]. The common characteristic in all these application areas is that the network structure, i.e. the underlying graph topology, is subject to *discrete changes over time*. In this paper we adopt a simple and natural model for time-varying networks which is given with time-labels on the edges of a graph, while the vertex set remains unchanged. This formalism originates in the foundational work of Kempe et al. [20].

▶ **Definition 1** (temporal graph)**.** A *temporal graph* is a pair (G, λ), where $G = (V, E)$ is an underlying (static) graph and $\lambda : E \to 2^{\mathbb{N}}$ is a *time-labeling* function which assigns to every edge of G a set of discrete-time labels.

For every edge $e \in E$ in the underlying graph G of a temporal graph (G, λ), $\lambda(e)$ denotes the set of time slots at which e is *active* in (G, λ). Due to its vast applicability in many areas, this notion of temporal graphs has been studied from different perspectives under various names such as *time-varying* [1, 14, 29], *evolving* [4, 10, 13], *dynamic* [7, 15], and *graphs over time* [24]; for a recent attempt to integrate existing models, concepts, and results from the distributed computing perspective see the survey papers [5–7] and the references therein. Data analytics on temporal networks have also been very recently studied in the context of summarizing networks that represent sports teams' activity data to discover recurring strategies and understand team tactics [22], as well as extracting patterns from interactions between groups of entities in a social network [21].

Motivated by the fact that, due to causality, information in temporal graphs can "flow" only along sequences of edges whose time-labels are increasing, most temporal graph parameters and optimization problems that have been studied so far are based on the notion of temporal paths and other "path-related" notions, such as temporal analogues of distance, diameter, reachability, exploration, and centrality [2, 3, 12, 25, 26]. In contrast, only few attempts have been made to define "non-path" temporal graph problems. Motivated by the contact patterns among high-school students, Viard et al. [31, 32], and later Himmel et al. [17], introduced and studied Δ-cliques, an extension of the concept of cliques to temporal graphs, in which all vertices interact with each other at least once every Δ consecutive time steps within a given time interval.

In this paper we introduce and study two natural temporal extensions of the problem Vertex Cover in static graphs, which take into account the dynamic nature of the network. In the first and simpler of these extensions, namely Temporal Vertex Cover (for short, TVC), every edge e has to be "covered" at least once during the lifetime T of the network (by one of its endpoints), and this must happen at a time step t when e is active. The goal is then to cover all edges with the minimum total number of such "vertex appearances". On the other hand, in many real-world applications where scalability is important, the lifetime T can be arbitrarily large but the network still needs to remain sufficiently covered. In such cases, as well as in safety-critical systems (e.g. in military applications), it may not be satisfactory

enough that an edge is covered just *once* during the *whole lifetime* of the network. Instead, every edge must be covered at least once within *every small* Δ-*window* of time (for an appropriate value of Δ), regardless of how large the lifetime is; this gives rise to our second optimization problem, namely Sliding Window Temporal Vertex Cover (for short, SW-TVC). Formal definitions of our problems TVC and SW-TVC are given in Section 2.

Our two temporal extensions of Vertex Cover are motivated by applications in sensor networks and in transportation networks. In particular, several works in the field of sensor networks considered problems of placing sensors to cover a whole area or multiple critical locations, e.g. for reasons of surveillance. Such studies usually wish to minimize the number of sensors used or the total energy required [11, 16, 23, 28, 33]. Our temporal vertex cover notions are an abstract way to economically meet such covering demands as time progresses.

To further motivate the questions raised in this work, consider a network whose links represent transporting facilities which are not always available, while the availability schedule per link is known in advance. We wish to check each transporting facility and certify "OK" at least once per facility during every (reasonably small) window of time. It is natural to assume that the checking is done in the presence of an inspecting agent at an endpoint of the link (i.e. on a vertex), since such vertices usually are junctions with local offices. The agent can inspect more than one link at the same day, provided that these links share this vertex and that they are all alive (i.e. operating) at that day. Notice that the above is indeed an application drawn from real-life, as regular checks in roads and trucks are paramount for the correct operation of the transporting sector, according to both the European Commission[1] and the American Public Transportation Association[2].

1.1 Our contribution

In this paper we present a thorough investigation of the complexity and approximability of the problems Temporal Vertex Cover (TVC) and Sliding Window Temporal Vertex Cover (SW-TVC) on temporal graphs. We first prove in Section 3 that Set Cover is equivalent to a special case of TVC on star temporal graphs (i.e. when the underlying graph G is a star), which immediately provides several complexity and algorithmic consequences for TVC. In particular, TVC remains NP-complete even on star temporal graphs, and it does not admit a polynomial-time $(1-\varepsilon)\ln n$-approximation algorithm, unless NP has $n^{O(\log\log n)}$-time deterministic algorithms. On the positive side, TVC on star temporal graphs with n vertices can be $(H_{n-1}-\frac{1}{2})$-approximated in polynomial time, where $H_n=\sum_{i=1}^{n}\frac{1}{i}\approx\ln n$ is the nth harmonic number. Similar equivalence with Hitting Set yields that for any $\varepsilon<1$, TVC on star temporal graphs cannot be optimally solved in $O(2^{\varepsilon n})$ time, assuming the Strong Exponential Time Hypothesis (SETH). We complement these results by showing that TVC on general temporal graphs admits a polynomial-time randomized approximation algorithm with expected ratio $O(\ln n)$.

In Section 4 and in the reminder of the paper we deal with our second problem, SW-TVC. We prove in Section 4.1 a strong complexity lower bound on arbitrary temporal graphs. More specifically we prove that, for *any* (arbitrarily growing) functions $f:\mathbb{N}\to\mathbb{N}$ and $g:\mathbb{N}\to\mathbb{N}$,

[1] According to the European Commission (see `https://ec.europa.eu/transport/road_safety/topics/vehicles/inspection_en`), "roadworthiness checks (such as on-the-spot roadside inspections and periodic checks) not only make sure your vehicle is working properly, they are also important for environmental reasons and for ensuring fair competition in the transport sector".

[2] According to the American Public Transportation Association (see `http://www.apta.com/resources/standards/Documents/APTA-RT-VIM-RP-019-03.pdf` "developing minimum inspection, maintenance, testing and alignment procedures maintains rail transit trucks in a safe and reliable operating condition".

there exists a constant $\varepsilon \in (0,1)$ such that SW-TVC cannot be solved in $f(T) \cdot 2^{\varepsilon n \cdot g(\Delta)}$ time, assuming the Exponential Time Hypothesis (ETH). This ETH-based lower bound turns out to be asymptotically almost tight, as we present an exact dynamic programming algorithm with running time $O(T\Delta(n+m) \cdot 2^{n(\Delta+1)})$. This worst-case running time can be significantly improved in certain special temporal graph classes. In particular, when the "snapshot" of (G,λ) at every time step has vertex cover number bounded by k, the running time becomes $O(T\Delta(n+m) \cdot n^{k(\Delta+1)})$. That is, when Δ is a constant, this algorithm is polynomial in the input size on temporal graphs with bounded vertex cover number at every time step. Notably, when every snapshot is a star (i.e. a superclass of the star temporal graphs studied in Section 3) the running time of the algorithm is $O(T\Delta(n+m) \cdot 2^{\Delta})$.

In Section 5 we prove strong inapproximability results for SW-TVC even when restricted to temporal graphs with length $\Delta = 2$ of the sliding window. In particular, we prove that this problem is APX-hard (and thus does not admit a *Polynomial Time Approximation Scheme (PTAS)*, unless P = NP), even when $\Delta = 2$, the maximum degree in the underlying graph G is at most 3, and every connected component at every graph snapshot has at most 7 vertices. Finally, in Section 6 we provide a series of approximation algorithms for the general SW-TVC problem, with respect to various incomparable temporal graph parameters. In particular, we provide polynomial-time approximation algorithms with approximation ratios (i) $O(\ln n + \ln \Delta)$, (ii) $2k$, where k is the maximum number of times that each edge can appear in a sliding Δ time window (thus implying a ratio of 2Δ in the general case), (iii) d, where d is the maximum vertex degree at every snapshot of (G,λ). Note that, for $d = 1$, the latter result implies that SW-TVC can be optimally solved in polynomial time whenever every snapshot of (G,λ) is a matching.

2 Preliminaries and notation

A theorem proving that a problem is NP-hard does not provide much information about how efficiently (although not polynomially, unless P = NP) this problem can be solved. In order to prove some useful complexity lower bounds, we mostly need to rely on some complexity hypothesis that is stronger than"P $\neq$ NP". The *Exponential Time Hypothesis (ETH)* is one of the established and most well-known such complexity hypotheses.

▶ **Exponential Time Hypothesis** (ETH [19]). *There exists an $\varepsilon < 1$ such that* 3SAT *cannot be solved in $O(2^{\varepsilon n})$ time, where n is the number of variables in the input 3-CNF formula.*

Given a (static) graph G, we denote by $V(G)$ and $E(G)$ the sets of its vertices and edges, respectively. An edge between two vertices u and v of G is denoted by uv, and in this case u and v are said to be *adjacent* in G. The maximum label assigned by λ to an edge of G, called the *lifetime* of (G,λ), is denoted by $T(G,\lambda)$, or simply by T when no confusion arises. That is, $T(G,\lambda) = \max\{t \in \lambda(e) : e \in E\}$. For every $i,j \in \mathbb{N}$, where $i \leq j$, we denote $[i,j] = \{i, i+1, \ldots, j\}$. Throughout the paper we consider temporal graphs with *finite lifetime* T, and we refer to each integer $t \in [1,T]$ as a *time slot* of (G,λ). The *instance* (or *snapshot*) of (G,λ) *at time* t is the static graph $G_t = (V, E_t)$, where $E_t = \{e \in E : t \in \lambda(e)\}$. For every $i,j \in [1,T]$, where $i \leq j$, we denote by $(G,\lambda)|_{[i,j]}$ the restriction of (G,λ) to the time slots $i, i+1, \ldots, j$, i.e. $(G,\lambda)|_{[i,j]}$ is the sequence of the instances $G_i, G_{i+1}, \ldots, G_j$. We assume in the remainder of the paper that every edge of G appears in at least one time slot until T, namely $\bigcup_{t=1}^{T} E_t = E$.

Although some optimization problems on temporal graphs may be hard to solve in the worst case, an optimal solution may be efficiently computable when the input temporal

graph (G, λ) has special properties, i.e. if (G, λ) belongs to a special *temporal graph class* (or *time-varying graph class* [5, 7]). To specify a temporal graph class we can restrict (a) the *underlying topology* G, or (b) the *time-labeling* λ, i.e. the temporal pattern in which the time-labels appear, or both.

▶ **Definition 2.** Let (G, λ) be a temporal graph and let $\mathcal{X}$ be a class of (static) graphs. If $G \in \mathcal{X}$ then (G, λ) is an $\mathcal{X}$ *temporal graph*. On the other hand, if $G_i \in \mathcal{X}$ for every $i \in [1, T]$, then (G, λ) is an *always* $\mathcal{X}$ *temporal graph*.

In the remainder of the paper we denote by $n = |V|$ and $m = |E|$ the number of vertices and edges of the underlying graph G, respectively, unless otherwise stated. Furthermore, unless otherwise stated, we assume that the labeling λ is arbitrary, i.e. (G, λ) is given with an explicit list of labels for every edge. That is, the *size* of the input temporal graph (G, λ) is $O\left(|V| + \sum_{t=1}^{T} |E_t|\right) = O(n + mT)$. In other cases, where λ is more restricted, e.g. if λ is periodic or follows another specific temporal pattern, there may exist more succinct representations of the input temporal graph.

For every $u \in V$ and every time slot t, we denote the *appearance of vertex u at time t* by the pair (u, t). That is, every vertex u has T different appearances (one for each time slot) during the lifetime of (G, λ). Similarly, for every vertex subset $S \subseteq V$ and every time slot t we denote the *appearance of set S at time t* by (S, t). With a slight abuse of notation, we write $(S, t) = \bigcup_{v \in S}(v, t)$. A *temporal vertex subset* of (G, λ) is a set $\mathcal{S} \subseteq \{(v, t) : v \in V, 1 \leq t \leq T\}$ of vertex appearances in (G, λ). Given a temporal vertex subset $\mathcal{S}$, for every time slot $t \in [1, T]$ we denote by $\mathcal{S}_t = \{(v, t) : (v, t) \in \mathcal{S}\}$ the set of all vertex appearances in $\mathcal{S}$ at the time slot t. Similarly, for any pair of time slots $i, j \in [1, T]$, where $i \leq j$, $\mathcal{S}|_{[i,j]}$ is the restriction of the vertex appearances of $\mathcal{S}$ within the time slots $i, i + 1, \ldots, j$. Note that the *cardinality* of the temporal vertex subset $\mathcal{S}$ is $|\mathcal{S}| = \sum_{1 \leq t \leq T} |\mathcal{S}_t|$.

2.1 Temporal Vertex Cover

Let $\mathcal{S}$ be a temporal vertex subset of (G, λ). Let $e = uv \in E$ be an edge of the underlying graph G and let (w, t) be a vertex appearance in $\mathcal{S}$. We say that vertex w *covers* the edge e if $w \in \{u, v\}$, i.e. w is an endpoint of e; in that case, edge e is *covered* by vertex w. Furthermore we say that the vertex appearance (w, t) *temporally covers* the edge e if (i) w covers e and (ii) $t \in \lambda(e)$, i.e. the edge e is *active* during the time slot t; in that case, edge e is *temporally covered* by the vertex appearance (w, t). We now introduce the notion of a *temporal vertex cover* and the optimization problem Temporal Vertex Cover.

▶ **Definition 3.** Let (G, λ) be a temporal graph. A *temporal vertex cover* of (G, λ) is a temporal vertex subset $\mathcal{S} \subseteq \{(v, t) : v \in V, 1 \leq t \leq T\}$ of (G, λ) such that every edge $e \in E$ is *temporally covered* by at least one vertex appearance (w, t) in $\mathcal{S}$.

Temporal Vertex Cover (TVC)
Input: A temporal graph (G, λ).
Output: A temporal vertex cover $\mathcal{S}$ of (G, λ) with the smallest cardinality $\lvert\mathcal{S}\rvert$.

Note that TVC is a natural temporal extension of the problem Vertex Cover on static graphs. In fact, Vertex Cover is the special case of TVC where $T = 1$. Thus TVC is clearly NP-complete, as it also trivially belongs to NP.

2.2 Sliding Window Temporal Vertex Cover

In the notion of a temporal vertex cover given in Section 2.1, the requirement is that every edge is temporally covered at least once during the lifetime T of the input temporal graph (G, λ). On the other hand, in many real-world applications where scalability is important, the lifetime T can be arbitrarily large. In such cases it may not be satisfactory enough that an edge is temporally covered just *once* during the whole lifetime of the temporal graph. Instead, in such cases it makes sense that every edge is temporally covered by some vertex appearance at least once during *every small period* Δ of time, regardless of how large the lifetime T is. Motivated by this, we introduce in this section a natural *sliding window* variant of the TVC problem, which offers a greater scalability of the solution concept.

For every time slot $t \in [1, T - \Delta + 1]$, we define the *time window* $W_t = [t, t + \Delta - 1]$ as the sequence of the Δ consecutive time slots $t, t + 1, \ldots, t + \Delta - 1$. Furthermore we denote by $E[W_t] = \bigcup_{i \in W_t} E_i$ the union of all edges appearing at least once in the time window W_t. Finally we denote by $\mathcal{S}[W_t] = \{(v, i) \in \mathcal{S} : i \in W_t\}$ the restriction of the temporal vertex subset $\mathcal{S}$ to the window W_t. We are now ready to introduce the notion of a *sliding Δ-window temporal vertex cover* and the optimization problem Sliding Window Temporal Vertex Cover.

▶ **Definition 4.** Let (G, λ) be a temporal graph with lifetime T and let $\Delta \leq T$. A *sliding Δ-window temporal vertex cover* of (G, λ) is a temporal vertex subset $\mathcal{S} \subseteq \{(v, t) : v \in V, 1 \leq t \leq T\}$ of (G, λ) such that, for every time window W_t and for every edge $e \in E[W_t]$, e is *temporally covered* by at least one vertex appearance (w, t) in $\mathcal{S}[W_t]$.

Sliding Window Temporal Vertex Cover (SW-TVC)

Input: A temporal graph (G, λ) with lifetime T, and an integer $\Delta \leq T$.
Output: A sliding Δ-window temporal vertex cover $\mathcal{S}$ of (G, λ) with the smallest cardinality $|\mathcal{S}|$.

Whenever the parameter Δ is a fixed constant, we will refer to the above problem as the Δ-TVC (i.e. Δ is now a part of the problem name). Note that the problem TVC defined in Section 2.1 is the special case of SW-TVC where $\Delta = T$, i.e. where there is only one Δ-window in the whole temporal graph. Another special case[3] of SW-TVC is the problem 1-TVC, whose optimum solution is obtained by iteratively solving the (static) problem Vertex Cover on each of the T static instances of (G, λ); thus 1-TVC fails to fully capture the time dimension in temporal graphs.

3 Hardness and approximability of TVC

In this section we investigate the complexity of Temporal Vertex Cover (TVC). First we prove in Section 3.1 that TVC on star temporal graphs is equivalent to both Set Cover and Hitting Set, and derive several complexity and algorithmic consequences for TVC.

In Section 3.2 we use randomized rounding technique to prove that TVC on general temporal graphs admits a polynomial-time randomized approximation algorithm with expected ratio $O(\ln n)$. This result is complemented by our results in Section 6.1 where we prove that SW-TVC (and thus also TVC) can be deterministically approximated with ratio $H_{2n\Delta} - \frac{1}{2} \approx \ln n + \ln 2\Delta - \frac{1}{2}$ in polynomial time.

[3] The problem 1-TVC has already been investigated under the name "evolving vertex cover" in the context of maintenance algorithms in dynamic graphs [8]; similar "evolving" variations of other graph covering problems have also been considered, e.g. the "evolving dominating set" [6].

3.1 Hardness on star temporal graphs

In the next theorem we reduce Set Cover to TVC on star temporal graphs, and vice versa. Our hardness results are complemented in Theorem 6 by reducing from Hitting Set.

▶ **Theorem 5.** TVC *on star temporal graphs is NP-complete and it admits a polynomial-time* $(H_{n-1} - \frac{1}{2})$*-approximation algorithm. Furthermore, for any* $\varepsilon > 0$, TVC *on star temporal graphs does not admit any polynomial-time* $(1-\varepsilon)\ln n$*-approximation algorithm, unless NP has* $n^{O(\log\log n)}$*-time deterministic algorithms.*

▶ **Theorem 6.** *For every* $\varepsilon < 1$, TVC *on star temporal graphs cannot be optimally solved in* $O(2^{\varepsilon n})$ *time, unless the Strong Exponential Time Hypothesis (SETH) fails.*

3.2 A randomized rounding algorithm for TVC

In this section we provide a linear programming relaxation of TVC, and then, with the help of a randomized rounding technique, we construct a feasible solution whose expected size is within a factor of $O(\ln n)$ of the optimal size.

▶ **Theorem 7.** *There exists a polynomial-time randomized approximation algorithm for* TVC *with expected approximation factor* $O(\ln n)$.

4 An almost tight algorithm for SW-TVC

In this section we investigate the complexity of Sliding Window Temporal Vertex Cover (SW-TVC). First we prove in Section 4.1 a strong lower bound on the complexity of optimally solving this problem on arbitrary temporal graphs. More specifically we prove that, for *any* (arbitrarily growing) functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$, there exists a constant $\varepsilon \in (0,1)$ such that SW-TVC cannot be solved in $f(T) \cdot 2^{\varepsilon n \cdot g(\Delta)}$ time, assuming the Exponential Time Hypothesis (ETH). This ETH-based lower bound turns out to be asymptotically almost tight. In fact, we present in Section 4.2 an exact dynamic programming algorithm for SW-TVC whose running time on an arbitrary temporal graph is $O(T\Delta(n+m)\cdot 2^{n(\Delta+1)})$, which is asymptotically almost optimal, assuming ETH. In Section 4.3 we prove that our algorithm can be refined so that, when the vertex cover number of each snapshot G_i is bounded by a constant k, the running time becomes $O(T\Delta(n+m)\cdot n^{k(\Delta+1)})$. That is, when Δ is a constant, this algorithm is polynomial in the input size on temporal graphs with bounded vertex cover number at every slot. Notably, for the class of always star temporal graphs (i.e. a superclass of the star temporal graphs studied in Section 3.1) the running time of the algorithm is $O(T\Delta(n+m)\cdot 2^{\Delta})$.

4.1 A complexity lower bound

In the the following theorem we prove a strong ETH-based lower bound for SW-TVC. This lower bound is asymptotically almost tight, as we present in Section 4.2 a dynamic programming algorithm for SW-TVC with running time $O(T\Delta(n+m)\cdot 2^{n\Delta})$, where n and m are the numbers of vertices and edges in the underlying graph G, respectively.

▶ **Theorem 8.** *For* any *two (arbitrarily growing) functions* $f, g : \mathbb{N} \to \mathbb{N}$*, there exists a constant* $\varepsilon \in (0,1)$ *such that* SW-TVC *cannot be solved in* $f(T)\cdot 2^{\varepsilon n \cdot g(\Delta)}$ *time assuming ETH, where* n *is the number of vertices in the underlying graph* G *of the temporal graph.*

4.2 An exact dynamic programming algorithm

The main idea of our dynamic programming algorithm for SW-TVC is to scan the temporal graph from left to right with respect to time (i.e. to scan the snapshots G_i increasingly on i), and at every time slot to consider all possibilities for the vertex appearances at the previous Δ time slots. Let (G, λ) be a temporal graph with n vertices and lifetime T, and let $\Delta \leq T$. For every $t = 1, 2, \ldots, T - \Delta + 1$ and every Δ-tuple of vertex subsets $A_1, \ldots, A_\Delta$ of G, we define $f(t; A_1, A_2, \ldots, A_\Delta)$ to be the smallest cardinality of a sliding Δ-window temporal vertex cover $\mathcal{S}$ of $(G, \lambda)|_{[1,t+\Delta-1]}$, such that $\mathcal{S}_t = (A_1, t)$, $\mathcal{S}_{t+1} = (A_2, t+1)$, $\ldots$, $\mathcal{S}_{t+\Delta-1} = (A_\Delta, t+\Delta-1)$. If there exists no sliding Δ-window temporal vertex cover $\mathcal{S}$ of $(G, \lambda)|_{[1,t+\Delta-1]}$ with these prescribed vertex appearances in the time slots $t, t+1, \ldots, t+\Delta-1$, then we define $f(t; A_1, A_2, \ldots, A_\Delta) = \infty$. Note that, once we have computed all possible values of the function $f(\cdot)$, then the optimum solution of SW-TVC on (G, λ) has cardinality

$$\mathrm{OPT}_{\text{SW-TVC}}(G, \lambda) = \min_{A_1, A_2, \ldots, A_\Delta \subseteq V} \{f(T - \Delta + 1; A_1, A_2, \ldots, A_\Delta)\}. \tag{1}$$

▶ **Lemma 9.** *Let (G, λ) be a temporal graph, where $G = (V, E)$. Let $2 \leq t \leq T - \Delta + 1$ and let $A_1, A_2, \ldots A_\Delta$ be a Δ-tuple of vertex subsets of the underlying graph G. Suppose that $\bigcup_{i=1}^{\Delta}(A_i, t + i - 1)$ is a temporal vertex cover of $(G, \lambda)|_{[t,t+\Delta-1]}$. Then*

$$f(t; A_1, A_2, \ldots, A_\Delta) = |A_\Delta| + \min_{X \subseteq V} \{f(t-1; X, A_1, \ldots, A_{\Delta-1})\}. \tag{2}$$

Using the recursive computation of Lemma 9, we are now ready to present Algorithm 1 for computing the value of an optimal solution of SW-TVC on a given arbitrary temporal graph (G, λ). Note that Algorithm 1 can be easily modified such that it also computes the actual optimum solution of SW-TVC (instead of only its optimum cardinality). The proof of correctness and running time analysis of Algorithm 1 are given in the next theorem.

Algorithm 1 SW-TVC.

Input: A temporal graph (G, λ) with lifetime T, where $G = (V, E)$, and a natural $\Delta \leq T$.
Output: The smallest cardinality of a sliding Δ-window temporal vertex cover in (G, λ).

```
1: for t = 1 to T − Δ + 1 do
2:   for all A_1, A_2, ..., A_Δ ⊆ V do
3:     if ⋃_{i=1}^{Δ}(A_i, t + i − 1) is a temporal vertex cover of (G, λ)|_[t,t+Δ−1] then
4:       if t = 1 then
5:         f(t; A_1, A_2, ..., A_Δ) ← Σ_{i=1}^{Δ} |A_i|
6:       else
7:         f(t; A_1, A_2, ..., A_Δ) ← |A_Δ| + min_{X⊆V} {f(t − 1; X, A_1, ..., A_{Δ−1})}
8:     else
9:       f(t; A_1, A_2, ..., A_Δ) ← ∞
10: return min_{A_1,...,A_Δ⊆V} {f(T − Δ + 1; A_1, ..., A_Δ)}
```

▶ **Theorem 10.** *Let (G, λ) be a temporal graph, where $G = (V, E)$ has n vertices and m edges. Let T be its lifetime and let Δ be the length of the sliding window. Algorithm 1 computes in $O(T\Delta(n + m) \cdot 2^{n(\Delta+1)})$ time the value of an optimal solution of* SW-TVC *on (G, λ).*

4.3 Always bounded vertex cover number temporal graphs

Let k be a constant and let $\mathcal{C}_k$ be the class of graphs with the vertex cover number at most k. The next theorem follows now from the analysis of Theorem 10.

▶ **Theorem 11.** SW-TVC *on always $\mathcal{C}_k$ temporal graphs can be solved in* $O(T\Delta(n+m)\cdot n^{k(\Delta+1)})$ *time.*

In particular, in the special, yet interesting, case of always star temporal graphs, our search at every step reduces to just one binary choice for each of the previous Δ time slots, of whether to include the central vertex of a star in a snapshot or not. Hence we have the following theorem as a direct implication of Theorem 11.

▶ **Theorem 12.** SW-TVC *on always star temporal graphs can be solved in* $O(T\Delta(n+m)\cdot 2^{\Delta})$ *time.*

5 Approximation hardness of 2-TVC

In this section we study the complexity of Δ-TVC where Δ is constant. We start with an intuitive observation that, for every fixed Δ, the problem $(\Delta+1)$-TVC is at least as hard as Δ-TVC. Indeed, let $\mathcal{A}$ be an algorithm that computes a minimum-cardinality sliding $(\Delta+1)$-window temporal vertex cover of (G,λ). It is easy to see that a minimum-cardinality sliding Δ-window temporal vertex cover of (G,λ) can also be computed using $\mathcal{A}$, if we amend the input temporal graph by inserting one edgeless snapshot after every Δ consecutive snapshots of (G,λ).

Since the 1-TVC problem is equivalent to solving T instances of Vertex Cover (on static graphs), the above reduction demonstrates in particular that, for any natural Δ, Δ-TVC is at least as hard as Vertex Cover. Therefore, if Vertex Cover is hard for a class $\mathcal{X}$ of static graphs, then Δ-TVC is also hard for the class of always $\mathcal{X}$ temporal graphs. In this section, we show that the converse is not true. Namely, we reveal a class $\mathcal{X}$ of graphs, for which Vertex Cover can be solved in *linear* time, but 2-TVC is NP-hard on always $\mathcal{X}$ temporal graphs. In fact, we show the even stronger result that 2-TVC is APX-hard (and thus does not admit a PTAS, unless P = NP) on always $\mathcal{X}$ temporal graphs.

To prove the main result (in Theorem 14) we start with an auxiliary lemma, showing that Vertex Cover is APX-hard on the class $\mathcal{Y}$ of graphs which can be obtained from a cubic graph by subdividing every edge exactly 4 times.

▶ **Lemma 13.** Vertex Cover *is APX-hard on* $\mathcal{Y}$.

Let now $\mathcal{X}$ be the class of graphs whose connected components are induced subgraphs of the graph obtained from the star with three leaves by subdividing each of its edges exactly once. Clearly, Vertex Cover is linearly solvable on graphs from $\mathcal{X}$. We will show that 2-TVC is APX-hard on always $\mathcal{X}$ temporal graphs by using a reduction from Vertex Cover on $\mathcal{Y}$.

▶ **Theorem 14.** 2-TVC *is APX-hard on always* $\mathcal{X}$ *temporal graphs.*

Proof. To prove the theorem we will reduce Vertex Cover on $\mathcal{Y}$ to 2-TVC on always $\mathcal{X}$ temporal graphs. Let $H=(V,E)$ be a graph in $\mathcal{Y}$. First we will show how to construct an always $\mathcal{X}$ temporal graph (G,λ) of lifetime 2. Then we will prove that the size τ of a minimum vertex cover of H is equal to the size σ of a minimum-cardinality sliding 2-window temporal vertex cover of (G,λ).

Let $R\subseteq V$ be the set of vertices of degree 3 in H. We define (G,λ) to be a temporal graph of lifetime 2, where snapshot G_1 is obtained from H by removing the edges with both ends being at distance exactly 2 from R, and snapshot $G_2=H-R$. Figure 1 illustrates the reduction for $H=K_4$.

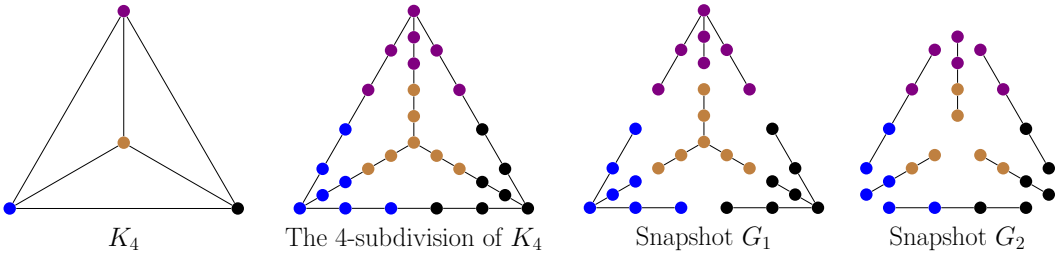


Figure 1 A cubic graph K_4, its 4-subdivision, and the corresponding snapshots G_1 and G_2

Let $\mathcal{S} = (S_1, 1) \cup (S_2, 2)$ be an arbitrary sliding 2-window temporal vertex cover of (G, λ) for some $S_1, S_2 \subseteq V$. Since every edge of H belongs to at least one of the graphs G_1 and G_2, the set $S_1 \cup S_2$ covers all the edges of H. Hence, $\tau \leq |S_1 \cup S_2| \leq |S_1| + |S_2| = |\mathcal{S}|$. As $\mathcal{S}$ was chosen arbitrarily we further conclude that $\tau \leq \sigma$.

To show the converse inequality, let $C \subseteq V$ be a minimum vertex cover of H. Let S_1 be those vertices in C which either have degree 3, or have a neighbor of degree 3. Let also $S_2 = C \setminus S_1$. We claim that $(S_1, 1) \cup (S_2, 2)$ is a sliding 2-window temporal vertex cover of (G, λ). First, let $e \in E$ be an edge in H incident to a vertex of degree 3. Then, by the construction, e is active only in time slot 1, i.e. $e \in E_1 \setminus E_2$, and a vertex v in C covering e belongs to S_1. Hence, e is temporally covered by $(v, 1)$ in (G, λ). Let now $e \in E$ be an edge in H whose both end vertices have degree 2. If one of the end vertices of e is adjacent to a vertex of degree 3 in H, then, by the construction, e is active in both time slots 1 and 2. Therefore, since $C = S_1 \cup S_2$, edge e will be temporally covered in (G, λ) in at least one of the time slots. Finally, if none of the end vertices of e is adjacent to a vertex of degree 3 in H, then e is active only in time slot 2, i.e. $e \in E_2 \setminus E_1$. Moreover, by the construction a vertex v in C covering e belongs to S_2. Hence, e is temporally covered by $(v, 2)$ in (G, λ). This shows that $(S_1, 1) \cup (S_2, 2)$ is a sliding 2-window temporal vertex cover of (G, λ), and therefore $\sigma \leq |S_1| + |S_2| = |C| = \tau$.

Note that the size of a minimum vertex cover of H is equal to the size of a minimum-cardinality sliding 2-window temporal vertex cover of (G, λ) and that any feasible solution to 2-TVC on (G, λ) of size r defines a vertex cover of H of size at most r. Thus, since Vertex Cover is APX-hard on $\mathcal{Y}$ by Lemma 13 and the reduction is approximation-preserving, it follows that 2-TVC is APX-hard as well. ◀

6 Approximation algorithms

In this section we provide several approximation algorithms for SW-TVC with respect to different temporal graph parameters. As the various approximation factors that are achieved are incomparable, the best option for approximating an optimal solution depends on the specific application domain and the specific values of those parameters.

6.1 Approximations in terms of T, Δ, and the largest edge frequency

We begin by presenting a reduction from SW-TVC to Set Cover, which proves useful for deriving approximation algorithms for the original problem. Consider an instance, (G, λ) and $\Delta \leq T$, of the SW-TVC problem. Construct an instance of Set Cover as follows: Let the universe be $U = \{(e, t) : e \in E[W_t], t \in [1, T - \Delta + 1]\}$, i.e. the set of all pairs (e, t) of an edge e and a time slot t such that e appears (and so must be temporally

covered) within window W_t. For every vertex appearance (v, s) we define $C_{v,s}$ to be the set of elements (e, t) in the universe U, such that (v, s) temporally covers e in the window W_t. Formally, $C_{v,s} = \{(e, t) : v \text{ is an endpoint of } e, e \in E_s, \text{ and } s \in W_t\}$. Let $\mathcal{C}$ be the family of all sets $C_{v,s}$, where $v \in V, s \in [1, T]$. The following lemma shows that finding a minimum-cardinality sliding Δ-window temporal vertex cover of (G, λ) is equivalent to finding a minimum-cardinality family of sets $C_{v,s}$ that covers the universe U.

▶ **Lemma 15.** *A family $\mathcal{C} = \{C_{v_1,t_1}, \ldots, C_{v_k,t_k}\}$ is a set cover of U if and only if $\mathcal{S} = \{(v_1, t_1), \ldots, (v_k, t_k)\}$ is a sliding Δ-window temporal vertex cover of (G, λ).*

$O(\ln n + \ln \Delta)$-approximation. In the instance of Set Cover constructed by the above reduction, every set $C_{v,s}$ in $\mathcal{C}$ contains at most $n\Delta$ elements of the universe U. Indeed, the vertex appearance (v, s) temporally covers at most $n-1$ edges, each in at most Δ windows (namely from window $W_{s-\Delta+1}$ up to window W_s). Thus we can apply the polynomial-time greedy algorithm from [9] for Set Cover which achieves an approximation ratio of $H_{n\Delta} - \frac{1}{2} = \sum_{i=1}^{n\Delta} \frac{1}{i} - \frac{1}{2} \approx \ln n + \ln \Delta - \frac{1}{2}$.

$2k$-approximation, where k is the maximum edge frequency. Given a temporal graph (G, λ) and an edge e of G, the Δ-frequency of e is the maximum number of time slots at which e appears within a Δ-window. Let k denote the maximum Δ-frequency over all edges of G. Clearly, for a particular Δ-window W_t, an edge $e \in E[W_t]$ can be temporally covered in W_t by at most $2k$ vertex appearances. So in the above reduction to Set Cover, every element $(e, t) \in U$ belongs to at most $2k$ sets in $\mathcal{C}$. Therefore, the optimal solution of the constructed instance of Set Cover can be approximated within a factor of $2k$ in polynomial time [30], yielding a $2k$-approximation for SW-TVC.

2Δ-approximation. Since the maximum Δ-frequency of an edge is always upper-bounded by Δ, the previous algorithm gives a worst-case polynomial-time 2Δ-approximation for SW-TVC on arbitrary temporal graphs.

6.2 Approximation in terms of maximum degree of snapshots

In this section we give a polynomial-time d-approximation algorithm for the SW-TVC problem on *always degree at most d* temporal graphs, that is, temporal graphs where the maximum degree in each snapshot is at most d. In particular, the algorithm computes an optimum solution (i.e. with approximation ratio $d = 1$) for always matching (i.e. always degree at most 1) temporal graphs. As a building block, we first provide an exact $O(T)$-time algorithm for optimally solving SW-TVC in the class of single-edge temporal graphs, namely temporal graphs whose underlying graph is a single edge.

Single-edge temporal graphs

Consider a temporal graph (G_0, λ) where G_0 is the single-edge graph, i.e. $V(G_0) = \{u, v\}$ and $E(G_0) = \{uv\}$. We reduce SW-TVC on (G_0, λ) to an instance of Interval Covering.

Interval Covering

Input: A family $\mathcal{I}$ of intervals in the line.
Output: A minimum-cardinality subfamily $\mathcal{I}' \subseteq \mathcal{I}$ such that $\bigcup_{I \in \mathcal{I}} = \bigcup_{I \in \mathcal{I}'}$.

An easy linear-time greedy algorithm for the Interval Covering picks at each iteration, among the intervals that cover the leftmost uncovered point, the one with largest finishing time. Algorithm 2 implements this simple rule in the context of the SW-TVC problem.

Algorithm 2 SW-TVC on single-edge temporal graphs.

Input: A temporal graph (G_0, λ) of lifetime T with $V(G_0) = \{u, v\}$, and $\Delta \leq T$.
Output: A minimum-cardinality sliding Δ-window temporal vertex cover $\mathcal{S}$ of (G_0, λ).

$\mathcal{S} \leftarrow \emptyset$
$t = 1$
while $t \leq T - \Delta + 1$ **do**
  if $\exists r \in [t, t + \Delta - 1]$ such that $uv \in E_t$ **then**
    choose maximum such r and add (u, r) to $\mathcal{S}$
    $t \leftarrow r + 1$
  else
    $t \leftarrow t + 1$
return $\mathcal{S}$

▶ **Lemma 16.** *Algorithm 2 solves* SW-TVC *on a single-edge temporal graph and can be implemented to work in time* $O(T)$.

Always degree at most d temporal graphs

We present now the main algorithm of this section, the idea of which is to independently solve SW-TVC for every possible single-edge temporal subgraph of a given temporal graph by Algorithm 2, and take the union of these solutions. We will show that this algorithm is a d-approximation algorithm for SW-TVC on always degree at most d temporal graphs.

Let (G, λ) be a temporal graph, where $G = (V, E)$, $|V| = n$, and $|E| = m$. For every edge $e = uv \in E$, let $(G[\{u, v\}], \lambda)$ denote the temporal graph where the underlying graph is the induced subgraph $G[\{u, v\}]$ of G and the labels of e are exactly the same as in (G, λ).

Algorithm 3 d-approximation of SW-TVC on always degree at most d temporal graphs

Input: An always degree at most d temporal graph (G, λ) of lifetime T, and $\Delta \leq T$.
Output: A sliding Δ-window temporal vertex cover $\mathcal{S}$ of (G, λ).

for $i = 1$ to T **do**
  $\mathcal{S}_i \leftarrow \emptyset$
for every edge $e = uv \in E(G)$ **do**
  Compute an optimal solution $\mathcal{S}'(uv)$ of the problem for $(G[\{u, v\}], \lambda)$ by Algorithm 2
  for $i = 1$ to T **do**
    $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \mathcal{S}'_i(uv)$
return $\mathcal{S}$

▶ **Lemma 17.** *Algorithm 3 is a* $O(mT)$*-time* d*-approximation algorithm for* SW-TVC *on always degree at most* d *temporal graphs.*

Note that, in the case of always matching temporal graphs, the maximum degree in each snapshot is $d = 1$, so the above d-approximation actually yields an exact algorithm.

▶ **Corollary 18.** SW-TVC *can be optimally solved in* $O(mT)$ *time on the class of always matching temporal graphs.*

References

1 Eric Aaron, Danny Krizanc, and Elliot Meyerson. DMVP: foremost waypoint coverage of time-varying graphs. In *Proceedings of the 40th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 29–41, 2014.

2 Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. Ephemeral networks with random availability of links: The case of fast networks. *Journal of Parallel and Distributed Computing*, 87:109–120, 2016.

3 Eleni C. Akrida, Leszek Gasieniec, George B. Mertzios, and Paul G. Spirakis. The complexity of optimal design of temporally connected graphs. *Theory of Computing Systems*, 61(3):907–944, 2017.

4 Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003.

5 Arnaud Casteigts and Paola Flocchini. Deterministic Algorithms in Dynamic Networks: Formal Models and Metrics. Technical report, Defence R&D Canada, April 2013. URL: `https://hal.archives-ouvertes.fr/hal-00865762`.

6 Arnaud Casteigts and Paola Flocchini. Deterministic Algorithms in Dynamic Networks: Problems, Analysis, and Algorithmic Tools. Technical report, Defence R&D Canada, April 2013. URL: `https://hal.archives-ouvertes.fr/hal-00865764`.

7 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems (IJPEDS)*, 27(5):387–408, 2012.

8 Arnaud Casteigts, Bernard Mans, and Luke Mathieson. On the feasibility of maintenance algorithms in dynamic graphs. *CoRR*, abs/1107.2722, 2011. URL: `http://arxiv.org/abs/1107.2722`.

9 Rong chii Duh and Martin Fürer. Approximation of k-set cover by semi-local optimization. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 256–264, 1997.

10 Andrea E. F. Clementi, Claudio Macci, Angelo Monti, Francesco Pasquale, and Riccardo Silvestri. Flooding time of edge-markovian evolving graphs. *SIAM Journal on Discrete Mathematics (SIDMA)*, 24(4):1694–1712, 2010.

11 Stefan Dobrev, Stephane Durocher, Mohsen Eftekhari Hesari, Konstantinos Georgiou, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, Jaroslav Opatrny, Sunil M. Shende, and Jorge Urrutia. Complexity of barrier coverage with relocatable sensors in the plane. *Theoretical Computer Science*, 579:64–73, 2015.

12 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 444–455, 2015.

13 Afonso Ferreira. Building a reference combinatorial model for MANETs. *IEEE Network*, 18(5):24–29, 2004.

14 Paola Flocchini, Bernard Mans, and Nicola Santoro. Exploration of periodically varying graphs. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC)*, pages 534–543, 2009.

15 George Giakkoupis, Thomas Sauerwald, and Alexandre Stauffer. Randomized rumor spreading in dynamic graphs. In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 495–507, 2014.

16 Mohsen Eftekhari Hesari, Evangelos Kranakis, Danny Krizanc, Oscar Morales Ponce, Lata Narayanan, Jaroslav Opatrny, and Sunil M. Shende. Distributed algorithms for barrier coverage using relocatable sensors. *Distributed Computing*, 29(5):361–376, 2016.

17 Anne-Sophie Himmel, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Adapting the bron-kerbosch algorithm for enumerating maximal cliques in temporal graphs. *Social Network Analysis and Mining*, 7(1):35:1–35:16, 2017.

18 P. Holme and J. Saramäki, editors. *Temporal Networks*. Springer, 2013.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

20 David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of the 32nd annual ACM symposium on Theory of computing (STOC)*, pages 504–513, 2000.

21 Orestis Kostakis and Aristides Gionis. On mining temporal patterns in dynamic graphs, and other unrelated problems. In *Proceedings of the 6th International Conference on Complex Networks and Their Applications (COMPLEX NETWORKS)*, pages 516–527, 2017.

22 Orestis Kostakis, Nikolaj Tatti, and Aristides Gionis. Discovering recurring activity in temporal networks. *Data Mining and Knowledge Discovery*, 31(6):1840–1871, 2017.

23 Evangelos Kranakis, Danny Krizanc, Flaminia L. Luccio, and Brett Smith. Maintaining intruder detection capability in a rectangular domain with sensors. In *Proceedings of the 11th International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS)*, pages 27–40, 2015.

24 Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data*, 1(1), 2007.

25 George B. Mertzios, Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP), Part II*, pages 657–668, 2013.

26 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1–23, 2016.

27 Othon Michail and Paul G. Spirakis. Elements of the theory of dynamic networks. *Communications of the ACM*, 61(2):72–72, 2018.

28 Sotiris E. Nikoletseas and Paul G. Spirakis. Probabilistic distributed algorithms for energy efficient routing and tracking in wireless sensor networks. *Algorithms*, 2(1):121–157, 2009. doi:10.3390/a2010121.

29 John Kit Tang, Mirco Musolesi, Cecilia Mascolo, and Vito Latora. Characterising temporal distance and reachability in mobile and online social networks. *ACM Computer Communication Review*, 40(1):118–124, 2010.

30 Vijay V. Vazirani. *Approximation algorithms*. Springer, 2003.

31 Jordan Viard, Matthieu Latapy, and Clémence Magnien. Revealing contact patterns among high-school students using maximal cliques in link streams. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 1517–1522, 2015.

32 Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609:245–252, 2016.

33 Chuan Zhu, Chunlin Zheng, Lei Shu, and Guangjie Han. A survey on coverage and connectivity issues in wireless sensor networks. *Journal of Network and Computer Applications*, 35(2):619–632, 2012.

On the Complexity of Sampling Vertices Uniformly from a Graph

Flavio Chierichetti[1]
Dipartimento di Informatica, Sapienza University, Rome, Italy
flavio@di.uniroma1.it
https://orcid.org/0000-0001-8261-9058

Shahrzad Haddadan[2]
Dipartimento di Informatica, Sapienza University, Rome, Italy
shahrzad.haddadan@uniroma1.it
https://orcid.org/0000-0002-7702-8250

Abstract

We study a number of graph exploration problems in the following natural scenario: an algorithm starts exploring an undirected graph from some *seed* vertex; the algorithm, for an arbitrary vertex v that it is aware of, can ask an oracle to return the set of the neighbors of v. (In the case of social networks, a call to this oracle corresponds to downloading the profile page of user v.) The goal of the algorithm is to either learn something (e.g., average degree) about the graph, or to return some random function of the graph (e.g., a uniform-at-random vertex), while accessing/downloading as few vertices of the graph as possible.

Motivated by practical applications, we study the complexities of a variety of problems in terms of the graph's mixing time $\mathrm{t_{mix}}$ and average degree $\mathrm{d_{avg}}$ – two measures that are believed to be quite small in real-world social networks, and that have often been used in the applied literature to bound the performance of online exploration algorithms.

Our main result is that the algorithm has to access $\Omega\left(\mathrm{t_{mix}}\,\mathrm{d_{avg}}\,\epsilon^{-2}\ln\delta^{-1}\right)$ vertices to obtain, with probability at least $1-\delta$, an ϵ additive approximation of the average of a bounded function on the vertices of a graph – this lower bound matches the performance of an algorithm that was proposed in the literature.

We also give tight bounds for the problem of returning a close-to-uniform-at-random vertex from the graph. Finally, we give lower bounds for the problems of estimating the average degree of the graph, and the number of vertices of the graph.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Social Networks, Sampling, Graph Exploration, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.149

Related Version A full version of the paper is available at `https://arxiv.org/abs/1710.08815`.

Acknowledgements The authors would like to thank Anirban Dasgupta, Ravi Kumar, Silvio Lattanzi and Tamás Sarlós for several useful discussions.

[1] [Supported in part by the ERC Starting Grant DMAP 680153, by a Google Focused Research Award and by the SIR Grant RBSI14Q743.]
[2] [Supported in part by the ERC Starting Grant DMAP 680153, by a Google Focused Research Award and by the SIR Grant RBSI14Q743.]

1 Introduction

Hundreds of millions of people share messages, videos and pictures on Google+ and Facebook each day – these media have an increasingly high political, economical, and social importance in today's world. Data miners have consequently devoted significant amounts of attention to the study of large social networks.

In data mining, one often seeks algorithms that can return (approximate) properties of online social networks, so to study and analyze them, but without having to download the millions, or billions, of vertices that they are made up of. The properties of interest range from the order of the graph [18, 17], to its average degree (or its degree distribution) [10, 13, 16], to the average clustering coefficient [22, 23] or triangle counting [2], to non-topological properties such as the average score that the social network's users assign to a movie or a song, or to the fraction of people that *like* a specific article or page. All these problems have trivial solutions when the graph (with its non-topological attributes) is stored in main memory, or in the disk: choosing a few independent and uniform at random vertices from the graph, and computing their contribution to the (additive) property of interest, is sufficient to estimate the (unknown) value of the graph property – the empirical average of the contributions of the randomly chosen vertices will be close to the right value with high probability, by the central limit theorem.

In applications, though, it is often impossible to have random access to the (vertices of the) graph. Consider, for instance, an online undirected social graph, such as the Facebook friendship graph. An algorithm can *download* a webpage of a given (known) user `alice` from this social graph (e.g., `http://sn.com/user.php?id=alice`), parse the HTML, and get the URLs of the pages of her friends (e.g., `http://sn.com/user.php?id=bob`, `http://sn.com/user.php?id=charles`, etc.) and that user's non-topological attributes (e.g., the set of movies she likes) – an algorithm, though, cannot download the webpage of a vertex without knowing its URL: thus, to download a generic vertex `zoe` from the graph, the algorithm first needs to download all the vertices in (at least) one path from the seed vertex (e.g., `alice`) to `zoe`.

Clearly, given enough many resources, the algorithm could crawl the whole social network (that is, download each of the social network's vertices), and then reduce the problem of computing the online graph property to the centralized one – unfortunately, it is practically infeasible to download millions, or billions, of vertices from a social network (the APIs that can be used to access the network usually enforce strict limits on how many vertices can be downloaded per day). Several techniques have been proposed in the literature for studying properties of online graphs – almost all of them assume to have access to a random oracle that returns a random vertex of the graph according to a certain distribution (usually, either uniform, or proportional to the degree), e.g., [5, 12, 18, 10, 16].

When running algorithms on online social networks, it is often hard or impossible to implement a uniform-at-random random oracle, and to get samples out of it – the complexity of this oracle is one of the main problems that we tackle in this paper.

In practice, an algorithm is given (the URL of) a seed vertex (or, some seed vertices) of the social network; the algorithm has to download that seed vertex, get the URLs of its neighbors, and then decide which of them to download next; after having downloaded the second node, the algorithm (might) learn of the existence of some other URLs/vertices, and can then decide which of the known (but unexplored) URLs to download – and so on, and so forth, until the algorithm can return a good approximation of the property to be estimated.

The natural cost function in this setting is the (random) number of vertices that the algorithm has to download, or *query*, before making its guess – the cost function is usually bounded in terms of properties of the graph (e.g., its order, its average degree, etc.), and in terms of the quality of the algorithm's guess.

Many problems of this form can be found in the literature. In this paper, we consider two natural problems, that are at the heart of many others, and whose complexity (as far as we know) was open before this work:

- the "average score problem": assuming that each vertex holds some score in $[0, 1]$, compute an approximation of the average of the scores;
- the "uniform at random sample": return a random vertex from the graph whose distribution is approximately uniform.

In a sense, the latter problem is technically more interesting than the former (a solution to the latter provides a solution to the former). In practice, though, the average score problem is much more significant (and ubiquitous), given its many applications [11, 1, 14, 21] (e.g., computing the favorability rating of a candidate, or the average star-rating of a movie). These problems are similar in flavor to some graph property testing problems [15]. Observe that our setting is similar, but different, than various graph property testing settings – many of the existing graph property testing algorithms require, as a primitive, the ability to sample a uniform at random vertex. Our work can be seen as a way of implementing that primitive using the oracles mentioned above. (See also, [4]).

A number of algorithms have been proposed for the uniform-at-random sample problem [7, 18] – the best known algorithms require roughly $\tilde{O}\left(\mathrm{t_{mix}} \cdot \mathrm{d_{avg}}\right)$ vertex queries/downloads to return a vertex whose distribution is (close to) uniform at random , where $\mathrm{t_{mix}}$ is the mixing time of the lazy random walk on the graph, and $\mathrm{d_{avg}}$ is its average degree [7][3]. These algorithms do not use any knowledge of the average degree of the graph $\mathrm{d_{avg}}$, but need to know a constant approximation of its mixing time $\mathrm{t_{mix}}$. To our knowledge, the best lower bound for the uniform-at-random sample problem before this work, was $\Omega(\mathrm{t_{mix}} + \mathrm{d_{avg}})$ [7] – one of the main results of this paper is (i) an almost tight lower bound of $\Omega(\mathrm{t_{mix}} \cdot \mathrm{d_{avg}})$ for this problem, for wide (in fact, polynomial) ranges of the two parameters.[4] The lower bound holds even for algorithms that know constant approximations of $\mathrm{d_{avg}}$.

Our lower bound construction for the uniform-at-random sample problem also provides (ii) a tight lower bound of $\Omega\left(\mathrm{d_{avg}}\,\mathrm{t_{mix}}\right)$ for the average score problem – in fact, we resolve the complexity of the average score problem by showing that our lower bound coincides with the complexity of some previously proposed algorithms, whose analysis we improve.

The same lower bound construction further resolves (iii) the complexity of the average-degree estimation problem, and (iv) entails a non-tight, but significant, lower bound for the problem of guessing the graph order (that is, the number of vertices in the graph).

It is interesting to note that all the algorithms that were proposed require $O(\log n)$ space, while our lower bounds hold for general algorithms with no space restriction. Thus, the problems we consider can be solved optimally using only tiny amounts of space.

---

[3] Real-world social networks are known to have a small average degree $\mathrm{d_{avg}}$; their mixing time $\mathrm{t_{mix}}$ has been observed [20] to be quite small, as well.

[4] Observe that the two parameters have to obey some bound for such a lower bound to hold – in general, any problem can be solved by downloading all the n vertices of the graph: thus, if $\mathrm{t_{mix}} \cdot \mathrm{d_{avg}} > \omega(n)$, one can solve the uniform-at-random sample problem with less than $o(\mathrm{t_{mix}} \cdot \mathrm{d_{avg}})$ vertex queries.

2 Preliminaries

Consider a connected and undirected graph $\mathcal{G} = \langle V_{\mathcal{G}}, E_{\mathcal{G}} \rangle$ with no self-loops (e.g., the Facebook friendship graph), and a function on its vertices $\mathcal{F} : V_{\mathcal{G}} \to [0, 1]$.[5] We aim to estimate the average value of this function, i.e., $f_{avg} = \sum_{v \in V_{\mathcal{G}}} \mathcal{F}(v)/n$ where $n = |V_{\mathcal{G}}|$.

Motivated by applications, we assume that accessing the graph is a costly operation, and that there is little or no information about its global parameters such as the average degree, the number of vertices or the maximum degree. However, we can access a "friendship" oracle: that is, an oracle which, given a vertex $v \in V_{\mathcal{G}}$, outputs references (their ids, or their URLs) to its neighbors $N_v = \{u \in V_{\mathcal{G}} | (v, u) \in E_{\mathcal{G}}\}$. In such a setting, it is natural to approximate f_{avg} by taking samples from a Markov chain based on the graph structure (see, e.g., [9, 8]). A simple random walk on the graph, though, will not serve our purposes since it samples vertices with probability proportional to their degree, while our goal is to take a uniform average of the values of $\mathcal{F}$.

On a graph $\mathcal{G} = \langle V_{\mathcal{G}}, E_{\mathcal{G}} \rangle$, a lazy simple random walk is a Markov chain which being at vertex $v \in V_{\mathcal{G}}$, stays on v with probability $1/2$ and moves to $u \in N_v$ with probability $1/(2 \deg(v))$. Given that $\mathcal{G}$ is connected, the lazy random walk will converge to its unique stationary distribution which we denote by Π^1 and which is equal to $\Pi^1(v) = \deg(v)/2|E_{\mathcal{G}}|$, $\forall v \in V_{\mathcal{G}}$.

By $\mathrm{t}_{\mathrm{mix}}(\mathcal{G})$ we refer to the mixing time of the lazy random walk on $\mathcal{G}$, which is the minimum integer satisfying: for any $\tau \geq \mathrm{t}_{\mathrm{mix}}(\mathcal{G})$, $\left|X^\tau - \Pi^1\right|_1 \leq 1/4$, where X^τ is the distribution of the lazy walk at time τ, and $|\cdot|_\ell$ is the ℓ-norm of a vector. Note that by the theory of Markov chains, by taking $\tau \geq \mathrm{t}_{\mathrm{mix}}(\mathcal{G}) \log(1/\epsilon)$ we have $\left|X^\tau - \Pi^1\right|_1 \leq \epsilon$. We denote the uniform distribution on vertices of $\mathcal{G}$ by Π^0, i.e., $\Pi^0(v) = 1/|V_{\mathcal{G}}|, \forall v \in \mathcal{G}$. In general, we denote a distribution on $V_{\mathcal{G}}$ weighing each vertex $v \in V_{\mathcal{G}}$ proportional to $\deg(v)^\zeta$ by Π^ζ. We may drop all the subscripts when doing so does not cause ambiguity.

Following the framework of [7], we consider two measures of time complexity. First the number of downloaded vertices, and second the number of steps the algorithm takes to produce the output. Note that accessing an already downloaded vertex has a negligible cost, and hence, the most relevant cost of the algorithm is the number of downloaded vertices. As mentioned in the introduction, the algorithms considered in [7] and in this paper, only require space to store constantly many vertices, while our lower bound results hold regardless of the space complexity of the algorithms.

Our Contribution. We begin by discussing the problem of producing an approximately-uniform sample vertex from an unknown graph (Problem 1); showing that some algorithm presented in the literature are optimal (Theorem 1). Then, we proceed to the problem of estimating f_{avg} for a function $\mathcal{F} : V_{\mathcal{G}} \to [0, 1]$ (Problem 2). We extend the positive results of [7]; we particularly study one algorithm, the "Maximum Degree algorithm", which we show to be suboptimal in the number of downloaded vertices. This algorithm requires knowledge of some constant approximation of the graph's mixing time, and and upper bound on its maximum degree. We also show new lower bounds for constant approximations of the order and the average degree of a graph. A summary of our contribution is presented in Table 1. We remark that, in practice, only an upper bound on the mixing time is available. In the equations of Table 1, $\mathrm{t}_{\mathrm{mix}}$ can be substituted with any upper bound on the mixing time of the graph.

[5] Note that from any bounded function we can get a function with range $[0, 1]$, through a simple affine transform. Therefore, our results can be trivially extended to functions with any bounded range.

Table 1 Upper bounds and lower bounds on number of queried vertices for algorithms which explore the graph using a neighborhood oracle and a seed vertex. As mentioned before, $\mathrm{t_{mix}}$ is the mixing time of the lazy random walk on the graph, $\mathrm{d_{avg}}$ is its average degree, $\mathcal{D}$ is an upper bound on its maximum degree, Π^1 is its stationary distribution, and ϵ and δ are the precision parameters. The lower bounds for estimating the number of vertices and the average degree hold for any constant approximation.

	Upper Bound	Lower Bound
Average of a Bounded Function	$O(\mathrm{t_{mix}}\,\mathrm{d_{avg}}\log(\delta^{-1})\epsilon^{-2})$ (Theorem 2, with an Algorithm of [7])	$\Omega(\mathrm{t_{mix}}\,\mathrm{d_{avg}}\log(\delta^{-1})\epsilon^{-2})$ (Theorem 3)
Uniform Sample	$O(\mathrm{t_{mix}}\,\mathrm{d_{avg}}\log(\epsilon^{-1}))$ ([7])	$\Omega(\mathrm{t_{mix}}\,\mathrm{d_{avg}})$ (Theorem 1)
Number of Vertices	$O(\mathrm{t_{mix}}\max\{\mathrm{d_{avg}}, \lvert\Pi^1\rvert_2^{-1/2}\}\log(\delta^{-1})\log(\epsilon^{-1})\epsilon^{-2})$ ([18])	$\Omega(\mathrm{t_{mix}}\,\mathrm{d_{avg}})$ (Theorem 4)
Average Degree	$O(\mathcal{D}^2\mathrm{t_{mix}}\,\mathrm{d_{avg}}\log(\delta^{-1})\epsilon^{-2})$ (Application of Theorem 2)	$\Omega(\mathrm{t_{mix}}\,\mathrm{d_{avg}})$ (Theorem 4)

In Section 3, we prove our lower bound results on the number of oracle calls for the following problems: sampling a vertex, learning the order, and the average degree of the graph. Estimations of these parameters in a graph are intertwined meaning that a knowledge about one of them the complexity of estimating the other one changes. For instance, Goldreich and Ron [16] show that, if a uniform sample generator is accessible at zero cost (alternatively, if the order of graph is precisely known), then the average degree is computable in $\sqrt{|V_{\mathcal{G}}|/\,\mathrm{d_{avg}}}$ steps. Our lower bounds for the aforementioned problems hold if the algorithm has no $\epsilon-$approximation of the order, and of the average degree of the graph. On the other hand, the lower bound we obtain for an $\epsilon-$approximation of a bounded function's average holds even if the graph's structure is precisely known.

Number of downloads to produce a close-to-uniform sample. We prove a lower bound of $\Omega(\mathrm{t_{mix}}\,\mathrm{d_{avg}})$, thus, showing that the rejection algorithm and the maximum degree algorithm suggested in the literature [7] can be optimal (Theorem 1). We observe that the two algorithms require some knowledge of $\mathrm{t_{mix}}$. In most scenarios, only an upper bound on $\mathrm{t_{mix}}$ is available. Thus, it is more accurate to claim that the two algorithms are optimal when some constant approximation of the mixing time is available.

Number of downloads to estimate the number of vertices. The problem of estimating the order of a graph is widely studied [9, 18]. Katzir et al. [18] (2011) propose an algorithm that, having access to an oracle that produces random vertices from the graph's stationary distribution, requires $\max\{\frac{1}{|\Pi^1|_2}, \mathrm{d_{avg}}\}(\frac{1}{\epsilon^2\delta})$ samples to obtain an ϵ approximation with probability at least $1-\delta$. It has been shown the number of samples in Katzir's algorithm is necessary ([17]). The Katzir et al. algorithm implies an upper bound of $\mathrm{t_{mix}}\max\{\frac{1}{|\Pi^1|_2}, \mathrm{d_{avg}}\}(\frac{\log(\epsilon^{-1})}{\epsilon^2\delta})$ vertex queries to obtain an ϵ approximation with probability at least $1-\delta$ in our friendship-oracle model. In Theorem 4 we present a lower bound on the number of accesses to the vertices, to get a constant approximation of the graph's order in our friendship-oracle model. Our lower bound is tight for the graphs that satisfy $\frac{1}{|\Pi^1|_2} < \mathrm{d_{avg}}$ – that is, the graphs whose variance

of the degree distribution is greater than n.[6] This class include, say, all the graphs having a power-law degree distribution with exponent smaller than 3/2 (e.g., social networks [19]).

We remark that this lower bound problem is still open for graphs with $\frac{1}{|\Pi^1|_2} > \mathrm{d_{avg}}$, for example regular sparse graphs. Recently, Ben-Hamou et al. studied the problem of estimating the size of a regular graph when having access to an oracle analogous to ours, and presented an constant approximation algorithm querying $(t_{\mathrm{unif}})^{3/4}\sqrt{n}$ vertices, where t_{unif} is the minimum integer satisfying: $\tau \geq t_{\mathrm{unif}}(\mathcal{G})$, $\max_{v\in V} |X^\tau(v)/\Pi^1(v) - 1| \leq 1/4$ [3].

Number of downloads to estimate the average degree. There are quite a few results on estimating the average degree of a graph. The first one by Feige et al. [13] introduced a sublinear algorithm of complexity $\sqrt{|V_\mathcal{G}|}$ for a 2-approximation. Goldreich et al. [16] extends Feige et al.'s result and presents an $(1 \pm \epsilon)$ approximation algorithm with running time $O(1/\epsilon)\sqrt{|V_G/\,\mathrm{d_{avg}}|}$ – they also prove a lower bound on the number of samples of $\sqrt{|V_G|/\,\mathrm{d_{avg}}}$ – both of [13] and [16] assume to have access to an oracle capable of producing a uniform at random vertex. Recently, Dasgupta et al. [10] showed that by sampling $O(\log(\mathcal{D})\log\log(\mathcal{D}))$ vertices of a graph from some weighted distribution[7] one can obtain a $(1\pm\epsilon)$ approximation of its average degree, where $\mathcal{D}$ is an upper bound on the maximum degree. By factoring in the the cost of sampling, the complexity becomes $O(\mathrm{t_{mix}}\log(\mathcal{D})\log\log(\mathcal{D}))$. Taking $\mathcal{D} = n$ and adding the cost of estimating the graph size, takes the upper bound to: $O\left(\mathrm{t_{mix}}\left((\log(n)\log\log(n)) + \mathrm{d_{avg}} + \frac{1}{|\Pi^1|_2}\right)\right)$.

In Theorem 4 of this paper we show that by downloading $o(\mathrm{t_{mix}}\,\mathrm{d_{avg}})$ vertices, it is impossible for an algorithm to have any constant approximation of the average degree $\mathrm{d_{avg}}$ with probability more than some constant.

Finally, our main result is the following lower bound – unlike the above three lower bounds, this one holds even if we know exactly the graph's structure.

Number of downloads to find an ϵ, δ approximation for the average of a bounded function. In Theorem 3, we show that an algorithm requires $\Omega\left(\mathrm{t_{mix}}\,\mathrm{d_{avg}}(1/\epsilon^2)\log(1/\delta)\right)$ vertex downloads to produce an $\epsilon-$additive approximation of f_{avg}, with probability at least $1-\delta$. This lower bound, together with Theorem 2, allows us to conclude that the "maximum degree algorithm" is an optimal algorithm for this problem. Note that this algorithm has to have some upper bound $\mathcal{D}$ on the maximum degree of the graph. In many situations, one can assume that this information is available – for instance $\mathcal{D} \leq n$ and, in many cases, one can assume to have a constant approximation to the order of the graph (for instance, in Facebook, one could claim that $\mathcal{D}$ is no larger than the world's population.) Observe that the maximum degree algorithm suffers no loss in getting a large $\mathcal{D}$, as opposed to a tighter one, since $\mathcal{D}$ does not impact the upper bound on the number of downloaded vertices.

2.1 Statement of Problems and Results

▶ **Problem 1.** *Input: A seed vertex $s \in V$ in graph $\mathcal{G} = \langle V_\mathcal{G}, E_\mathcal{G}\rangle$. Output: A random vertex $v \in V_\mathcal{G}$ whose distribution is at total variation distance at most ϵ from the uniform one on $V_\mathcal{G}$.*

---

[6] Let pr_k be the fraction of vertices with degree k. We have $\left|\Pi^1\right|_2 = n\sum_{i=1}^n \mathrm{pr}_k \frac{k^2}{4|E|^2} = \frac{1}{n\,\mathrm{d_{avg}}^2}\sum_{i=1}^n \mathrm{pr}_k k^2$. Thus, to have $1/\sqrt{|\Pi^1|_2} \leq \mathrm{d_{avg}}$, it is necessary and sufficient to have $\sum_{i=1}^n k^2\mathrm{pr}_k > n$.

[7] Dasgupta et al. use an oracle samples each $v \in V_\mathcal{G}$ proportional to $\deg(v) + c$ for some constant c. Note that for $c = 0$ this distribution will be the same as the stationarity.

Several algorithms have been proposed for Problem 1 [7, 18] – we will specifically consider the "maximum degree sampling" algorithm, the "rejection sampling" algorithm, and the "Metropolis Hasting" algorithm.

The efficiency of the three algorithms has been studied in terms of the number of their running time (or the number of steps they make on the Markov chain they are based on) and, more importantly, on the number of *queries*[8] (or downloaded vertices) that the algorithm performs. The rejection sampling and maximum degree algorithms produce a close-to-uniform random vertex by querying[9] $\tilde{O}\left(\mathrm{t_{mix}}\,\mathrm{d_{avg}}\right)$ distinct vertices from the graph, where $\mathrm{t_{mix}}$ is the mixing time of a simple random walk on $\mathcal{G}$, and $\mathrm{d_{avg}}$ is the average degree of $\mathcal{G}$. In terms of space complexity, each of these algorithms is based on a simple random walk on $\mathcal{G}$ and thus only require space to save constant number of vertices.

One of the main results of this paper is Theorem 1, which shows the optimality of the maximum degree, and of the rejection sampling, algorithms for Problem 1 – their running time. We observe that our lower bound holds regardless of the amount of space available to the algorithm.

▶ **Theorem 1.** *For any large enough constant c, and for any arbitrary n, $d = \omega(\log n)$, and $t = o(\frac{n}{d^2})$ there exists a distribution over graphs $\mathcal{G} = \langle V, E\rangle$, each having mixing time $\Theta(t)$, $\mathbb{E}(|V|) = \Theta(n)$, $\mathrm{d_{avg}} = \mathbb{E}_{v\in V}(\deg_v) = \Theta(d)$, such that any algorithm $\mathcal{A}$ that queries less than $\mathrm{d_{avg}}\,\mathrm{t_{mix}}/c$ vertices of $\mathcal{G}$, and that returns some random vertex of $\mathcal{G}$ with distribution $\Pi_{\mathcal{A}}$, is such that*[10] $\mathbb{E}\left[\left|\Pi_{\mathcal{A}} - \Pi^0\right|_1\right] \geq \frac{24}{100} - \frac{202}{c-1}$.

The same lower bounds also hold if one aims to obtain the generic Π^ζ distribution [11] *: if $\zeta > 1$, and d and t satisfy $d = o(t^{\frac{\zeta-1}{\zeta}})$, then any algorithm $\mathcal{A}$ that queries less than $\mathrm{d_{avg}}\,\mathrm{t_{mix}}/c$ vertices of $\mathcal{G}$, and that returns some random vertex of $\mathcal{G}$ with distribution $\Pi_{\mathcal{A}}$, is such that: $\mathbb{E}\left[\left|\Pi_{\mathcal{A}} - \Pi^\zeta\right|_1\right] \geq \frac{24}{100} - \frac{202}{c-1}$.*

The above theorem, and the other lower bound results that we mention in this section, will be proved in Section 3.

Then, we consider the problem of finding the average of a function $\mathcal{F}$ defined on vertices of a graph and ranging in $[0, 1]$.

▶ **Problem 2.** *Input: A seed vertex $s \in V$ in graph $\mathcal{G} = \langle V, E\rangle$ – each vertex v holds a value $0 \leq \mathcal{F}(v) \leq 1$ which we learn upon visiting it. Output: $\bar{f}$ such that $\mathbb{P}\left(|\bar{f} - f_{avg}| \leq \epsilon\right) \geq 1 - \delta$.*

Note that having a uniform sampler (the maximum degree or rejection sampling algorithm of [7]), we can have an ϵ approximation of f_{avg} with probability $1-\delta$ by taking $O(\epsilon^{-2}\log(\delta^{-1}))$ independent samples which are ϵ close to uniformity. In total, the number of queries will be $O(\mathrm{t_{mix}}\,\mathrm{d_{avg}}\log(\delta^{-1})\epsilon^{-2}\log(\epsilon^{-1}))$. Here we propose a slight variation of the "maximum degree" algorithm to obtain a tight upper bound. We improve the analysis of the "maximum degree algorithm" in Theorem 2 – its performance beats the other natural three algorithms, and the main result of this paper is that this performance is optimal (Theorem 3).

The proof of the following Theorem is omitted from this extended abstract.

---

[8] A vertex is "queried", when the set of its neighbors is obtained from the oracle for the first time – equivalently, when it is downloaded.

[9] To get ϵ close to the uniform distribution we need $O\left(\mathrm{t_{mix}}\,\mathrm{d_{avg}}\log(\epsilon^{-1})\right)$ downloads.

[10] Observe that the expected ℓ_1 distance between the distributions is over the random variable $\Pi_{\mathcal{A}}$.

[11] The Π^ζ distribution is the distribution in which the probability of a vertex $v \in V$ is proportional to $deg(v)^\zeta$.

Algorithm 1 The Maximum Degree Algorithm.

Input: Seed vertex $s \in V_{\mathcal{G}}$, $\mathrm{t}'_{\mathrm{mix}}$ a constant approximation of $\mathrm{t}_{\mathrm{mix}}$, and an upper bound $\mathcal{D}$ on d_{max}

Output: An ϵ additive approximation of f_{avg} with probability at least $1-\delta$

Consider the maximum degree Markov chain: at vertex $v \in V$ go to the generic $u \in N_v$ with probability $1/\mathcal{D}$, otherwise stay at v.
$T \leftarrow \mathrm{t}'_{\mathrm{mix}}\mathcal{D}/\mathrm{d}_{\min}$
Starting from s, run the chain for $T \cdot \left(1+\epsilon^{-2}\ln\delta^{-1}\right)$ steps – let $v_0 = s, v_1, v_2, \ldots$ be the states that are visited by the walk
$S \leftarrow 0, t \leftarrow 0, i \leftarrow 0$
while $t < \frac{\mathrm{d}_{\mathrm{avg}}}{d_{min}}(\mathrm{t}'_{\mathrm{mix}}/\epsilon^2)\log(1/\delta)$ **do**
  $i \leftarrow i+1$
  if $v_i \neq v_{i-1}$ **then**
    $t \leftarrow t+1$
  $S = S + \mathcal{F}(v_i)$
return S/i

▶ **Theorem 2.** *Consider a graph $\mathcal{G} = \langle V, E\rangle$, and a function $\mathcal{F} : V \to [0,1]$. Let $\mathrm{t}_{\mathrm{mix}}$ be the mixing time of the simple lazy random walk on $\mathcal{G}$. Let $\bar{f}$ be the value returned by Algorithm 1. We assume the algorithm knows a constant approximation to $\mathrm{t}_{\mathrm{mix}}$. i.e. $\mathrm{t}'_{\mathrm{mix}} = \Theta(\mathrm{t}_{\mathrm{mix}})$. Then,*

$$\mathbb{P}\left(|\bar{f} - f_{avg}| \geq \epsilon\right) \leq \delta \tag{1}$$

This algorithm queries $\Theta(\mathrm{t}_{\mathrm{mix}}\,\mathrm{d}_{\mathrm{avg}}\,\epsilon^{-2}\log(\delta^{-1}))$ vertices from the graph, and requires space for saving a constant number of them. The number of computational steps it performs is $\Theta(\mathcal{D}\mathrm{t}_{\mathrm{mix}}\epsilon^{-2}\log(\delta^{-1}))$.

The main result of this paper is the following lower bound which complements the upper bound given in the previous theorem:

▶ **Theorem 3.** *For any arbitrary n, $d = \omega(\log n)$, and $t = o(\frac{n}{d^2})$ there exists a distribution over graphs $\mathcal{G} = \langle V, E\rangle$ with mixing time $\Theta(t)$, $\mathbb{E}(|V|) = 4n$, $\mathrm{d}_{\mathrm{avg}} = \mathbb{E}_{v\in V}(\deg_v) = \Theta(d)$, and a function $\mathcal{F} : V \to \{0,1\}$ such that, any algorithm $\mathcal{A}$ as described above which aims to return the average of $\mathcal{F}$, with ϵ precision for arbitrary $0 < \epsilon, \delta < 1$, and queries less than $\Omega(\mathrm{t}_{\mathrm{mix}}\,\mathrm{d}_{\mathrm{avg}}\,\epsilon^{-2}\log(\delta^{-1}))$ vertices of $\mathcal{G}$ fails with probability greater than δ.*

Finally, we consider the problems of obtaining an approximation of the average degree, and the number of vertices, of a graph:

▶ **Problem 3.** *Input: A seed vertex $s \in V$ in graph $\mathcal{G} = \langle V, E\rangle$. Output: an integer $\bar{n}$ such that $\mathbb{P}\left(|\bar{n} - |V|| \leq \epsilon\right) \geq 1-\delta$.*

By a result of Katzir [18], we know by taking $\max\{\mathrm{d}_{\mathrm{avg}}, 1/\sqrt{|\Pi^2|_2}\}\epsilon^{-2}\delta^{-1}$ samples from the stationary distribution we are capable to obtain an ϵ approximation with probability at least $1-\delta$. To implement a sampling oracle using our neighborhood oracle, we can run a Markov chain for $\mathrm{t}_{\mathrm{mix}}\log(\epsilon^{-1})$ steps. Thus, the runtime will be $\mathrm{t}_{\mathrm{mix}}\max\{\mathrm{d}_{\mathrm{avg}}, 1/\sqrt{|\Pi^1|_2}\}\log(\epsilon^{-1})\epsilon^{-2}\delta^{-1}$, which for constant ϵ and δ is $\mathrm{t}_{\mathrm{mix}}\max\{\mathrm{d}_{\mathrm{avg}}, 1/\sqrt{|\Pi^1|_2}\}$. Theorem 4 provides a lower bound for a constant approximation which is as mentioned before tight when the variance of the degree distribution is greater than n.

▶ **Problem 4.** *Input: A seed vertex $s \in V$ in graph $\mathcal{G} = \langle V, E \rangle$. Output: an integer $\bar{d}$ such that $\mathbb{P}\left(|\bar{d} - \mathrm{d_{avg}}| \leq \epsilon\right) \geq 1 - \delta$.*

Normalize the function $deg : V_{\mathcal{G}} \to \mathbb{R}$ by dividing its value to $\mathcal{D}$. By Theorem 2, Algorithm 1 provides an ϵ approximation with probability at least $1 - \delta$ after downloading $O(\mathcal{D}^2\, \mathrm{t_{mix}}\, \mathrm{d_{avg}}\, \epsilon^{-2} \log(\delta^{-1}))$ many vertices – that is, $O(\mathcal{D}^2\, \mathrm{t_{mix}}\, \mathrm{d_{avg}})$ many vertices for constant ϵ and δ. With Theorem 4, we provide a lower bound for a constant approximation.

▶ **Theorem 4.** *For any arbitrary n, $d = \omega(\log n)$, and $t = o(\frac{n}{d^2})$ there exists a distribution of graphs $\mathcal{G} = \langle V, E \rangle$ with mixing time $\Theta(t)$, $\mathbb{E}(|V|) = \Theta(n)$, $\mathrm{d_{avg}} = \mathbb{E}_{v \in V}(\deg_v) = \Theta(d)$ such that, for arbitrary constants $c' > 1$ and large enough c, any algorithm that queries at most $\mathrm{d_{avg}}\, \mathrm{t_{mix}}\, /c$ vertices of the graph, and that outputs an estimation $\bar{n}$ of $n = |V|$ (resp., an estimation $\bar{d}$ of $\mathrm{d_{avg}} = 2|E|/n$), has to satisfy $\max\{\bar{n}/n, n/\bar{n}\} > c'$ (resp., $\max\{\bar{d}/\,\mathrm{d_{avg}}, \mathrm{d_{avg}}\,/\bar{d}\} > c'$), with probability at least $\frac{99}{100} - \frac{202}{c-1}$.*

3 Proof Strategy of the Main Theorems

The proof of our Lower Bounds will be based on the following high-level strategy. Nature will first randomly sample a graph $\mathcal{H}$ according to some distribution; with probability 1/2, $\mathcal{H}$ will be the unknown graph traversed by the algorithm; with the remaining probability, the algorithm will traverse a graph $\mathcal{G}$ which is obtained from $\mathcal{H}$ by means of a transformation that we call the *decoration construction*. We will prove that, for the right choice of the distribution over $\mathcal{H}$, an algorithm that performs too few queries to the unknown graph will be unable to tell with probability more than $1/2 + o(1)$, whether the unknown graph it is traversing is distributed like $\mathcal{H}$, or like $\mathcal{G}$.

The decoration construction will guarantee that the properties (e.g., number of nodes, average degree, or even the values assigned by the bounded function to the vertices) will be quite far from each other in $\mathcal{H}$ and $\mathcal{G}$. This will make it impossible for the algorithm to get good approximation of any of those properties – we will also show it impossible for the algorithm to return a close to uniform-at-random vertex (essentially because the decoration construction will add a linear number of nodes to $\mathcal{H}$, and the algorithm will be unable to visit any of them with the given budget of queries.)

We present a roadmap of our proof strategy here, and omit the detailed proofs in this extended abstract. We start by describing the *decoration construction* which, given any graph $\mathcal{H}$, produces a graph $\mathcal{G}$ with similar mixing time and average degree, but with a linear number of "hidden" new vertices. After presenting the definition for the decoration construction, in Definition 6 we introduce a class of graphs to which we apply this construction. These graphs' mixing time and average degree can be set arbitrarily. Later, in Lemma 7 we prove that if an algorithm, equipped only with the neighborhood oracle traverses a graph of this type and queries few vertices of it, it will not be capable of finding any of its hidden vertices. This is our main lemma from which Theorems 1, 3, and 4 can be concluded. We now proceed to the formal definitions:

▶ **Definition 5** (The Decoration Construction)**.** Let $\mathcal{H} = \langle V, E \rangle$ be an arbitrary graph. We construct $\mathcal{G}$ from $\mathcal{H}$ in the following way:

Take $t := \mathrm{t_{mix}}(\mathcal{H})$, and mark any vertex $v \in V$ with probability $1/t$. For any marked vertex $v \in V$, add a vertex v^* and connect it to v via an edge. For a constant c_1, attach $c_1 t - 1$ new degree one vertices to v^* – this makes the degree of v^* equal to $c_1 t$. Let this new graph be $\mathcal{G}$. We denote the set of marked vertices by MARKED and the set $\mathcal{G} \setminus \mathcal{H}$ by STARRED.

By saying a vertex v is STARRED (MARKED) we mean $v \in$ STARRED ($v \in$ MARKED), and to indicate their numbers we use a preceding #. We call the STARRED vertices with degree $c_1 t$ the STARRED centers. Note that to any STARRED vertex we can associate a unique MARKED vertex.

We now introduce the random graph to which we will apply the decoration construction, and which will be at the heart of our lower bounds.

▶ **Definition 6.** We define the graph $\mathcal{H}_{n,d,\psi}$ as follows: given arbitrary parameters n, d, and $0 < \psi < 1$, take two Erdös-Rényi graphs $H_1 = \langle V_{H_1}, E\rangle$ and $H_2 = \langle V_{H_2}, E\rangle$ with parameters $\langle n, d/n\rangle$. Choose ψn vertices uniformly at random from V_{H_1} namely $v_1, v_2, \dots v_{\psi n}$, and then ψn vertices uniformly at random from V_{H_2} namely $u_1, u_2, \dots n_{\psi n}$. Select a uniformly random permutation σ of ψn numbers and put an edge between the vertices v_i and $u_{\sigma(i)}$ for each $1 \le i \le \psi n$.

The decoration construction does not change the mixing time of $\mathcal{H}$ drastically, in fact, $\mathrm{t_{mix}}(\mathcal{H}) \le \mathrm{t_{mix}}(\mathcal{G}) \le c\,\mathrm{t_{mix}}(\mathcal{H})$ for some constant c. The $\mathcal{H}_{n,d,\psi}$s are useful in our proofs, for the reason that, by changing the parameters n, d, and ψ in certain ranges these graphs acquire arbitrary $\mathrm{t_{mix}}$ and $\mathrm{d_{avg}}$, yet their behaviour remains similar to Erdös-Rényi random graphs.

We now present the following lemma (proof ommited), which states if few queries are performed by an algorithm, then the probability of finding the STARRED vertices is tiny. Then, we conclude our main theorems; Theorem 1, 3, and 4:

▶ **Lemma 7.** *Consider arbitrary n, $d > \omega(\log n)$, $\Omega(\log n) < t < o(n/d^2)$, so that $t/d = \Omega(1)$, take $\mathcal{G}$ to be the graph obtained from the decoration construction applied to $\mathcal{H}_{n,d,d/t}$[12]. We have, $\mathrm{t_{mix}}(\mathcal{G}) = \Theta(t)$ and $\mathrm{d_{avg}} = \Theta(d)$.*

If, instead, $t = O(\log n/\log d)$, and $d = \Theta(\log^d n)$, take $\mathcal{G}$ to be the decorated version of an Erdös-Rényi graph with parameters $\langle n, d/n\rangle$.

Then,

- *if an algorithm traverses the edges of $\mathcal{G}$ and queries at most td/c vertices of $\mathcal{G}$; c being a constant, then with probability at least $99/100 - 202/(c-1)$ there is no* STARRED *vertex among its queried vertices.*
- *If an algorithm traverses the edges of $\mathcal{G}$ and queries $q \le \frac{n}{cd}$ vertices of $\mathcal{G}$; c being a constant, then with probability $1 - o(1)$ the expected number of* STARRED *centers which have been queried is less than $\frac{8c}{c-1}\left(\frac{q}{dt}\right)$.*

We can finally prove our three main Theorems:

*** Proof of Theorem 1.** Consider the two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, $\mathcal{G}_1$ being the graph of Lemma 7 with $c_1 = 1$ and $\mathcal{G}_2$ the same graph without the STARRED vertices (the graph before the decoration construction). Any algorithm which queries less than $\mathrm{t_{mix}}\,\mathrm{d_{avg}}/c$ vertices of $\mathcal{G}_1$ or $\mathcal{G}_2$ will fail to distinguish between them with probability at least $\frac{1}{100} + \frac{202}{c-1}$. Let $\Pi^0_{\mathcal{G}_i}$ be the uniform distribution on vertices of $\mathcal{G}_i$; $i = 1, 2$. In $\mathcal{G}_1$ with high probability we have at least $2|V_{\mathcal{G}_1}|$ STARRED vertices.

- *Part 1.* Note that $|\Pi^0_{\mathcal{G}_1}, \Pi^0_{\mathcal{G}_2}|_1 \ge 1/4$. Thus, if the natures selects $\mathcal{G}_1$, any algorithm $\mathcal{A}$ which aims to outputs $\Pi^0_{\mathcal{G}_1}$ will return a distribution $\Pi_{\mathcal{A}}$ satisfying $|\Pi^0_{\mathcal{G}_1}, \Pi_{\mathcal{A}}|_1 \ge 1/4 - \frac{1}{100} - \frac{202}{c-1}$.

[12] If $c > d/t \ge 1$, for constant c, let $\psi = d/tc$.

- *Part 2.* For $\zeta > 1$, let $\Pi^{\zeta}_{\mathcal{G}_i}$ be the probability distribution on vertices of $\mathcal{G}_i$; $i = 1, 2$ weighing each vertex v proportional to $deg(v)^{\zeta}$. If we take a sample from distribution $\Pi^{\zeta}_{\mathcal{G}_1}$ it will be STARRED vertex with probability $(t^{\zeta-1}+1)/2(t^{\zeta-1}+d^{\zeta}+1)$. Thus, for $t^{\zeta-1} \geq d^{\zeta}$ we have, $|\Pi^{\zeta}_{\mathcal{G}_1}, \Pi^{\zeta}_{\mathcal{A}}|_1 \geq 1/4$. Thus, $|\Pi^{0}_{\mathcal{G}_1}, \Pi_{\mathcal{A}}|_1 \geq 24/100 - 202/c$. □

* **Proof of Theorem 4.** Take the two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, $\mathcal{G}_1$ being the graph of Lemma 7, and $\mathcal{G}_2$ the same graph without the STARRED vertices (the graph before the decoration construction). We have: $\mathbb{E}(|V_{\mathcal{G}_2}|) = n$, $\mathbb{E}(|V_{\mathcal{G}_1}|) = (1+c_1)n$, and $\mathbb{E}(\mathrm{d_{avg}}(\mathcal{G}_2)|) = d$, $\mathbb{E}(|\,\mathrm{d_{avg}}(\mathcal{G}_1)|) = \frac{d+c_1}{1+c_1}$. □

* **Proof of Theorem 3.** Take $c_1 = 1$, and let $\mathcal{G} = \langle V_{\mathcal{G}}, E_{\mathcal{G}} \rangle$ be the graph constructed as in Lemma 7. Consider two functions $\mathcal{F}_1 : V_{\mathcal{G}} \to [0,1]$ and $\mathcal{F}_2 : V_{\mathcal{G}} \to [0,1]$. Let the function $\forall v \in V_{\mathcal{G}} \setminus \text{STARRED}, \mathcal{F}_1(v) = \mathcal{F}_2(v) = 0$. For any $v \in$ STARRED we set $\mathcal{F}_1(v) = 1$ with probability $1/2 + \epsilon$ and $\mathcal{F}_2(v) = 1$ with probability $1/2 - \epsilon$.

Note that $|\mathcal{F}_1 - \mathcal{F}_2|_1 \geq \epsilon$, and by employing the following classical result [6], with probability $1 - o(1)$ we will not be able to distinguish between $\mathcal{F}_1$ and $\mathcal{F}_2$.

▶ **Lemma 8** ([6]). *Consider a $\left(\frac{1}{2}-\epsilon, \frac{1}{2}+\epsilon\right)$-biased coin (that is, a coin whose most likely outcome has probability $\frac{1}{2}+\epsilon$. To determine with probability at least $1-\delta$ what is the most likely outcome of the coin, one needs at least $\Omega(1/\epsilon^2 \log(1/\delta))$ coin flips.*

By Lemma 7 with probability $1 - o(1)$, the expected number of STARRED centers will be $\frac{8q}{dt(1-o(1))}$. Let QUERIED be the set of queried vertices by the algorithm. If $q = \omega(dt)$

$$\mathbb{P}\left(\#\ \text{QUERIED} \cap \text{STARRED}_{\text{ center}} \geq 48q/dt\right) \leq e^{-16q/dt} \leq o(1).$$

Therefore, since in order to distinguish between $\mathcal{F}_1$ and $\mathcal{F}_2$ with probability at least $1-\delta$, we need at to see at least $\Omega\left(\log(1/\delta)(1/\epsilon^2)\right)$ starred centers, or equivalently $\Omega\left(dt \log(1/\delta)(1/\epsilon^2)\right)$ queries. □

4 Conclusion

In this paper we have studied the complexity of computing a number of functions of online graphs, such as online social networks, in terms of their average degree and their mixing time. We have obtained a tight bound for the problem of computing the average of a bounded function on the vertices of the graph (e.g., the average star rating of a movie), and a near-tight bound for the problem of sampling a close-to uniform-at-random vertex (many algorithms in the literature assume to have access to such an oracle), and lower bounds for the problems of estimating the order, and the average degree of the graphs.

It will be interesting to pursue the study of these online graphs problems in order to bridge the gap between theoretical algorithms, and applied ones. Besides the obvious questions (what are the optimal bounds for estimating the order and the average degree of a graph?), an interesting open problem is to understand which structural properties of online social networks could be used by algorithms to improve the complexity of the various problems that practitioners have been considering.

References

1 Yong-Yeol Ahn, Seungyeop Han, Haewoon Kwak, Sue Moon, and Hawoong Jeong. Analysis of topological characteristics of huge online social networking services. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 835–844, New York, NY, USA, 2007. ACM. doi:10.1145/1242572.1242685.

2 Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient algorithms for large-scale local triangle counting. *ACM Trans. Knowl. Discov. Data*, 4(3):13:1–13:28, October 2010.

3 Anna Ben-Hamou, Roberto I. Oliveira, and Yuval Peres. *Estimating graph parameters via random walks with restarts*, pages 1702–1714. SIAM, 2018. doi:10.1137/1.9781611975031.111.

4 E. Blais, C. Canonne, S. Chakraborty, G. Kamath, and C. Seshadhri. Property testing review,the latest in property testing and sublinear time algorithms(blog post). URL: https://ptreview.sublinear.info/?p=918.

5 Marco Bressan, Enoch Peserico, and Luca Pretto. Simple set cardinality estimation through random sampling. arXiv:1512.07901, 2015.

6 Ran Canetti, Guy Even, and Oded Goldreich. Lower bounds for sampling algorithms for estimating the average. *Information Processing Letters*, 53(1):17–25, 1995. doi:10.1016/0020-0190(94)00171-T.

7 Flavio Chierichetti, Anirban Dasgupta, Ravi Kumar, Silvio Lattanzi, and Tamás Sarlós. On sampling nodes in a network. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 471–481, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee. doi:10.1145/2872427.2883045.

8 Kai-Min Chung, Henry Lam, Zhenming Liu, and Michael Mitzenmacher. Chernoff-Hoeffding bounds for markov chains: Generalized and simplified. In Thomas Wilke Christoph Dürr, editor, *STACS'12 (29th Symposium on Theoretical Aspects of Computer Science)*, volume 14, pages 124–135, Paris, France, 2012. LIPIcs. URL: https://hal.archives-ouvertes.fr/hal-00678208.

9 Colin Cooper, Tomasz Radzik, and Yiannis Siantos. Estimating network parameters using random walks. *Social Network Analysis and Mining*, 4(1):168, 2014. doi:10.1007/s13278-014-0168-6.

10 Anirban Dasgupta, Ravi Kumar, and Tamas Sarlos. On estimating the average degree. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 795–806, New York, NY, USA, 2014. ACM. doi:10.1145/2566486.2568019.

11 Anirban Dasgupta, Ravi Kumar, and D. Sivakumar. Social sampling. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 235–243, New York, NY, USA, 2012. ACM. doi:10.1145/2339530.2339572.

12 Talya Eden and Will Rosenbaum. On sampling edges almost uniformly. arXiv:1706.09748, 2017.

13 Uriel Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM J. Comput.*, 35(4):964–984, 2006. doi:10.1137/S0097539704447304.

14 Minas Gjoka, Maciej Kurant, Carter T. Butts, and Athina Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, pages 2498–2506, Piscataway, NJ, USA, 2010. IEEE Press. URL: http://dl.acm.org/citation.cfm?id=1833515.1833840.

15 Oded Goldreich. *Introduction to Testing Graph Properties*, pages 105–141. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-16367-8_7.

16 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random Struct. Algorithms*, 32(4):473–493, 2008. doi:10.1002/rsa.v32:4.

17 Varun Kanade, Frederik Mallmann-Trenn, and Victor Verdugo. How large is your graph? *CoRR*, abs/1702.03959, 2017. URL: http://arxiv.org/abs/1702.03959, arXiv:1702.03959.

18 Liran Katzir, Edo Liberty, Oren Somekh, and Ioana A. Cosma. Estimating sizes of social networks via biased sampling. *Internet Mathematics*, 10(3-4):335–359, 2014. doi:10.1080/15427951.2013.862883.

19 Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 177–187, New York, NY, USA, 2005. ACM. doi:10.1145/1081870.1081893.

20 Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and MichaelW̃. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Math.*, 6(1):29–123, 2009. URL: http://projecteuclid.org/euclid.im/1283973327.

21 Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 29–42, New York, NY, USA, 2007. ACM. doi:10.1145/1298306.1298311.

22 Thomas Schank and Dorothea Wagner. Approximating clustering coefficient and transitivity. *Journal of Graph Algorithms and Applications*, 9:265–275, 2005. doi:10.7155/jgaa.00108.

23 C. Seshadhri, Ali Pinar, and Tamara G. Kolda. Wedge sampling for computing clustering coefficients and triangle counts on large graphs. *Stat. Anal. Data Min.*, 7(4):294–307, 2014. doi:10.1002/sam.11224.

The Price of Stability of Weighted Congestion Games

George Christodoulou[1]
Department of Computer Science, University of Liverpool, Liverpool, UK
G.Christodoulou@liverpool.ac.uk

Martin Gairing
Department of Computer Science, University of Liverpool, Liverpool, UK
gairing@liverpool.ac.uk

Yiannis Giannakopoulos[2]
Department of Mathematics, TU Munich, Munich, Germany
Yiannis.Giannakopoulos@tum.de
https://orcid.org/0000-0003-2382-1779

Paul G. Spirakis
Department of Computer Science, University of Liverpool, Liverpool, UK
Computer Engineering and Informatics Department, University of Patras, Patras, Greece
P.Spirakis@liverpool.ac.uk
https://orcid.org/0000-0001-5396-3749

Abstract

We give exponential lower bounds on the Price of Stability (PoS) of weighted congestion games with polynomial cost functions. In particular, for any positive integer d we construct rather simple games with cost functions of degree at most d which have a PoS of at least $\Omega(\Phi_d)^{d+1}$, where $\Phi_d \sim d/\ln d$ is the unique positive root of equation $x^{d+1} = (x+1)^d$. This essentially closes the huge gap between $\Theta(d)$ and Φ_d^{d+1} and asymptotically matches the Price of Anarchy upper bound. We further show that the PoS remains exponential even for singleton games. More generally, we also provide a lower bound of $\Omega((1+1/\alpha)^d/d)$ on the PoS of α-approximate Nash equilibria, even for singleton games. All our lower bounds extend to network congestion games, and hold for mixed and correlated equilibria as well.

On the positive side, we give a general upper bound on the PoS of α-approximate Nash equilibria, which is sensitive to the range W of the player weights and the approximation parameter α. We do this by explicitly constructing a novel approximate potential function, based on Faulhaber's formula, that generalizes Rosenthal's potential in a continuous, analytic way. From the general theorem, we deduce two interesting corollaries. First, we derive the existence of an approximate pure Nash equilibrium with PoS at most $(d+3)/2$; the equilibrium's approximation parameter ranges from $\Theta(1)$ to $d+1$ in a smooth way with respect to W. Secondly, we show that for unweighted congestion games, the PoS of α-approximate Nash equilibria is at most $(d+1)/\alpha$.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory, Theory of computation → Quality of equilibria, Theory of computation → Network games

Keywords and phrases Congestion games, price of stability, Nash equilibrium, approximate equilibrium, potential games

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.150

[1] Supported by EPSRC grant EP/M008118/1.
[2] Supported by the Alexander von Humboldt Foundation with funds from the German Federal Ministry of Education and Research (BMBF).

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 150; pp. 150:1–150:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version A full version of this paper is available at [26], `https://arxiv.org/abs/1802.09952`.

1 Introduction

In the last 20 years, a central strand of research within Algorithmic Game Theory has focused on understanding and quantifying the inefficiency of equilibria compared to centralized, optimal solutions. There are two standard concepts that measure this inefficiency. The Price of Anarchy (PoA) [34] which takes the worst-case perspective, compares the worst-case equilibrium with the system optimum. It is a very robust measure of performance. On the other hand, the Price of Stability (PoS) [44, 5], which is also the focus of this work, takes an optimistic perspective, and uses the best-case equilibrium for this comparison. The PoS is an appropriate concept to analyse the ideal solution that we would like our protocols to produce.

The initial set of problems that arose from the Price of Anarchy theory have now been resolved. The most rich and well-studied among these models are, arguably, the atomic and non-atomic variants of congestion games (see [38, Ch. 18] for a detailed discussion). This class of games is very descriptive and captures a large variety of scenarios where users compete for resources, most prominently routing games. The seminal work of Roughgarden and Tardos [42, 43] gave the answer for the non-atomic variant, where each player controls a negligible amount of traffic. Awerbuch et al. [6], Christodoulou and Koutsoupias [17] resolved the Price of Anarchy for atomic congestion games with affine latencies, generalized by Aland et al. [3] to polynomials; this led to the development of Roughgarden's Smoothness Framework [41] which extended the bounds to general cost functions, but also distilled and formulated previous ideas to bound the Price of Anarchy in an elegant, unified framework. At the computational complexity front, we know that even for simple congestion games, finding a (pure) Nash equilibrium is a PLS-complete problem [21, 2].

Allowing the players to have different loads, gives rise to the class of *weighted* congestion games [40]; this is a natural and very important generalization of congestion games, with numerous applications in routing and scheduling. Unfortunately though, an immediate dichotomy between weighted and unweighted congestion games occurs: the former may *not* even have pure Nash equilibria [36, 25, 27, 30]; as a matter of fact, it is a strongly NP-hard problem to even determine if that's the case [20]. Moreover, in such games there does not, in general, exist a potential function [37, 31], which is the main tool for proving equilibrium existence in the unweighted case.

As a result, a sharp contrast with respect to our understanding of the two aforementioned inefficiency notions arises. The Price of Anarchy has been studied in depth and general techniques for providing tight bounds are known. Moreover, the asymptotic behaviour of weighted and unweighted congestion games with respect to the Price of Anarchy is identical; it is $\Theta(d/\log d)^d$ for both classes when latencies are polynomials of degree at most d [3].

The situation for the Price of Stability though, is completely different. For unweighted games we have a good understanding[3] and the values are much lower than the Price of

[3] Much work has been done on the PoS for network design games, which is though not so closely related to our work. This problem was first studied by Anshelevich et al. [5] who showed a tight bound of H_n, the harmonic number of the number of players n, for directed networks. Finding tight bounds on undirected networks is still a long-standing open problem (see, e.g., [23, 9, 35]). Recently, Bilò et al. [10] (asymptotically) resolved the question for broadcast networks. For the weighted variant of this problem, Albers [4] showed a lower bound of $\Omega(\log W/\log\log W)$, where W is the sum of the players'

Table 1 Previous results (*left table*) regarding the Price of Anarchy and Stability for unweighted and weighted congestion games, with polynomial latency functions of maximum degree d. Φ_d is the unique positive solution of $(x+1)^d = x^{d+1}$ and $\Phi_d = \Theta(d/\log d)$. Tight answers were known for all settings, except for the Price of Stability of the weighted case were only trivial bounds existed. In this paper (*right table*) we essentially close this gap by showing a lower bound of $\Omega(\Phi_d)^{d+1}$ (Theorem 1), even for network games, which is exponential even for singleton games Theorem 5.

	PoA	PoS
unweighted	$\lfloor\Phi_d\rfloor^{d+1}$ [3]	$\Theta(d)$ [15]
weighted	Φ_d^{d+1} [3]	$[\Theta(d), \Phi_d^{d+1}]$

	PoS lower bound
general	$\Omega(\Phi_d)^{d+1}$
singleton	$\Omega(2^d/d)$
α-approximate equilibria	$\Omega((1+1/\alpha)^d/d)$

Anarchy values, and also *tight*; approximately 1.577 for affine functions [16, 14], and $\Theta(d)$ [15] for polynomials. For weighted games though there is a huge gap; the current state of the art lower bound is $\Theta(d)$ and the upper bound is $\Theta(d/\ln d)^d$. These previous results are summarized at the left of Table 1.

The main focus of this work is precisely to deal with this lack of understanding, and to determine the Price of Stability of weighted congestion games. What makes this problem challenging is that the only general known technique for showing upper bounds for the Price of Stability is the potential method, which is applicable only to potential games. In a nutshell, the idea of this method is to use the global minimizer of Rosenthal's potential [39] as an equilibrium refinement. This equilibrium is also a pure Nash equilibrium and can serve as an upper bound of the Price of Stability. Interestingly, it turns out that, for several classes of potential games, this technique actually provides the tight answer (see for example [5, 16, 14, 15]). However, as already mentioned above, unlike their unweighted counterparts, weighted congestion games are not potential games;[4] so, a completely fresh approach is required.

One way to override the aforementioned limitations of non-existence of pure Nash equilibria, but also their computational hardness, is to consider *approximate* equilibria. In this direction, Hansknecht et al. [29] have shown that $(d+1)$-approximate pure Nash equilibria always exist in weighted congestion games with polynomial latencies of maximum degree d, while, in the negative side, there exist games that do not have 1.153-approximate pure Nash equilibria. Notice here, that these results do not take into account computational complexity considerations; if we insist in polynomial-time algorithms for actually finding those equilibria, then the currently best approximation parameter becomes $d^{O(d)}$ [12, 13, 22].

1.1 Our Results

We provide lower and upper bounds on the Price of Stability for the class of weighted congestion games with polynomial latencies with nonnegative coefficients. We consider both exact and approximate equilibria. Our lower bounds are summarized at the right of Table 1.

weights. See [10] and references therein for a thorough discussion of those results.

[4] For the special case of weighted congestion games with linear latency functions, a potential does exist [25] and this was used by [8] to provide a PoS upper bound of 2.

Lower Bound for Weighted Congestion Games. In our main result in Theorem 1, we resolve a long-standing open problem by providing almost tight bounds for the Price of Stability of weighted congestion games with polynomial latency functions. We construct an instance having a Price of Stability of $\Omega(\Phi_d)^{d+1}$, where d is the maximum degree of the latencies and $\Phi_d \sim \frac{d}{\ln d}$ is the unique positive solution of equation $(x+1)^d = x^{d+1}$.

This bound essentially closes the previously huge gap between $\Theta(d)$ and Φ_d^{d+1} for the PoS of weighted congestion games. The previously best lower and upper bounds were rather trivial: the lower bound corresponds to the PoS results of Christodoulou and Gairing [15] for the unweighted case (and thus, it is also a valid lower bound for the general weighted case as well) and the upper bound comes from the Price of Anarchy results of Aland et al. [3] (PoA, by definition, upper-bounds PoS).

We stress that, although as mentioned before, weighted congestion games do not always possess pure equilibria, our lower bound construction involves a *unique* equilibrium occurring by iteratively eliminating strongly dominated strategies. As a result, this lower bound holds not only for pure, but mixed and correlated equilibria as well.

Singleton Games. Next we switch to the class of singleton congestion games, where a pure strategy for each player is a *single* resource. This class is very well-studied as, on one hand, it abstracts scheduling environments, and on the other, it has very attractive equilibrium properties; unlike general weighted congestion games, there exists an (ordinal) lexicographic potential [24, 32], thus implying the existence of *pure* Nash equilibria. It is important to note that, the tight lower bounds for the Price of Anarchy of general weighted congestion games, hold also for the class of singleton games [14, 7, 11].

Nevertheless, even for this special class, we show in Theorem 5 an exponential lower bound of $\Omega(2^d/d)$. The previous best upper and lower bounds were the same as those of the general case, namely $\Theta(d)$ and Φ_d^{d+1}, respectively. As a matter of fact, this new lower bound comes as a corollary of a more general result that we show in Theorem 5, that extends to approximate equilibria and gives a lower bound of $\Omega((1+1/\alpha)^d/d)$ on the PoS of α-approximate equilibria, for any (multiplicative) approximation parameter $\alpha \in [1, d)$. Setting $\alpha = 1$ we recover the special case of exact equilibria and the aforementioned exponential lower bound on the standard, exact notion of the PoS. Notice here that, as we show in [26, Appendix D], the optimal solution (which, in general, is not an equilibrium) itself constitutes a $(d+1)$-approximate equilibrium with a (trivially) optimal PoS of 1.

Positive Results for Approximate Equilibria. In light of the above results, in Section 4, we turn our attention to identifying environments with more structure or flexibility with respect to the underlying solution concept, for which we can hope for improved quality of equilibria. Both our lower bound constructions discussed above use players' weights that form a geometric sequence. In particular the ratio W of the largest over the smallest weight is equal to w^n (for some $w > 1$), which grows very large as the number of players $n \to \infty$. On the other hand, for games where the players have equal weights, i.e. $W = 1$, we know that the PoS is at most $d+1$. It is therefore natural to ask how the performance of the good equilibria captured by the notion of PoS varies with respect to W. In Theorem 9, we are able to give a general upper bound for α-approximate equilibria which is sensitive to this parameter W and to α. This general theorem has two immediate, interesting corollaries.

Firstly (Corollary 10), by allowing the ratio W to range in $[1, \infty)$, we derive the existence of an α-approximate pure Nash equilibrium with PoS at most $(d+3)/2$; the equilibrium's approximation parameter α ranges from $\Theta(1)$ to $d+1$ in a smooth way with respect to W.

This is of particular importance in settings where player weights are not very far away from each other (that is, W is small). Secondly (Corollary 11), by setting $W = 1$ and allowing α to range, we get an upper bound of $\frac{d+1}{\alpha}$ for the α-approximate PoS of *unweighted* congestion games which, to the best of our knowledge, was not known before, degrading gracefully from $d + 1$ (which is the actual PoS of exact equilibria in the unweighted case [15]) down to the optimal value of 1 if we allow $(d + 1)$-approximate equilibria (which in fact can be achieved by the optimum solution itself; see [26, Appendix D]).

Our Techniques. An advantage of our main lower bound (Theorem 1) is the simplicity of the underlying construction, as well as its straightforward adaptation to network games (see Section 3.1.1)). However, fine-tuning the parameters of the game (player weights and latency functions), to ensure uniqueness of the equilibrium at the "bad" instance, was a technically involved task. This was in part due to the fact that, in order to guarantee uniqueness (via iteratively dominant strategies), each player interacts with a window of μ other players. This μ depends on d in a delicate way (see [26, Fig. 1] and Lemma 2); it has to be an integer but, at the same time, needs also to balance nicely with the algebraic properties of Φ_d. Moreover we needed to provide deeper insights on the asymptotic, analytic behaviour of Φ_d, and to explore some new algebraic characteristics of Φ_d (see, e.g., [26, Lemma 7]).

In order to derive our upper bounds, we need to define a novel *approximate potential function* [18, 29]. First, in Lemma 6, we identify clear algebraic sufficient conditions for the existence of approximate equilibria with good social-cost guarantees, and then explicitly define (see the (8) and the proof of Theorem 9 in [26]) a function that satisfies them. This continuous function, which is defined in the entire space of positive reals, essentially generalizes that of Rosenthal's in a smooth way: by setting $W = \alpha = 1$, we recover exactly the first significant terms of the well known Rosenthal potential [39] polynomial, with which one can demonstrate the usual PoS results for the unweighted case (see, e.g. [16]). The simple, analytic way in which this function is defined, is the very reason why we can handle both the approximation parameter α of the equilibrium and the ratio W of the weights in a smooth manner while at the same time providing good PoS guarantees.

It is important to stress that, by the purely analytical way in which our approximate potential function is defined, in principle it can also incorporate more general cost functions than polynomials; so, we believe that this technique may be of independent interest. We point towards that direction in [26, Appendix C].

Due to space limitations, all omitted proofs and various supplementary figures can be found in the full version of our paper [26].

2 Model and Notation

Weighted Congestion Games. A *weighted congestion game* consists of a finite, nonempty set of players N and resources (or facilities) E. Each player $i \in N$ has a real *weight* w_i 0 and a *strategy set* $S_i \subseteq 2^E$. Associated with each resource $e \in E$ is a *cost* (or *latency*) *function* $c_e : (0, \infty) \longrightarrow [0, \infty)$. In this paper we mainly focus on polynomial cost functions with maximum degree $d \geq 0$ and nonnegative coefficients; that is, every cost function is of the form $c_e(x) = \sum_{j=0}^{d} a_{e,j} \cdot x^j$, with $a_{e,j} \geq 0$ for all j. In the following, whenever we refer to polynomial cost functions we mean cost functions of this particular form.

A *pure strategy profile* is a choice of strategies $\mathbf{s} = (s_1, s_2, ...s_n) \in S = S_1 \times \cdots \times S_n$ by the players. We use the standard game-theoretic notation $\mathbf{s}_{-i} = (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots s_n)$, $S_{-i} = S_1 \times \cdots \times S_{i-1} \times S_{i+1} \times \cdots \times S_n$, such that $\mathbf{s} = (s_i, \mathbf{s}_{-i})$. Given a pure strategy profile

$\mathbf{s}$, we define the *load* $x_e(\mathbf{s})$ of resource $e \in E$ as the total weight of players that use resource e on $\mathbf{s}$, i.e., $x_e(\mathbf{s}) = \sum_{i \in N : e \in s_i} w_i$. The *cost* player i is defined by $C_i(\mathbf{s}) = \sum_{e \in s_i} c_e(x_e(\mathbf{s}))$.

A *singleton* weighted congestion game is a special form of congestion games where the strategies of all players consist only of single resources; that is, for all players $i \in N$, $|s_i| = 1$ for all $s_i \in S_i$. In a weighted *network* congestion games the resources E are given as the edge set of some directed graph $G = (V, E)$, and each player $i \in N$ has a source $o_i \in V$ and destination $t_i \in V$ node; then, the strategy set S_i of each player is implicitly given as the edge sets of all directed $o_i \to t_i$ paths in G.

Nash Equilibria. A pure strategy profile $\mathbf{s}$ is a pure *Nash equilibrium* if and only if for every player $i \in N$ and for all $s'_i \in S_i$, we have $C_i(\mathbf{s}) \leq C_i(s'_i, \mathbf{s}_{-i})$. Similarly a strategy profile is an *α-approximate pure Nash equilibrium*, for $\alpha \geq 1$, if $C_i(\mathbf{s}) \leq \alpha \cdot C_i(s'_i, \mathbf{s}_{-i})$ for all players $i \in N$ and $s'_i \in S_i$. As discussed in the introduction, weighted congestion games do not always admit pure Nash equilibria. However, by Nash's theorem they have mixed Nash equilibria. A tuple $\sigma = (\sigma_1, \cdots, \sigma_N)$ of independent probability distributions over players' strategy sets is a *mixed Nash equilibrium* if

$$\mathop{\mathbb{E}}_{\mathbf{s} \sim \sigma}[C_i(\mathbf{s})] \leq \mathop{\mathbb{E}}_{\mathbf{s}_{-i} \sim \sigma_{-i}}[C_i(s'_i, \mathbf{s}_{-i})]$$

holds for every $i \in N$ and $s'_i \in S_i$. Here σ_{-i} is a product distribution of all σ_j's with $j \neq i$, and $\mathbf{s}_{-i}$ denotes a strategy profile drawn from this distribution. We use $\mathrm{NE}(G)$ to denote the set of all mixed Nash equilibria of a game G.

Social Cost and Price of Stability. Fix a weighted congestion game G. The *social cost* of a pure strategy profile $\mathbf{s}$ is the weighted sum of the players' costs

$$C(\mathbf{s}) = \sum_{i \in N} w_i \cdot C_i(\mathbf{s}) = \sum_{e \in E} x_e(\mathbf{s}) \cdot c_e(x_e(\mathbf{s})).$$

Denote by $\mathrm{OPT}(G) = \min_{\mathbf{s} \in S} C(\mathbf{s})$ the *optimum social cost* over all strategy profiles $\mathbf{s} \in S$. Then, the *Price of Stability (PoS)* of G is the social cost of the best-case Nash equilibrium over the optimum social cost:

$$\mathrm{PoS}(G) = \min_{\sigma \in \mathrm{NE}(G)} \frac{\mathbb{E}_{\mathbf{s} \sim \sigma}[C(\mathbf{s})]}{\mathrm{OPT}(G)}.$$

The Price of Stability of α-approximate Nash equilibria is defined accordingly. The PoS for a class $\mathcal{G}$ of games is the worst (i.e., largest) PoS among all games in the class, that is, $\mathrm{PoS}(\mathcal{G}) = \sup_{G \in \mathcal{G}} \mathrm{PoS}(G)$. For example, our focus in this paper is determining the Price of Stability for the class $\mathcal{G}$ of weighted congestion games with polynomial cost functions.

Finally, notice that, by using a straightforward scaling argument, it is without loss with respect to the PoS metric to analyse games with player weights in $[1, \infty)$; if not, divide all w_i's with $\min_i w_i$ and scale cost functions accordingly.

3 Lower Bounds

In this section, we present our lower bound constructions. In Section 3.1 we present the general lower bound and then in Section 3.2 the lower bound for singleton games.

3.1 General Congestion Games

The next theorem presents our main negative result on the Price of Stability of weighted congestion games with polynomial latencies of degree d, that essentially matches the Price of Anarchy upper bound of Φ_d^{d+1} from Aland et al. [3]. Our result, shows a strong separation of the Price of Stability of weighted and unweighted congestion games, where the Price of Stability is at most $d+1$ [15]. This is in sharp contrast to the Price of Anarchy of these two classes, where the respective bounds are essentially the same.

We will need to introduce some notation. Let $\Phi_d = \Theta\left(\frac{d}{\ln d}\right)$ be the unique positive root of the equation $(x+1)^d = x^{d+1}$ and let β_d be a parameter such that $\beta_d \geq 0.38$ for any d, $\lim_{d\to\infty} \beta_d = \frac{1}{2}$. A plot of its values can be seen in [26, Fig. 1].

▶ **Theorem 1.** *The Price of Stability of weighted congestion games with polynomial latency functions of degree at most $d \geq 9$ is at least $(\beta_d \Phi_d)^{d+1}$.*

▶ **Lemma 2.** *For any positive integer d define*

$$c_d = \frac{1}{d}\left\lfloor d\frac{\ln(\Phi_d^{1+2/d} - \Phi_d) - \ln(\Phi_d^{1+2/d} - \Phi_d - 1)}{\ln \Phi_d}\right\rfloor \qquad \text{and} \qquad \beta_d = 1 - \Phi_d^{-c_d}, \tag{1}$$

Then $\Phi_d^{d+2} \leq \left(\Phi_d + \frac{1}{\beta_d}\right)^d$; furthermore, for all $d \geq 9$ we have: $d \cdot c_d \geq 3$, $0.38 \leq \beta_d \leq \frac{1}{2}$ and $\lim_{d\to\infty} \beta_d = \frac{1}{2}$.

Proof of Theorem 1. We now move on to the description of our congestion game instance. Fix some integer[5] $d \geq 9$. Our instance consists of $n+\mu$ players and $n+\mu+1$ facilities, where $\mu \equiv c \cdot d$ for some real $c \geq \frac{3}{d}$ (to be specifically determined later on, see (1)) such that $\mu \geq 3$ is an integer. You can think of n as a very large integer, since at the end we will take $n \to \infty$. Every player $i = 1, 2, \ldots, n+\mu$ has a weight of $w_i = w^i$, where $w = 1 + \frac{1}{\Phi_d}$.

It will be useful for subsequent computations to notice that $w^d = \left(1 + \frac{1}{\Phi_d}\right)^d = \frac{(\Phi_d+1)^d}{\Phi_d^d} = \frac{\Phi_d^{d+1}}{\Phi_d^d} = \Phi_d$ and $w^{d+1} = w^d \cdot w = \Phi_d\left(1 + \frac{1}{\Phi_d}\right) = \Phi_d + 1$. Let also define

$$\alpha = \alpha(\mu) \equiv \sum_{j=1}^{\mu} w^{-j} = \frac{1 - w^{-\mu}}{w - 1} = \frac{1 - (w^d)^{-c}}{w - 1} = \frac{1 - \Phi_d^{-c}}{1 + \frac{1}{\Phi_d} - 1} = \Phi_d\left(1 - \Phi_d^{-c}\right) = \beta_d \Phi_d,$$

where $\beta_d \equiv 1 - \Phi_d^{-c} \in (0,1)$. Observe that

$$w^{-\mu} = 1 - \beta_d \Phi_d (w-1) = 1 - \beta_d \Phi_d \left(1 + \frac{1}{\Phi_d} - 1\right) = 1 - \beta_d$$

and furthermore, for every $i \geq \mu + 1$, $\sum_{j=i-\mu}^{i-1} w_j = \sum_{j=1}^{\mu} w^{i-j} = \alpha \cdot w^i$, $\sum_{j=i-\mu}^{i} w_j = (\alpha+1) \cdot w^i$ and $\sum_{\ell=1}^{\infty} w^{-\ell} = \frac{1}{w-1} = \frac{1}{1+\frac{1}{\Phi_d}-1} = \Phi_d$.

The facilities have latency functions

$$\begin{aligned} c_j(t) &= (\alpha+1)^d, && \text{if } j = 1, \ldots, \mu, \\ c_j(t) &= w^{-j(d+1)} t^d, && \text{if } j = \mu+1, \ldots, \mu+n, \\ c_{n+\mu+1}(t) &= 0, \end{aligned}$$

where for simplicity we use j instead e_j to refer to the j-th facility.

[5] For polynomial latencies of smaller degrees $d \leq 8$ we can instead apply the simpler lower-bound instance for singleton games given in Section 3.2.

Every player i has two available strategies, s_i^* and $\tilde{s}_i$. Eventually we will show that the profile $\mathbf{s}^*$ corresponds to the optimal solution, while $\tilde{\mathbf{s}}$ corresponds to the *unique* Nash equilibrium of the game. Informally, at the former the player chooses to stay at her "own" i-th facility, while at the latter she chooses to deviate and play the μ following facilities $i+1,\ldots,i+\mu$. However, special care shall be taken for the boundary cases of the first μ and last μ players, so for any player i we formally define $S_i = \{s_i^*, \tilde{s}_i\}$ where $s_i^* = \{i\}$ and

$$\tilde{s}_i = \begin{cases} \{\mu+1,\ldots,\mu+i\}, & \text{if } i=1,\ldots,\mu, \\ \{i+1,\ldots,i+\mu\}, & \text{if } i=\mu+1,\ldots,n, \\ \{i+1,\ldots,n+\mu+1\}, & \text{if } i=n+1,\ldots,n+\mu. \end{cases}$$

These two outcomes, $\mathbf{s}^*$ and $\tilde{\mathbf{s}}$, are shown in [26, Fig. 2].

Notice here that any facility j cannot get a load greater than the sum of the weights of the previous μ players plus the weight of the j-th player. So, for and any strategy profile $\mathbf{s}$:

$$x_j(\mathbf{s}) \leq \sum_{\ell=j-\mu}^{j} w_\ell = (\alpha+1)w^j \qquad \text{for all } j \geq \mu+1 \tag{2}$$

Next we will show that the strategy profile $\tilde{\mathbf{s}} = (\tilde{s}_1,\ldots,\tilde{s}_{n+\mu})$ is the *unique* Nash equilibrium of our congestion game. We do that by proving that

1. It is a strongly *dominant* strategy for any player $i=1,\ldots,\mu$ to play $\tilde{s}_i$.
2. For any $i=\mu+1,\ldots,n+\mu$, given that every player $k<i$ has chosen to play $\tilde{s}_k$, then it is a strongly *dominant* strategy for player i to deviate to $\tilde{s}_i$ as well.

For the first condition, fix some player $i \leq \mu$ and a strategy profile $\mathbf{s}_{-i}$ for the other players and observe that by choosing $\tilde{s}_i$, player i incurs a cost of at most

$$\begin{aligned} C_i(\tilde{s}_i,\mathbf{s}_{-i}) &= \sum_{j\in\tilde{s}_i} c_j(x_j(\tilde{s}_i)) \leq \sum_{\ell=\mu+1}^{\mu+i} c_\ell\left((\alpha+1)w^\ell\right) \\ &= \sum_{\ell=d+1}^{d+i} w^{-\ell(d+1)}(\alpha+1)^d w^{\ell d} = (\alpha+1)^d \sum_{\ell=d+1}^{d+i} w^{-\ell} \\ &< (\alpha+1)^d w^{-d} \sum_{\ell=1}^{\infty} w^{-\ell} = (\alpha+1)^d \frac{1}{\Phi_d}\Phi_d = (\alpha+1)^d = C_i(s_i^*,\mathbf{s}_{-i}), \end{aligned}$$

where in the first inequality we used the bound from (2).

For the second condition, we will consider the deviations of the remaining players.[6] Fix now some $i=\mu+1,\ldots,n$ and assume a strategy profile $\mathbf{s}_{-i} = (\tilde{s}_1,\ldots,\tilde{s}_{i-1},s_{i+1},\ldots,s_{n+\mu})$ for the remaining players. If player i chooses strategy s_i^* she will experience a cost of

$$C_i(s_i^*,\mathbf{s}_{-i}) = c_i\left(\sum_{\ell=i-\mu}^{i} w_\ell\right) = c_i\left((\alpha+1)w^i\right) = w^{-i(d+1)}(\alpha+1)^d w^{id} = (\alpha+1)^d w^{-i}.$$

It remains to show that

$$C_i(\tilde{s}_i,\mathbf{s}_{-i}) < C_i(s_i^*,\mathbf{s}_{-i}) = (\alpha+1)^d w^{-i}. \tag{3}$$

[6] For the remaining last μ players $i=n+1,\ldots,n+\mu$ the proof is similar to the text, and as a matter of fact easier, since when these players deviate to $\tilde{s}_i$ they also use the final "dummy" facility $n+\mu+1$ that has zero cost.

The cost $C_i(\tilde{s}_i, \mathbf{s}_{-i})$ is complicated to bound immediately, for any profile $\mathbf{s}_{-i}$. Instead, we will resort to the following claim which characterizes the profile $\mathbf{s}_{-i}$ where this cost is maximized. [7]

▶ **Claim 3.** *There exists a profile* $\mathbf{s}'$ *with*

1. $s'_j = s_j$ *for all* $j \leq i$ *and* $i > i + \mu$
2. $s'_{i+\mu} = s^*_{i+\mu}$
3. *there exists some* $k \in \{i+1, \ldots, i+\mu-1\}$ *such that*

$$s'_j = \tilde{s}_j \qquad \text{for all } j \in \{i+1, \ldots, i+\mu-1\} \setminus \{k\},$$

that dominates $\mathbf{s}$*, i.e.* $C_i(\tilde{s}_i, \mathbf{s}_{-i}) \leq C_i(\tilde{s}_i, \mathbf{s}'_{-i})$.

By use of Claim 3, it remains to show

$$C_i(\tilde{s}_i, \mathbf{s}'_{-i}) < (\alpha+1)^d w^{-i}, \tag{4}$$

just for the special case of profiles $\mathbf{s}'$ that are described in Claim 3. We do this in [26, Appendix A.4].

Summarizing, we proved that indeed $\tilde{\mathbf{s}}$ is the *unique* Nash equilibrium of our congestion game. Finally, to conclude with lower-bounding the Price of Stability, let us compute the social cost on profiles $\tilde{\mathbf{s}}$ and $\mathbf{s}^*$. On $\mathbf{s}^*$, any facility j (except the last one) gets a load equal to the weight of player j, so

$$\begin{aligned}
C(\mathbf{s}^*) &= \sum_{j=1}^{n+\mu} w_j c_j(w_j) \\
&= \sum_{j=1}^{\mu} w^j (\alpha+1)^d + \sum_{j=\mu+1}^{n+\mu} w^j w^{-j(d+1)} (w^j)^d \\
&= (\alpha+1)^d \sum_{j=1}^{\mu} w^j + \sum_{j=\mu+1}^{\mu+n} 1 \\
&= (\alpha+1)^d w \frac{w^\mu - 1}{w-1} + n \\
&= n + (\beta\Phi_d + 1)^d \left(1 + \frac{1}{\Phi_d}\right) \frac{\frac{1}{1-\beta} - 1}{1 + \frac{1}{\Phi_d} - 1} \\
&= n + (\beta\Phi_d + 1)^d (\Phi_d + 1) \frac{\beta}{1-\beta} \\
&\leq n + \frac{\beta}{1-\beta} (\Phi_d + 1)^{d+1}.
\end{aligned}$$

On the other hand, at the unique Nash equilibrium $\tilde{\mathbf{s}}$ each facility $j \geq \mu + 1$ receives a load equal to the sum of the weights of the previous μ players, i.e.

$$x_j(\tilde{\mathbf{s}}) = \sum_{\ell=j-\mu}^{j-1} w_\ell = \alpha w^j$$

[7] For an explanatory figure and a proof of this claim, see the full version of our paper [26, Fig. 3].

so

$$C(\tilde{\mathbf{s}}) \geq \sum_{j=\mu+1}^{n+\mu} x_j(\tilde{\mathbf{s}}) c_j(x_j(\tilde{\mathbf{s}})) = \sum_{j=\mu+1}^{n+\mu} w^{-j(d+1)} \left(\alpha w^j\right)^{d+1} = \alpha^{d+1} \sum_{j=\mu+1}^{\mu+n} 1 = \alpha^{d+1} n.$$

By taking n arbitrarily large we get a lower bound on the Price of Stability of

$$\lim_{n\to\infty} \frac{C(\tilde{\mathbf{s}})}{C(\mathbf{s}^*)} \geq \lim_{n\to\infty} \frac{\alpha^{d+1} n}{n + \frac{\beta}{1-\beta}(\Phi_d + 1)^{d+1}} = \alpha^{d+1} = (\beta \Phi_d)^{d+1},$$

where from Lemma 2 we know that $\frac{1}{3} \leq \beta = \frac{1}{2} - o(1)$. ◀

3.1.1 Network Games

Due to the rather simple structure of the players' strategy sets in the lower bound construction of Theorem 1, it can be readily extended to network games as well:

▶ **Proposition 4.** *Theorem 1 applies also to* network *weighted congestion games.*

3.2 Singleton Games

In this section we give an exponential lower bound for *singleton* weighted congestion games with polynomial latency functions. The following theorem handles also approximate equilibria and provides a lower bound on the Price of Stability in a very strong sense; even if one allows for the best approximate equilibrium with approximation factor $\alpha = o\left(\frac{d}{\ln d}\right)$, then its cost is lower-bounded by $\omega(d)$ times the optimal cost.[8] In other words, in order to achieve linear guarantees on the Price of Stability, one has to consider $\Omega\left(\frac{d}{\ln d}\right)$-approximate equilibria—almost linear in d; this shows that our positive result in Corollary 10, of the following Section 4.3, is essentially tight. This is furthermore complemented by [26, Appendix D], where we show that the socially optimum profile is a $(d+1)$-approximate equilibrium (achieving an optimal Price of Stability of 1).

▶ **Theorem 5.** *For any positive integer d and any real $\alpha \in [1, d)$, the α-approximate (mixed) Price of Stability of weighted (singleton) congestion games with polynomial latencies of degree at most d is at least $\frac{1}{e(d+1)} \left(1 + \frac{1}{\alpha}\right)^{d+1}$. In particular, for the special case of $\alpha = 1$, we derive that the Price of Stability of* exact *equilibria is $\Omega(2^d/d) = (2 - o(1))^{d+1}$.*

4 Upper Bounds

The negative results of the previous sections, involve constructions where the ratio W of the largest to smallest weight can be exponential in d. In the main theorem (Theorem 9) of this section we present an analysis which is sensitive to this parameter W, and identify conditions under which the performance of approximate equilibria can be significantly improved.

Our upper bound approach is based on the design of a suitable approximate potential function and has three main steps. First, in Section 4.1, we set up a framework for the definition of this function by identifying conditions that, on the one hand, certify the existence of an approximate equilibrium and, on the other, provide guarantees about its efficiency.

[8] To see this, just take any upper bound of $\frac{d+1}{c \ln(d+1)}$ on α, for a constant $c > 2$. Then, the lower bound in Theorem 5 becomes $\Omega(d^{c-1})$.

Then, in Section 4.2, by use of the Euler-Maclaurin summation formula we present a general form of an approximate potential function, which extends Rosenthal's potential for weighted congestion games (see also [26, Appendix C]). Finally, in Section 4.3, we deploy this potential for polynomial latencies. Due to its analytic description, our potential differs from other extensions of the Rosenthal's potential that have appeared in previous work, and we believe that this contribution might be of independent interest, and applied to other classes of latency functions.

4.1 The Potential Method

In the next lemma we lay the ground for the design and analysis of approximate potential functions, by supplying conditions that not only provide guarantees for the existence of approximate equilibria, but also for their performance with respect to the social optimum. In the premises of the lemma, we give conditions on the resource functions ϕ_e, having in mind that $\Phi(\mathbf{s}) = \sum_{e\in E} \phi_e(x_e(\mathbf{s}))$ will eventually serve as the "approximate" potential function.

▶ **Lemma 6.** *Consider a weighted congestion game with latency functions c_e, for each facility $e \in E$, and player weights w_i, for each player $i \in N$. If there exist functions $\phi_e : [0,\infty) \longrightarrow \mathbb{R}$ and parameters $\alpha_1, \alpha_2, \beta_1, \beta_2 > 0$ such that for any facility e and player weight $w \in \{w_1, \ldots, w_n\}$*

$$\alpha_1 \leq \frac{\phi_e(x+w) - \phi_e(x)}{w \cdot c_e(x+w)} \leq \alpha_2, \qquad \text{for all } x \geq 0, \tag{5}$$

and

$$\beta_1 \leq \frac{\phi_e(x)}{x \cdot c_e(x)} \leq \beta_2, \qquad \text{for all } x \geq \min_n w_n, \tag{6}$$

then our game has an $\frac{\alpha_2}{\alpha_1}$-approximate pure Nash equilibrium which, furthermore, has Price of Stability at most $\frac{\beta_2}{\beta_1}$.

4.2 Faulhaber's Potential

In this section we propose an approximate potential function, which is based on the following classic number-theoretic result, known as Faulhaber's formula[9], which states that for any positive integers n, m,

$$\begin{aligned}\sum_{k=1}^{n} k^m &= \frac{1}{m+1}\sum_{j=0}^{m}(-1)^j \binom{m+1}{j} B_j n^{m+1-j} \\ &= \frac{1}{m+1}n^{m+1} + \frac{1}{2}n^m + \frac{1}{m+1}\sum_{j=2}^{m}\binom{m+1}{j} B_j n^{m+1-j},\end{aligned} \tag{7}$$

where the coefficients B_j are the usual Bernoulli numbers.[10] In particular, this shows that the sum of the first n powers of m can be expressed as a polynomial of n with degree $m+1$. Furthermore, this sum corresponds to the well-known potential of Rosenthal [39] for *unweighted* congestion games when the latency function is the monomial $x \mapsto x^m$.

[9] See, e.g., [33, p. 287] or [19, p. 106]).

[10] See, e.g., [28, Chapter 6.5] or [1, Chapter 23]. The first Bernoulli numbers are: $B_0 = 1, B_1 = -1/2, B_2 = 1/6, B_3 = 0, B_4 = -1/30, \ldots$. Also, we know that $B_j = 0$ for all *odd* integers $j \geq 3$.

Based on the above observation, we go beyond just integer values of n, and generalize this idea to all positive reals; in that way, we design a "potential" function that can handle different player weights and, furthermore, incorporate in a more powerful, analytically smooth way, approximation factors with respect to both the Price of Stability, as well as the approximation parameter of the equilibrium (in the spirit of Lemma 6). A natural way to do that is to directly generalize (7) and simply define, for any real $x \geq 0$ and positive integer m,

$$S_m(x) \equiv \frac{1}{m+1}x^{m+1} + \frac{1}{2}x^m, \tag{8}$$

keeping just the first two significant terms.[11] For the special case of $m = 0$ we set $S_0(y) = y$.

For any nonnegative integer m we define the function $A_m : [1, \infty) \longrightarrow (0, \infty)$ with

$$A_m(x) \equiv \left[\frac{S_m(x)}{x^{m+1}}\right]^{-1} = \left(\frac{1}{m+1} + \frac{1}{2x}\right)^{-1} = \frac{2(m+1)x}{2x+m+1}. \tag{9}$$

Observe that A_m is strictly increasing (in x) for all $m \geq 1$,

$$A_m(1) = \frac{2(m+1)}{m+3} \in [1, 2), \qquad \text{and} \qquad \lim_{x\to\infty} A_m(x) = m+1. \tag{10}$$

For the special case of $m = 0$ we simply have $A_0(x) = 1$ for all $x \geq 0$. A graph of these functions can be found in [26, Fig. 6]. Since A_m is strictly increasing for $m \geq 1$, its inverse function, $A_m^{-1} : [2\frac{m+1}{m+3}, m+1] \longrightarrow [1, \infty)$, is well-defined and also strictly increasing for all $m \geq 1$.

The following two lemmas describe some useful properties regarding the algebraic behaviour, and the relation among, functions A_m and S_m:

▶ **Lemma 7.** *Fix any reals $y \geq x \geq 1$. Then the sequences $\frac{A_m(x)}{m+1}$ and $\frac{A_m(x)}{A_m(y)}$ are decreasing, and sequence $A_m(x)$ is increasing (with respect to m).*

▶ **Lemma 8.** *Fix any integer $m \geq 0$ and reals $\gamma, w \geq 1$. Then*

$$\frac{\gamma^{m+1}}{A_m(\gamma w)} \leq \frac{S_m(\gamma(x+w)) - S_m(\gamma x)}{w(x+w)^m} \leq \gamma^{m+1}, \qquad \textit{for all } x \geq 0, \tag{11}$$

and

$$\frac{\gamma^{m+1}}{m+1} \leq \frac{S_m(\gamma x)}{x^{m+1}} \leq \frac{\gamma^{m+1}}{A_m(\gamma)}, \qquad \textit{for all } x \geq 1. \tag{12}$$

4.3 The Upper Bound

Now we are ready to state our main positive result:

▶ **Theorem 9.** *At any congestion game with polynomial latency functions of degree at most $d \geq 1$ and player weights ranging in $[1, W]$, for any $\frac{2(d+1)W}{2W+d+1} \leq \alpha \leq d+1$ there exists an α-approximate pure Nash equilibrium that, furthermore, has Price of Stability at most*

$$1 + \left(\frac{d+1}{\alpha} - 1\right) W.$$

[11] See [26, Sec. 4.4] for further discussion on this choice.

Observe that, as the approximation parameter α increases, the Price of Stability decreases, in a smooth way, from $\frac{d+3}{2}$ down to the optimal value of 1. Furthermore, notice how the interval within which α ranges, shrinks as the range of player weights W grows; in particular, its left boundary $\frac{2(d+1)W}{2W+d+1}$ goes from $2\frac{d+1}{d+3}$ (for $W = 1$) up to $d + 1$ (for $W \to \infty$).

As a result, Theorem 9 has two interesting corollaries, one for $\alpha = \frac{2(d+1)W}{2W+d+1}$ and one for $W = 1$ (unweighted games):

▶ **Corollary 10.** *At any congestion game with polynomial latencies of degree at most $d \geq 1$ where player weights lie within the range $[1, W]$, there is an $\frac{2(d+1)W}{2W+d+1}$-approximate pure Nash equilibrium with Price of Stability at most $\frac{d+3}{2}$.*

It is interesting to point out here that, in light of Theorem 5, the above result of Corollary 10 is essentially asymptotically tight as far as the Price of Stability is concerned (see the discussion preceding Theorem 5).

▶ **Corollary 11.** *At any* unweighted *congestion game with polynomial latencies of degree at most $d \geq 1$, the Price of Stability of α-approximate equilibria is at most $\frac{d+1}{\alpha}$, for any $2\frac{d+1}{d+3} \leq \alpha \leq d + 1$.*

References

1. Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, 9th printing, 10th gpo printing edition, 1964. URL: `http://people.maths.ox.ac.uk/~macdonald/aands/index.html`.
2. Heiner Ackermann, Heiko Röglin, and Berthold Vöcking. On the impact of combinatorial structure on congestion games. *Journal of the ACM*, 55(6):1–22, dec 2008. `doi:10.1145/1455248.1455249`.
3. Sebastian Aland, Dominic Dumrauf, Martin Gairing, Burkhard Monien, and Florian Schoppmann. Exact price of anarchy for polynomial congestion games. *SIAM Journal on Computing*, 40(5):1211–1233, jan 2011. `doi:10.1137/090748986`.
4. Susanne Albers. On the value of coordination in network design. *SIAM Journal on Computing*, 38(6):2273–2302, 2009.
5. Elliot Anshelevich, Anirban Dasgupta, Jon Kleinberg, Éva Tardos, Tom Wexler, and Tim Roughgarden. The price of stability for network design with fair cost allocation. *SIAM Journal on Computing*, 38(4):1602–1623, jan 2008. `doi:10.1137/070680096`.
6. Baruch Awerbuch, Yossi Azar, and Amir Epstein. The price of routing unsplittable flow. *SIAM Journal on Computing*, 42(1):160–177, jan 2013. `doi:10.1137/070702370`.
7. Kshipra Bhawalkar, Martin Gairing, and Tim Roughgarden. Weighted congestion games: The price of anarchy, universal worst-case examples, and tightness. *ACM Trans. Econ. Comput.*, 2(4):141–1423, oct 2014. `doi:10.1145/2629666`.
8. Vittorio Bilò. A unifying tool for bounding the quality of non-cooperative solutions in weighted congestion games. *Theory of Computing Systems*, 62, dec 2017. `doi:10.1007/s00224-017-9826-1`.
9. Vittorio Bilò, Ioannis Caragiannis, Angelo Fanelli, and Gianpiero Monaco. Improved Lower Bounds on the Price of Stability of Undirected Network Design Games. *Theory of Computing Systems*, 52(4):668–686, may 2013. `doi:10.1007/s00224-012-9411-6`.
10. Vittorio Bilò, Michele Flammini, and Luca Moscardelli. The price of stability for undirected broadcast network design with fair cost allocation is constant. *Games and Economic Behavior*, 2014.

11 Vittorio Bilò and Cosimo Vinci. On the impact of singleton strategies in congestion games. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 17:1–17:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.17.

12 Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Efficient computation of approximate pure Nash equilibria in congestion games. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, oct 2011. doi:10.1109/focs.2011.50.

13 Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Approximate pure Nash equilibria in weighted congestion games: Existence, efficient computation, and structure. *ACM Trans. Econ. Comput.*, 3(1):2:1–2:32, 2015. doi:10.1145/2614687.

14 Ioannis Caragiannis, Michele Flammini, Christos Kaklamanis, Panagiotis Kanellopoulos, and Luca Moscardelli. Tight bounds for selfish and greedy load balancing. *Algorithmica*, 61(3):606–637, 2010. doi:10.1007/s00453-010-9427-8.

15 George Christodoulou and Martin Gairing. Price of stability in polynomial congestion games. *ACM Transactions on Economics and Computation*, 4(2):1–17, dec 2015. doi:10.1145/2841229.

16 George Christodoulou and Elias Koutsoupias. On the price of anarchy and stability of correlated equilibria of linear congestion games. In *Algorithms - ESA 2005, 13th Annual European Symposium*, pages 59–70, 2005.

17 George Christodoulou and Elias Koutsoupias. The price of anarchy of finite congestion games. In *STOC '05: Proceedings of the 37th annual ACM symposium on Theory of computing*, pages 67–73, New York, NY, USA, 2005. ACM. doi:10.1145/1060590.1060600.

18 George Christodoulou, Elias Koutsoupias, and Paul G. Spirakis. On the performance of approximate equilibria in congestion games. *Algorithmica*, 61(1):116–140, 2011. doi:10.1007/s00453-010-9449-2.

19 John H. Conway and Richard K. Guy. *The Book of Numbers*. Springer New York, 1996. doi:10.1007/978-1-4612-4072-3.

20 Juliane Dunkel and Andreas S. Schulz. On the complexity of pure-strategy Nash equilibria in congestion and local-effect games. *Mathematics of Operations Research*, 33(4):851–868, 2008.

21 Alex Fabrikant, Christos Papadimitriou, and Kunal Talwar. The complexity of pure Nash equilibria. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 604–612, New York, NY, USA, 2004. doi:10.1145/1007352.1007445.

22 Matthias Feldotto, Martin Gairing, Grammateia Kotsialou, and Alexander Skopalik. Computing approximate pure Nash equilibria in Shapley value weighted congestion games. In *Web and Internet Economics*, pages 191–204. Springer International Publishing, 2017. doi:10.1007/978-3-319-71924-5_14.

23 Amos Fiat, Haim Kaplan, Meital Levy, Svetlana Olonetsky, and Ronen Shabo. On the price of stability for designing undirected networks with fair cost allocations. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming: 33rd International Colloquium (ICALP)*, pages 608–618, 2006. doi:10.1007/11786986_53.

24 Dimitris Fotakis, Spyros Kontogiannis, Elias Koutsoupias, Marios Mavronicolas, and Paul Spirakis. The structure and complexity of Nash equilibria for a selfish routing game. *Theoretical Computer Science*, 410(36):3305–3326, 2009. doi:10.1016/j.tcs.2008.01.004.

25 Dimitris Fotakis, Spyros Kontogiannis, and Paul Spirakis. Selfish unsplittable flows. *Theoretical Computer Science*, 348(2):226–239, 2005.

26 Yiannis Giannakopoulos, George Christodoulou, Martin Gairing, and Paul Spirakis. The price of stability of weighted congestion games. *CoRR*, abs/1802.09952, 2018. arXiv:arXiv:1802.09952.

27 M. Goemans, Vahab Mirrokni, and A. Vetta. Sink equilibria and convergence. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 142–151, Oct 2005. doi:10.1109/SFCS.2005.68.

28 Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

29 Christoph Hansknecht, Max Klimm, and Alexander Skopalik. Approximate pure Nash equilibria in weighted congestion games. In Klaus Jansen, José D. P. Rolim, Nikhil R. Devanur, and Cristopher Moore, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*, volume 28 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 242–257. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.242.

30 Tobias Harks and Max Klimm. On the existence of pure Nash equilibria in weighted congestion games. *Mathematics of Operations Research*, 37(3):419–436, 2012. doi:10.1287/moor.1120.0543.

31 Tobias Harks, Max Klimm, and Rolf H Möhring. Characterizing the existence of potential functions in weighted congestion games. *Theory Comput Syst*, 49:46–70, 2011. doi:10.1007/s00224-011-9315-x.

32 Tobias Harks, Max Klimm, and Rolf H Möhring. Strong equilibria in games with the lexicographical improvement property. *International Journal of Game Theory*, 42(2):461–482, 2012. doi:10.1007/s00182-012-0322-1.

33 Donald E. Knuth. Johann Faulhaber and sums of powers. *Mathematics of Computation*, 61(203):277–277, sep 1993. doi:10.1090/s0025-5718-1993-1197512-7.

34 Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *16th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '99, pages 404–413, 1999.

35 Euiwoong Lee and Katrina Ligett. Improved bounds on the price of stability in network cost sharing games. In *Proceedings of the 14th ACM Conference on Electronic Commerce*, EC '13, pages 607–620. ACM, 2013. doi:10.1145/2482540.2482562.

36 Lavy Libman and Ariel Orda. Atomic resource sharing in noncooperative networks. *Telecommunication Systems*, 17(4):385–409, Aug 2001. doi:10.1023/A:1016770831869.

37 Dov Monderer and Lloyd S. Shapley. Potential games. *Games and economic behavior*, 14(1):124–143, 1996.

38 Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 2007.

39 Robert W Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973.

40 Robert W Rosenthal. The network equilibrium problem in integers. *Networks*, 3(1):53–59, 1973.

41 Tim Roughgarden. Intrinsic robustness of the price of anarchy. *J. ACM*, 62(5):32:1–32:42, 2015. doi:10.1145/2806883.

42 Tim Roughgarden and Éva Tardos. How bad is selfish routing? *J. ACM*, 49(2):236–259, mar 2002. doi:10.1145/506147.506153.

43 Tim Roughgarden and Éva Tardos. Bounding the inefficiency of equilibria in nonatomic congestion games. *Games and Economic Behavior*, 47(2):389–403, may 2004. doi:10.1016/j.geb.2003.06.004.

44 Andreas S. Schulz and Nicolás Stier Moses. On the performance of user equilibria in traffic networks. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete*

Algorithms, SODA '03, pages 86–87, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

Demand-Independent Optimal Tolls

Riccardo Colini-Baldeschi
Core Data Science Group, Facebook Inc., 1 Rathbone Place, London, W1T 1FB, UK
rickuz@fb.com
https://orcid.org/0000-0001-5739-1178

Max Klimm
School of Business and Economics, HU Berlin, Spandauer Str. 1, 10099 Berlin, Germany
max.klimm@hu-berlin.de
https://orcid.org/0000-0002-9061-2267

Marco Scarsini
Dipartimento di Economia e Finanza, LUISS, Viale Romania 32, 00197 Roma, Italy
marco.scarsini@luiss.it
https://orcid.org/0000-0001-6473-794X

Abstract

Wardrop equilibria in nonatomic congestion games are in general inefficient as they do not induce an optimal flow that minimizes the total travel time. Network tolls are a prominent and popular way to induce an optimum flow in equilibrium. The classical approach to find such tolls is marginal cost pricing which requires the exact knowledge of the demand on the network. In this paper, we investigate under which conditions demand-independent optimum tolls exist that induce the system optimum flow for any travel demand in the network. We give several characterizations for the existence of such tolls both in terms of the cost structure and the network structure of the game. Specifically we show that demand-independent optimum tolls exist if and only if the edge cost functions are shifted monomials as used by the Bureau of Public Roads. Moreover, non-negative demand-independent optimum tolls exist when the network is a directed acyclic multi-graph. Finally, we show that any network with a single origin-destination pair admits demand-independent optimum tolls that, although not necessarily non-negative, satisfy a budget constraint.

2012 ACM Subject Classification Theory of computation → Network games

Keywords and phrases nonatomic congestion games, efficiency of equlibria, tolls

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.151

Related Version A full version of the paper is available at `https://arxiv.org/abs/1708.02737`.

Acknowledgements Riccardo Colini-Baldeschi and Marco Scarsini are members of GNAMPA-INdAM. Max Klimm gratefully acknowledges the support and hospitality of LUISS during a visit in which this research was initiated. The research of Max Klimm was carried out in the framework of MATHEON supported by Einstein Foundation Berlin.

1 Introduction

The impact of selfish behavior on the efficiency of traffic networks is a longstanding topic in the algorithmic game theory and operations research literature. Already more than half a century ago, Wardrop [41] stipulated a main principle of a traffic equilibrium that – in light of the omnipresence of modern route guidance systems – is as relevant as ever: "The

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 151; pp. 151:1–151:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

journey times on all the routes actually used are equal, and less than those which would be experienced by a single vehicle on any unused route." This principle can be formalized by modeling traffic as a flow in a directed network where edges correspond to road segments and vertices correspond to crossings. Each edge is endowed with a cost function that maps the total amount of traffic on it to a congestion cost that each flow particle traversing the edge has to pay. Further, we are given a set of commodities, each specified by an origin, a destination, and a flow demand. In this setting, a Wardrop equilibrium is a multi-commodity flow such that for each commodity the total cost of any used path is not larger than the total cost of any other path linking the commodity's origin and destination. Popular cost functions, put forward for the use in traffic models by the U.S. Bureau of Public Roads [40] are of the form

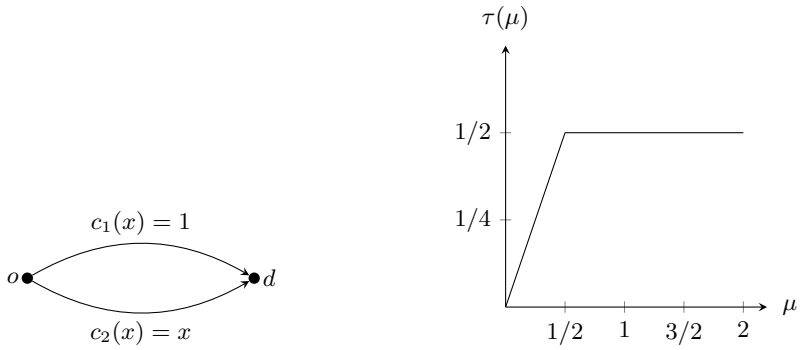
$$c_e(x_e) = t_e\left(1 + \alpha\left(\frac{x_e}{k_e}\right)^{\beta}\right), \tag{1}$$

where x_e is the traffic flow along edge e, $t_e \geq 0$ is the free-flow travel time, k_e is the capacity of edge e and α and β are parameters fitted to the model.

While Wardrop equilibria are guaranteed to exist under mild assumptions on the cost functions [4], it is well known that they are inefficient in the sense that they do not minimize the overall travel time of all commodities [34]. A popular mechanism to decrease the inefficiency of selfish routing are congestion tolls. A toll is a payment that the system designer defines for each edge of the graph and that has to be paid by each flow particle traversing the edge. By carefully choosing the edges' tolls, the system designer can steer the Wardrop equilibrium in a favorable direction. A classic approach first due to Pigou [34] is *marginal cost pricing* where the toll of each edge is equal to difference between the marginal social cost and the marginal private cost of the system optimum flow on that edge. Marginal cost pricing induces the system optimal flow – the one that minimizes the overall travel time – as a Wardrop equilibrium [4]. Congestion pricing is not only an interesting theoretical issue that links system optimal flows and traffic equilibria, but also a highly relevant tool in practice, as various cities of the world, including Stockholm, Singapore, Bergen, and London, implement congestion pricing schemes to mitigate congestion [22, 39].

The problem. Marginal cost pricing is an elegant way to induce the system optimum flow as a Wardrop equilibrium, but the concept crucially relies on the exact knowledge of the travel demand. As an example consider the Pigou network in Figure 1a for an arbitrary flow demand of $\mu > 0$ going from o to d. The optimal flow only uses the lower edge with cost function $c(x) = x$ as long as $\mu \leq 1/2$. For demands $\mu > 1/2$, a flow of $1/2$ is sent along the lower edge and the remaining flow of $\mu - 1/2$ is sent along the upper edge. Using marginal cost pricing the toll of the lower edge is $\min\{\mu, 1/2\}$ and no toll is to be payed for the upper edge, see Figure 1b.

This toll scheme is problematic as it depends on the flow demand μ in the network. In particular, when the demand is estimated incorrectly, the resulting tolls will be sub-optimal. Assume the network designer expects a flow of $\mu = 1/4$ and, thus, sets a (marginal cost) toll of $\tau_2 = 1/4$ on the lower link. When the actual total flow demand is higher than expected and equal to $\mu = 1$, a fraction $1/4$ of the flow uses the upper edge and $3/4$ of the flow uses the lower edge resulting in a cost of $1/4 + 9/16$. However, it is optimal to induce an equal split between the edge which can be achieved by a toll of $1/2$ on the lower edge. Quite strikingly, the toll of $\tau_2 = 1/2$ is optimal for all possible demand values as it always induces the optimal flow. For demands less than $1/2$, a toll of $1/2$ on the lower edge does not hinder any flow particle from using the lower edge, which is optimal. On the other hand, for any demand



(a) Pigou network. **(b)** Marginal cost toll for lower edge.

Figure 1 Dependence of the marginal cost tolls for the Pigou network.

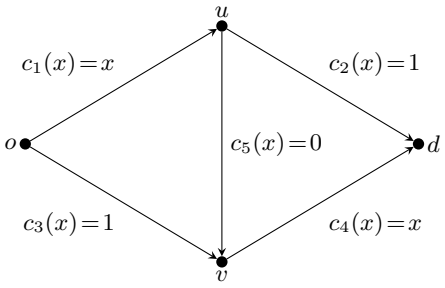


Figure 2 Braess network.

larger 1/2, the toll of 1/2 on the lower edge forces the flow on this edge to not exceed 1/2, which is optimal as well.

The situation is even more severe for the Braess network in Figure 2. When the system designer expects a traffic demand of 1 going from o to d, marginal cost pricing fixes a toll of 1 on both the upper left and the lower right edge (both with cost function $c(x) = x$). When the demand is lower than expected, say, $\mu = 1/2$, under marginal cost pricing, the flow is split equally between the lower and the upper path leading to a total cost of 5/4. The optimal flow with flow value 1/2, however, only uses the zig-zag-path $o \to u \to v \to d$ with cost 1. It is interesting to note that this flow is actually equal to the Wardrop equilibrium without any tolls. To conclude this example, marginal cost pricing may actually *increase* the total cost of the Wardrop equilibrium when the travel demand is estimated incorrectly. We note that also in the Braess graph, there is a distinct toll vector that enforces the optimum flow as a Wardop equilibrium *for any demand*. We will see that by setting a toll of 1 on the central edge from u to v, the Wardrop equilibrium for any flow demand is equal to the respective optimum flow.

We conclude that for both the Pigou network and the Braess network, marginal cost pricing is not robust with respect to changes in the demand since wrong estimates of the travel demand lead to sub-optimal tolls. Since such changes may occur frequently in road networks (e.g., due to sudden weather changes, accidents, or other unforeseen events), marginal cost pricing does not use the full potential of congestion pricing and may even be harmful for the traffic. On the other hand, for both networks, there exist tolls that enforce the optimum flow as an equilibrium for *any* flow demand. In this paper, we systematically study conditions for such demand-independent optimal tolls to exist.

Our results. In this paper we study the existence of demand-independent optimum tolls (DIOTs) that induce the optimum flow as an equilibrium for *any* flow demand. We give a precise characterization in terms of the cost structure on the edges for DIOTs to exists. Specifically, we show that DIOTs exist for any network where the cost of each edge is a BPR-type function, i.e.,

$$c_e(x) = t_e + a_e x_e^\beta \quad \text{for all } e \in \mathcal{E}, \tag{2}$$

where $t_e, a_e \in \mathbb{R}_+$ are arbitrary while $\beta \in \mathbb{R}_+$ is a common constant for all edges $e \in \mathcal{E}$. This existence result holds regardless of the topology of the network and on the number of origin-destination (O/D) pairs. On the other hand, for any cost function that is not of the form as in (2), there is a simple network consisting of two parallel edges with cost function c and cost function $c + b$ for some $b > 0$ that does not admit a DIOT. Our existence result for networks with BPR-type cost functions is proven in terms of a characterization that uniquely determines the sum of the tolls along each path that is used by the optimum flow for some demand.

In general the DIOTs used in the characterization may use negative tolls as well. We provide an example of a network with BPR-type cost functions where a non-negative DIOT does not exist. Besides conditions on the costs, conditions on the network are needed to guarantee the existence of non-negative DIOTs. We show that non-negative DIOTs exist for directed acyclic multi-graphs (DAMGs) with BPR-type cost functions, like the Pigou network and the Braess network discussed in Section 1. This condition on the network is sufficient, but not necessary for the existence of non-negative DIOTs.

Under a weaker condition than DAMG, we prove the existence of DIOTs that follow a budget constraint of non-negativity of the total amount of tolls. This condition is satisfied by networks with a single O/D pair.

Due to space constraint, some of the proofs are omitted and can be found in the arXiv version [15].

Related work. Marginal cost pricing as a means to reduce the inefficiency of selfish resource allocation was first proposed by Pigou [34] and subsequently discussed by Knight [31]. Wardrop [41] introduced the notion of a traffic equilibrium where each flow particle only uses shortest paths. Beckmann et al. [4] showed that marginal cost pricing always induces the system optimal flow as a Wardrop equilibrium. The set of feasible tolls that induce optimal flows was explored in [5, 26, 32]. They showed that the set of optimal tolls can be described by a set of linear equations and inequalities.

This characterization led to various developments regarding the optimization of secondary objectives of the edge tolls, such as the minimization of the tolls collected from the users [1, 16, 17], or the minimzation of the number of edges that have positive tolls [2, 3]. A problem closely related to the latter is to compute tolls for a given subset of edges with the objective to minimize the total travel time of the resulting equilibrium. Hoefer et al. [27] showed that this problem is NP-hard for general networks, and gave an efficient algorithm for parallel edges graphs with affine cost functions. Harks et al. [25] generalized their result to arbitrary cost functions satisfying a technical condition. Bonifaci et al. [8] studied generalizations of this problem with further restrictions on the set of feasible edge tolls.

For heterogenous flow particles that trade off money and time differently, marginal cost pricing cannot be applied to find tolls that induce the system optimum flow. In this setting, Ciole et al. [13] showed the existence of a set of tolls enforcing the system optimal flow, when there is a single commodity in the network. Similarly, Yang and Huang [42] studied how

to design toll structure when there are users with different toll sensitivity. Fleischer [18] showed that in single source series-parallel networks the tolls have to be linear in the latency of the maximum latency path. Karakostas and Kolliopoulos [28] and Fleischer et al. [19] independently generalized this result to arbitrary networks. Karakostas and Kolliopoulos [30] showed similar results for players with elastic demands. Han et al. [24] extended the previous results to different classes of cost functions.

Most of the literature assumes that the charged tolls cause no disutilty to the network users. For the case where tolls contribute to the cost, Cole et al. [14] showed that marginal cost tolls do not improve the equilibrium flow for a large class of instances, including all instances with affine costs. They further showed that for these networks it is NP-hard to approximate the minimal total cost that can be achieved as a Wardrop equilibrium with tolls. Karakostas and Kolliopoulos [29] proved that the total disutility due to taxation is bounded with respect to the social optimum for large classes of latency functions. Moreover, they showed that, if both the tolls and the latency are part of the social cost, then for some latency functions the coordination ratio improves when taxation is used. For networks of parallel edges, Christodoulou et al. [12] studied a generalization of edge tolls where cost functions are allowed to increase in an arbitrary way. They showed that for affine cost functions, the price of anarchy is strictly better than in the original network, even when the demand is not known.

Brown and Marden [9, 10] studied how marginal tolls can create perverse incentives when users have different sensitivity to the tolls and how it possible to circumvent this problem. Caragiannis et al. [11] studied the optimal toll problem for atomic congestion games. They proved that in the atomic case the optimal system performance cannot be achieved even in very simple networks. On the positive side they shown that there is a way to assign tolls to edges such that the induced social cost is within a factor of 2 to the optimal social cost. Singh [38] observed that marginal tolls weakly enforce optimal flows. Fotakis and Spirakis [21] showed that in series-parallel networks with increasing cost functions the optimal social cost can be induced with tolls. Fotakis et al. [20] were the first to consider the problem of defining tolls for heterogeneous users in atomic congestion games with unsplittable flow. In [7] the problem of defining tolls for atomic congestion games with polynomial cost functions was considered for the first time. Meir and Parkes [33] discussed how in atomic congestion games with marginal tolls multiple equilibria are near-optimal when there is a large number of players.

Sandholm [36, 37] studied tolls from a mechanism design perspective where the social planner has no information over the preferences of the users and has limited ability to observer the users' behavior. Bhaskar et al. [6] studied the problem of achieving target flows as equilibria of a nonatomic routing game without knowing the underlying latency functions and showed that a given target flow can be achieved using a polynomial number of queries to an oracle that takes tolls as input and outputs the resulting equilibrium flow. Roth et al. [35] considered the problem of finding optimal tolls in a polynomial number of rounds when latency functions are unknown. They solved the problem by embedding it in a broad class of Stackelberg game.

2 Model and preliminaries

In this section, we present some notation and basic definitions that are used in the sequel. We start with the underlying network model and will then introduce equilibria and tolls.

Network model. We consider a finite directed multi-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. We call $(v \to v')$ the set of all edges e whose tail is v and whose head is v'. We assume that there is a finite set of *origin-destination* (O/D) pairs $i \in \mathcal{I}$, each with an individual *traffic demand* $\mu^i \geq 0$ that has to be routed from an origin $o^i \in \mathcal{V}$ to a destination $d^i \in \mathcal{V}$ via $\mathcal{G}$. Denote the demand vector by $\boldsymbol{\mu} = (\mu^i)_{i\in\mathcal{I}}$. We call $\mathcal{P}^i$ the set of (simple) paths joining o^i to d^i, where each path $p \in \mathcal{P}^i$ is a finite sequence of edges such that the head of each edge meets the tail of the subsequent edge. For as long as all pairs (o^i, d^i) are different, the sets $\mathcal{P}^i$ are disjoint. Call $\mathcal{P} := \bigcup_{i\in\mathcal{I}} \mathcal{P}^i$ the union of all such paths.

Each path p is traversed by a *flow* $f_p \in \mathbb{R}_+$. Call $\boldsymbol{f} = (f_p)_{p\in\mathcal{P}}$ the vector of flows in the network. The set of feasible flows for $\boldsymbol{\mu}$ is defined as

$$\mathcal{F}(\boldsymbol{\mu}) = \Big\{ \boldsymbol{f} \in \mathbb{R}_+^{\mathcal{P}} : \textstyle\sum_{p\in\mathcal{P}^i} f_p = \mu^i \text{ for all } i \in \mathcal{I} \Big\}. \tag{3}$$

In turn, a routing flow $\boldsymbol{f} \in \mathcal{F}(\boldsymbol{\mu})$ induces a *load* on each edge $e \in \mathcal{E}$ as

$$x_e = \sum_{p\ni e} f_p. \tag{4}$$

We call $\boldsymbol{x} = (x_e)_{e\in\mathcal{E}}$ the corresponding *load profile* on the network. For each $e \in \mathcal{E}$ consider a nondecreasing, continuous *cost function* $c_e \colon \mathbb{R}_+ \to \mathbb{R}_+$. Denote $\boldsymbol{c} = (c_e)_{e\in\mathcal{E}}$. If $\boldsymbol{x}$ is the load profile induced by a feasible routing flow $\boldsymbol{f}$, then the incurred delay on edge $e \in \mathcal{E}$ is given by $c_e(x_e)$; hence, with a slight abuse of notation, the associated cost of path $p \in \mathcal{P}$ is given by the expression $c_p(\boldsymbol{f}) \equiv \sum_{e\in p} c_e(x_e)$. We call the tuple $\Gamma = (\mathcal{G}, \mathcal{I}, \boldsymbol{c})$ a (*nonatomic*) *routing game*.

Equilibrium Flows and Optimal Flows. A routing flow $\boldsymbol{f}^*$ is a *Wardrop equilibrium* (WE) of Γ if, for all $i \in \mathcal{I}$, we have:

$$c_p(\boldsymbol{f}^*) \leq c_{p'}(\boldsymbol{f^*}) \quad \text{for all } p, p' \in \mathcal{P}^i \text{ such that } f_p^* > 0.$$

This concept was introduced by Wardrop [41]. Beckmann et al. [4] showed that Wardrop equilibria are the optimal solutions to the convex optimization problem

$$\begin{aligned} \min \ & \sum_{e\in\mathcal{E}} \int_0^{x_e} c_e(s)\,\mathrm{d}s \\ \text{s.t.:} \ & x_e = \sum_{p\ni e} f_p \\ & \boldsymbol{f} \in \mathcal{F}, \end{aligned}$$

and, thus, are guaranteed to exist. A *social optimum* (SO) is a flow that minimizes the overall travel time, i.e., it solves the following total cost minimization problem:

$$\begin{aligned} \min \ & L(\boldsymbol{f}) = \sum_{p\in\mathcal{P}} f_p c_p(\boldsymbol{f}), \\ \text{s.t.:} \ & \boldsymbol{f} \in \mathcal{F}. \end{aligned} \tag{5}$$

As shown in [4], all Wardrop equilibria have the same social cost. We write $\mathrm{Eq}(\Gamma) = L(\boldsymbol{f}^*)$ and $\mathrm{Opt}(\Gamma) = \min_{\boldsymbol{f}\in\mathcal{F}} L(\boldsymbol{f})$, where $\boldsymbol{f}^*$ is a Wardrop equilibrium of Γ. The game's *price of anarchy* (PoA) is then defined as $\mathrm{PoA}(\Gamma) = \mathrm{Eq}(\Gamma)/\,\mathrm{Opt}(\Gamma)$. It is known that Wardrop equilibria need not minimize the social cost, in that case $\mathrm{PoA}(\Gamma) > 1$. For a pair $i \in \mathcal{I}$, we

denote the set of paths that are eventually used in a optimum flow for some demand vector $\boldsymbol{\mu}$ by

$$\tilde{\mathcal{P}}^i = \{p \in \mathcal{P}^i : \tilde{\boldsymbol{f}}_p(\boldsymbol{\mu}) > 0 \text{ for some demand } \boldsymbol{\mu} \text{ and corresponding social optimum } \tilde{\boldsymbol{f}}(\boldsymbol{\mu})\}.$$

Here and in the following we write $\boldsymbol{f}(\boldsymbol{\mu})$ instead of $\boldsymbol{f}$ when we want to indicate the corresponding demand vector $\boldsymbol{\mu}$.

Tolls. We want to explore the possibility of imposing tolls on the edges of the network in such a way that the equilibrium flow of the game with tolls produces a flow that is a solution of the original miminization problem (5). In other words, we want to see whether it is possible to achieve an optimum flow as an equilibrium of a modified game.

We call $\boldsymbol{\tau} = (\tau_e)_{e\in\mathcal{E}} \in \mathbb{R}^{\mathcal{E}}$ a *toll vector*. Note that we allow both for positive and negative tolls. We call c_e^τ the cost of edge e under the toll τ, i.e., $c_e^\tau(x_e) := c_e(x_e) + \tau_e$. Similarly $c_p^\tau(\boldsymbol{f}) := \sum_{e\in p} c_e^\tau(x_e)$. Define $\Gamma^{\boldsymbol{\tau}} := (\mathcal{G}, \mathcal{I}, \boldsymbol{c}^{\boldsymbol{\tau}})$. A toll vector $\boldsymbol{\tau}$ that for each demand vector $\boldsymbol{\mu} \in \mathbb{R}_+^{\mathcal{I}}$ enforces the corresponding system optimum as the equilibrium in $\Gamma^{\boldsymbol{\tau}}$ is called demand-independent optimal toll.

▶ **Definition 1** (Demand-independent optimal toll (DIOT))**.** Let $\Gamma = (\mathcal{G}, \mathcal{I}, \boldsymbol{c})$. A toll vector $\boldsymbol{\tau} \in \mathbb{R}^{\mathcal{E}}$ is called demand-independent optimal toll (DIOT) for Γ if for every demand vector $\boldsymbol{\mu} \in \mathbb{R}_+^{\mathcal{I}}$ every corresponding equilibrium with tolls $\boldsymbol{f}^{\boldsymbol{\tau}}(\boldsymbol{\mu}) \in \mathrm{Eq}(\Gamma^{\boldsymbol{\tau}})$ is optimal for Γ, i.e., $L(\boldsymbol{f}^{\boldsymbol{\tau}}(\boldsymbol{\mu})) = \sum_{p\in\mathcal{P}} f_p^{\boldsymbol{\tau}}(\boldsymbol{\mu}) c_p(\boldsymbol{f}^{\boldsymbol{\tau}}(\boldsymbol{\mu})) \le L(\boldsymbol{f}(\boldsymbol{\mu})) = \sum_{p\in\mathcal{P}} f_p c_p(\boldsymbol{f}(\boldsymbol{\mu}))$ for all $\boldsymbol{f}(\boldsymbol{\mu}) \in \mathcal{F}(\boldsymbol{\mu})$.

In Section 1 we visited two games, the Pigou networks and Braess' paradox, that admit a DIOT. The aim of this paper is to characterize the networks $\Gamma = (\mathcal{G}, \mathcal{I}, \boldsymbol{c})$ for which DIOTs exist.

3 BPR-type cost functions

In this section, we give a complete characterization of the sets of cost functions that admit a DIOT. On the positive side, we will show that any network with BPR-type cost functions admits a DIOT, independently of the number of commodities and the network topology. On the other hand, we show a strong lower bound proving that for any non-BPR cost function there is a single-commodity game on two parallel edges with costs functions c and $c + t$ for some $t \in \mathbb{R}_+$ that does not admit a DIOT. Formally, for $\beta > 0$, let

$$\mathcal{C}_{\mathrm{BPR}}(\beta) = \{c : \mathbb{R}_+ \to \mathbb{R}_+ : c(x) = t_c + a_c x^\beta \text{ for all } x \ge 0,\ a, t \in \mathbb{R}_+\}$$

be the set of BPR-type cost functions with degree β.

The following theorem gives a sufficient condition for games with BPR-type cost functions to admit a DIOT.

▶ **Theorem 2.** *Consider a game $\Gamma = (\mathcal{G}, \mathcal{I}, \boldsymbol{c})$ such that there is $\beta \in \mathbb{R}_+$ with $c_e \in \mathcal{C}_{\mathrm{BPR}}(\beta)$ for all $e \in \mathcal{E}$. Let $\boldsymbol{\tau}$ be a toll vector such that*

$$\sum_{e\in p}\left(\tau_e + \tfrac{\beta}{\beta+1} t_e\right) \le \sum_{e\in p'}\left(\tau_e + \tfrac{\beta}{\beta+1} t_e\right). \tag{6}$$

for all $i \in \mathcal{I}$ and all $p \in \tilde{\mathcal{P}}^i$ and all $p' \in \mathcal{P}^i$. Then, $\boldsymbol{\tau}$ is a DIOT.

Proof. Fix a demand vector $\boldsymbol{\mu} \in \mathbb{R}_+^{\mathcal{I}}$ and a corresponding optimum flow $\tilde{\boldsymbol{f}}(\boldsymbol{\mu})$ in Γ arbitrarily. We denote by $\tilde{x}_e = \sum_{p \in \mathcal{P}: e \in p} \tilde{f}_p(\boldsymbol{\mu})$ the load imposed on edge e by $\tilde{\boldsymbol{f}}(\boldsymbol{\mu})$. The local optimality conditions of $\tilde{\boldsymbol{f}}(\boldsymbol{\mu})$ imply that for all $i \in \mathcal{I}$ and all $p, p' \in \mathcal{P}^i$ with $\tilde{f}_p(\boldsymbol{\mu}) > 0$ we have

$$\sum_{e \in p} c_e(\tilde{x}_e) + c_e'(\tilde{x}_e)\tilde{x}_e \leq \sum_{e \in p'} c_e(\tilde{x}_e) + c_e'(\tilde{x}_e)\tilde{x}_e.$$

This implies that for all $i \in \mathcal{I}$ and all $p, p' \in \mathcal{P}^i$ with $\tilde{f}_p(\boldsymbol{\mu}) > 0$, there exists a non-negative constant $\lambda(p, p') \geq 0$ such that

$$\lambda(p, p') + \sum_{e \in p} \Big((\beta + 1) a_e \tilde{x}_e^\beta + t_e \Big) = \sum_{e \in p'} \Big((\beta + 1) a_e \tilde{x}_e^\beta + t_e \Big). \tag{7}$$

We proceed to show that when the toll vector $\boldsymbol{\tau}$ satisfies (6), then $\tilde{\boldsymbol{f}}(\boldsymbol{\mu})$ is a Wardop equilibrium of Γ^τ. To this end, consider arbitrary $p, p' \in \mathcal{P}^i$ with $\tilde{f}_p(\boldsymbol{\mu}) > 0$. We have

$$\begin{aligned}
\sum_{e \in p} \Big(c_e(\tilde{x}_e) + \tau_e \Big) &= \sum_{e \in p} \Big(a_e \tilde{x}_e^\beta + t_e + \tau_e \Big) \\
&= \sum_{e \in p} \Big((\beta + 1) a_e \tilde{x}_e^\beta + t_e \Big) - \sum_{e \in p} \Big(\beta a_e \tilde{x}_e^\beta - \tau_e \Big) \\
&= \sum_{e \in p'} \Big((\beta + 1) a_e \tilde{x}_e^\beta + t_e \Big) - \sum_{e \in p} \Big(\beta a_e \tilde{x}_e^\beta - \tau_e \Big) - \lambda(p, p'), \\
&= \sum_{e \in p'} \Big(c_e(\tilde{x}_e) + \tau_e \Big) + \sum_{e \in p'} \Big(\beta a_e \tilde{x}_e^\beta - \tau_e \Big) - \sum_{e \in p} \Big(\beta a_e \tilde{x}_e^\beta - \tau_e \Big) - \lambda(p, p'),
\end{aligned}$$

where the third equality comes from (7). By assumption, the toll vector $\boldsymbol{\tau}$ satisfies equation (6) which implies $\sum_{e \in p} \tau_e - \sum_{e \in p'} \tau_e \leq -\frac{\beta}{\beta+1} \left(\sum_{e \in p} t_e - \sum_{e \in p'} t_e \right)$. Therefore,

$$\begin{aligned}
\sum_{e \in p} \Big(c_e(\tilde{x}_e) + \tau_e \Big) &\leq \sum_{e \in p'} \Big(c_e(\tilde{x}_e) + \tau_e + \beta a_e \tilde{x}_e^\beta + \tfrac{\beta}{\beta+1} t_e \Big) - \sum_{e \in p} \Big(\beta a_e \tilde{x}_e^\beta + \tfrac{\beta}{\beta+1} t_e \Big) - \lambda(p, p') \\
&= \sum_{e \in p'} \Big(c_e(\tilde{x}_e) + \tau_e \Big) + \tfrac{\beta}{\beta+1} \lambda(p, p') - \lambda(p, p') \\
&= \sum_{e \in p'} \Big(c_e(\tilde{x}_e) + \tau_e \Big) - \tfrac{1}{\beta+1} \lambda(p, p'),
\end{aligned}$$

where the first equality stems from (7). The statement of the theorem follows from the fact that $\lambda(p, p') \geq 0$. ◀

As an immediate corollary of this theorem, we obtain the existence of DIOTs in arbitrary multi-commodity networks with BPR-type cost functions as long as negative tolls are allowed.

▶ **Corollary 3.** *Consider a game $\Gamma = (\mathcal{G}, \mathcal{I}, \boldsymbol{c})$ with BPR-type cost functions. Then there exists a DIOT for Γ.*

Proof. Setting the toll vector $\hat{\boldsymbol{\tau}} = (\hat{\tau}_e)_{e \in \mathcal{E}}$ as

$$\hat{\tau}_e = -\frac{\beta}{\beta + 1} t_e, \tag{8}$$

obviously satisfies (6), so that the claim follows from Theorem 2. ◀

In the following, we call $\hat{\boldsymbol{\tau}}$ the *trivial DIOT*. The necessary condition of Theorem 2 implies in particular that $\sum_{e\in p}\tau_e + \frac{\beta}{\beta+1}t_e$ must be equal for all paths $p \in \tilde{\mathcal{P}}^i$ and all pairs $i \in \mathcal{I}$. We proceed to show that this is actually a necessary condition for a DIOT.

▶ **Theorem 4.** *Consider a game $\Gamma = (\mathcal{G}, \mathcal{I}, \boldsymbol{c})$ with BPR-type cost functions. If $\boldsymbol{\tau}$ is a DIOT for Γ, then*

$$\sum_{e\in p}\Big(\tau_e + \tfrac{\beta}{\beta+1}t_e\Big) = \sum_{e\in p'}\Big(\tau_e + \tfrac{\beta}{\beta+1}t_e\Big).$$

for all $i \in \mathcal{I}$ and all $p, p' \in \tilde{\mathcal{P}}^i$.

Proof. Let $i \in \mathcal{I}$ and $p^*, p^{**} \in \tilde{\mathcal{P}}^i$ be arbitrary. By definition of $\tilde{\mathcal{P}}^i$ there are demands vectors $\boldsymbol{\mu}, \boldsymbol{\mu}' \in \mathbb{R}_+^{\mathcal{I}}$ such that $\tilde{f}_{p^*}(\boldsymbol{\mu}) > 0$ and $\tilde{f}_{p^{**}}(\boldsymbol{\mu}') > 0$. Before we prove that $\sum_{e\in p^*}(\frac{\beta}{\beta+1}t_e+\tau_e) = \sum_{e\in p^{**}}(\frac{\beta}{\beta+1}t_e + \tau_e)$, we need some observation regarding the continuity of the optimal path flow functions.

Hall [23] showed that the path flow functions of a Wardrop equilibrium are continuous functions in the travel demand. For $\lambda \in [0,1]$, let $\boldsymbol{\mu}(\lambda) = (1-\lambda)\boldsymbol{\mu} + \lambda\boldsymbol{\mu'}$ parametrize the travel demands on the convex combination of $\boldsymbol{\mu}$ and $\boldsymbol{\mu'}$. Then, by Hall's result, there are continuous path flow functions $f_p(\boldsymbol{\mu}(\cdot)) : [0,1] \to \mathbb{R}_+^{\mathcal{E}}$ for all $p \in \mathcal{P}$ such that $\boldsymbol{f}(\boldsymbol{\mu}(\lambda))$ is a Wardrop equilibrium for the travel demand vector $\boldsymbol{\mu}(\lambda)$ for all $\lambda \in [0,1]$. As the system optimal flow $\tilde{\boldsymbol{f}}$ is a Wardrop equilibrium with respect to the marginal cost function (cf. [4]), the same holds for the system optimum flow vector $\tilde{\boldsymbol{f}}$, i.e., there are continuous path flow functions $\tilde{f}_p(\boldsymbol{\mu}(\cdot)) : [0,1] \to \mathbb{R}_+^{\mathcal{E}}$ for all $p \in \mathcal{P}$ such that $\tilde{\boldsymbol{f}}(\boldsymbol{\mu}(\lambda))$ is an optimal flow for the travel demand vector $\boldsymbol{\mu}(\lambda)$ for all $\lambda \in [0,1]$.

For a flow vector $\boldsymbol{f}$, let $\mathcal{S}^i(\boldsymbol{f}) = \{p \in \mathcal{P}^i : f_p > 0\}$ denote the support of $\boldsymbol{f}$ for $i \in \mathcal{I}$. For a possible support set $S \in 2^{\mathcal{P}^i}$, let

$$L^i(S) = \big\{\lambda \in [0,1] : S = \mathcal{S}^i(\tilde{\boldsymbol{f}}(\boldsymbol{\mu}(\lambda))\big\}$$

denote the (possibly empty) set of values of λ for which the optimal flow $\tilde{\boldsymbol{f}}(\boldsymbol{\mu}(\lambda))$ has support S for i.

Consider an arbitrary support set S with $L^i(S) \neq \emptyset$ and an arbitrary $\lambda \in L^i(S)$. Since $\boldsymbol{\tau}$ is a DIOT for Γ, the optimal flow $\tilde{\boldsymbol{f}}(\boldsymbol{\mu}(\lambda))$ is a Wardrop equilibrium with respect to $\boldsymbol{\tau}$, i.e., $\sum_{e\in p}(c_e(\tilde{x}_e(\boldsymbol{\mu}(\lambda))) + \tau_e) = \sum_{e\in p'}(c_e(\tilde{x}_e(\boldsymbol{\mu}(\lambda))) + \tau_e)$ for all $p, p' \in S$ which implies

$$\sum_{e\in p}\left(a_e\tilde{x}_e(\boldsymbol{\mu}(\lambda))^\beta + t_e + \tau_e\right) = \sum_{e\in p'}\left(a_e\tilde{x}_e(\boldsymbol{\mu}(\lambda))^\beta + t_e + \tau_e\right). \tag{9}$$

for all $p, p' \in S$. By the local optimality conditions of $\tilde{\boldsymbol{f}}(\boldsymbol{\mu}(\lambda)))$, we further have

$$\sum_{e\in p} c_e(\tilde{x}_e(\boldsymbol{\mu}(\lambda))) + c_e'(\tilde{x}_e(\boldsymbol{\mu}(\lambda)))\tilde{x}_e(\boldsymbol{\mu}(\lambda)) = \sum_{e\in p'} c_e(\tilde{x}_e(\boldsymbol{\mu}(\lambda))) + c_e'(\tilde{x}_e(\boldsymbol{\mu}(\lambda)))\tilde{x}_e(\boldsymbol{\mu}(\lambda)), \tag{10}$$

which is equivalent to

$$\sum_{e\in p}\Big((\beta+1)a_e\tilde{x}_e(\boldsymbol{\mu}(\lambda))^\beta + t_e\Big) = \sum_{e\in p'}\Big((\beta+1)a_e\tilde{x}_e(\boldsymbol{\mu}(\lambda))^\beta + t_e\Big) \tag{11}$$

for all $p, p' \in S$. Subtracting (11) from $(\beta+1)$ times (9) and dividing by $\beta+1$ we obtain

$$\sum_{e\in p}\left(\tfrac{\beta}{\beta+1}t_e + \tau_e\right) = \sum_{e\in p'}\left(\tfrac{\beta}{\beta+1}t_e + \tau_e\right) \tag{12}$$

for all $p, p' \in S$.

By the continuity of the path flow functions $\tilde{f}_p(\boldsymbol{\mu}(\cdot))$, for each path $p \in \mathcal{P}$ the set $L^i(p) = \bigcup_{S \in 2^{\mathcal{P}^i} : p \in S} L^i(S)$ is open in $[0,1]$, i.e., it is an open set in the relativ topology of $[0,1]$. In addition, we have that neither $\mu_i(0)$ nor $\mu_i(1)$ are zero, so that for all $\lambda \in [0,1]$ there is a path $p \in \mathcal{P}^i$ such that $\tilde{f}_p(\boldsymbol{\mu}(\lambda)) > 0$. We conclude that $\bigcup_{p \in \mathcal{P}} L^i(p) = [0,1]$. Since the sets $L^i(p)$, $p \in \mathcal{P}^i$ are open and cover the compact $[0,1]$, there is a finite set $\{p_1, \dots, p_t\} \subseteq \mathcal{P}^i$ such that $0 \in L^i(p_1)$, $1 \in L^i(p_t)$, $L^i(p_1) \cup \dots \cup L^i(p_t) = [0,1]$ and $L^i(p_j) \cap L^i(p_{j+1}) \neq \emptyset$ for all $j \in \{1, \dots, t-1\}$. The latter condition implies that for $\lambda \in L^i(p_j) \cap L^i(p_{j+1})$ we have $p_j, p_{j+1} \in \mathcal{S}^i(\tilde{\boldsymbol{f}}(\boldsymbol{\mu}(\lambda))$. Equation (12) then implies $\sum_{e \in p_j} (\frac{\beta}{\beta+1} t_e + \tau_e) = \sum_{e \in p_{j+1}} (\frac{\beta}{\beta+1} t_e + \tau_e)$. Iterating this argument shows $\sum_{e \in p_1} (\frac{\beta}{\beta+1} t_e + \tau_e) = \sum_{e \in p_t} (\frac{\beta}{\beta+1} t_e + \tau_e)$. Finally using that $p_1, p^* \in \mathcal{S}^i(\tilde{\boldsymbol{f}}(\boldsymbol{\mu}(0))$ and $p_t, p^{**} \in \mathcal{S}^i(\tilde{\boldsymbol{f}}(\boldsymbol{\mu}(1))$ gives the claimed result. ◀

We proceed to show that the set of BPR-type cost functions is the largest set of cost functions that guarantee the existence of a DIOT, even for single-commodity networks consisting of two parallel edges. The proof can be found in the arXiv version.

▶ **Theorem 5.** *Let c be twice continuously differentiable, strictly semi-convex and strictly increasing, but not of BPR-type. Then there is a congestion game $\Gamma = (\mathcal{G}, \mathcal{I}, \boldsymbol{c})$ with two parallel edges and cost functions $c(x)$ and $c(x) + t$ for some $t \in \mathbb{R}_+$ that does not have a DIOT.*

A similar construction as in the proof of Theorem 5 shows also that a network with two parallel edges with cost functions $c_1 \in \mathcal{C}_{\mathrm{BPR}}(\beta_1)$ and $c_2 \in \mathcal{C}_{\mathrm{BPR}}(\beta_2)$ does not admit a DIOT if $\beta_1 \neq \beta_2$.

4 Nonnegative tolls

The trivial DIOT toll $\hat{\boldsymbol{\tau}}$ is the trivial solution for both the sufficient condition for a DIOT imposed by Theorem 2 and the necessary condition for a DIOT shown in Theorem 4. However, the trivial DIOT is always negative so that the system designer needs to subsidize the traffic in order to enforce the optimum flow. One may wonder whether the conditions imposed by Theorems 2 and 4 admit also a non-negative solution. Our next result shows that, for games played on a *directed acyclic multi-graph* (DAMG), a non-negative DIOT can always be found. The proof can be found in the arXiv version.

▶ **Theorem 6.** *Consider a game $\Gamma = (\mathcal{G}, \mathcal{I}, \boldsymbol{c})$ with BPR-type cost functions where $\mathcal{G}$ is a DAMG. Then there exists a non-negative DIOT for Γ.*

Proof. Given a DAMG there exists a topological sort, namely a linear ordering $\prec$ of its vertices such that, if $v \prec v'$, then there is no path from v' to v in the DAMG. Notice that, in general the topological sort of a DAMG is not unique. Let $|\mathcal{V}| = n$ and call $\boldsymbol{v}^{\prec} = (v_{(1)}, \dots, v_{(n)})$ the vector of ordered vertices. For each edge $e \in (v_{(i)} \to v_{(j)})$, define

$$\delta_e := j - i. \tag{13}$$

Let $\hat{\boldsymbol{\tau}}$ be the trivial DIOT of the game Γ and let

$$\xi := \min_{e \in \mathcal{E}} \frac{\hat{\tau}_e}{\delta_e} \quad \text{and} \quad \chi := \xi_-, \tag{14}$$

where $\xi_- = \max\{-\xi, 0\}$ is the negative part of ξ. Define now

$$\tau_e = \hat{\tau}_e + \delta_e \chi. \tag{15}$$

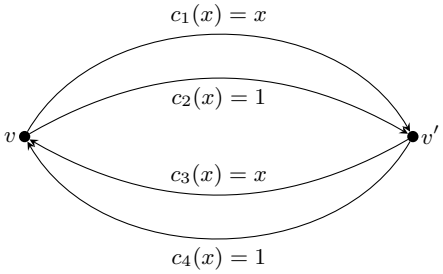


Figure 3 A cyclic network with positive tolls.

We first prove that the toll vector $\boldsymbol{\tau}$ is non-negative. Notice that χ, defined as in (14) is non-negative and $\chi = 0$ only if $\hat{\tau}_e \geq 0$ for all $e \in \mathcal{E}$. Assume that there exists a $\hat{\tau}_e < 0$ and let $e^* \in \arg\min_{e \in \mathcal{E}} \hat{\tau}_e/\delta_e$. Then

$$\tau_{e^*} = \hat{\tau}_{e^*} + \delta_{e^*}\chi = \hat{\tau}_{e^*} - \delta_{e^*}\frac{\hat{\tau_{e^*}}}{\delta_{e^*}} = 0.$$

In general, whenever $\tau_e < 0$, we have

$$\tau_e = \hat{\tau}_e + \delta_e\chi = \hat{\tau}_e - \delta_e\frac{\hat{\tau_{e^*}}}{\delta_{e^*}} \geq \hat{\tau}_e - \delta_e\frac{\hat{\tau}_e}{\delta_e} = 0.$$

Now we prove that the toll vector $\boldsymbol{\tau}$ is a DIOT. By Theorem 2, this means that it satisfies equation (6). First, notice that, by construction of the δ_e, for any $i \in \mathcal{I}$, we have

$$\sum_{e \in p} \delta_e = \sum_{e \in p'} \delta_e \quad \text{for all } p, p' \in \mathcal{P}^i. \tag{16}$$

By (6) we have

$$\sum_{e \in p}\Big((\beta+1)\hat{\tau}_e + \beta t_e\Big) = \sum_{e \in p'}\Big((\beta+1)\hat{\tau}_e + \beta t_e\Big),$$

hence, by (16),

$$\sum_{e \in p}\Big((\beta+1)(\hat{\tau}_e + \delta_e\chi) + \beta t_e\Big) = \sum_{e \in p'}\Big((\beta+1)(\hat{\tau}_e + \delta_e\chi) + \beta t_e\Big),$$

that is

$$\sum_{e \in p}\left((\beta+1)\tau_e + \beta t_e\right) = \sum_{e \in p'}\left((\beta+1)\tau_e + \beta t_e\right),$$

which finishes the proof. ◀

The condition that the graph $\mathcal{G}$ is a DAMG is sufficient for the existence of a non-negative DIOT. It is not necessary, as the following counterexample shows.

▶ **Example 7.** Let $\Gamma = (\mathcal{G}, \mathcal{I}, \boldsymbol{c})$ with $\mathcal{I} = \{1, 2\}$, $\mathcal{V} = \{v, v'\}$, $o_1 = d_2 = v$, $o_2 = d_1 = v'$, $e_1, e_2 \in (o_1 \to d_1)$, $e_3, e_4 \in (o_2 \to d_2)$, and the costs are as in Figure 3. The graph $\mathcal{G}$ is not a DAMG, but the following non-negative toll is a DIOT:

$$\tau_1 = \frac{1}{2} \quad \tau_2 = 0, \quad \tau_3 = \frac{1}{2}, \quad \tau_4 = 0.$$

We proceed to show that for graphs that contain a directed cycle, non-negative DIOTs need not exist, even in networks with affine costs.

▶ **Proposition 8.** *There are networks with affine costs that do not admit a non-negative DIOT.*

5 Aggregatively non-negative Tolls

When nonnegative DIOTs do not exist, it is conceivable that a social planner may sometime want to use negative tolls in order to achieve her goal. Nevertheless, the planner may be subject to budget constraints and not be able to afford a toll system that implies a global loss. Therefore it is interesting to study the existence of conditions for a DIOT $\boldsymbol{\tau}$ such that the following budget constraint is satisfied:

$$\sum_{e\in\mathcal{E}} \tau_e x_e \geq 0, \quad \text{for any feasible flow } \boldsymbol{f}. \tag{17}$$

Intuitively, (17) requires that the social planner does not loose money for any feasible flow. In this section, we show that when the origin-destination pairs (o^i, d^i) satisfy a order condition, then a DIOT satisfying the budget constraint exists.

▶ **Theorem 9.** *Consider a game $\Gamma = (\mathcal{G}, \mathcal{I}, \boldsymbol{c})$ with BPR-type cost functions. If there exists an order $\prec$ on $\mathcal{V}$ such that for all $i \in \mathcal{I}$ we have $o^i \prec d^i$, then there exists a DIOT $\boldsymbol{\tau}$ that satisfies* (17)*.*

We obtain the existence the budget feasible DIOTs for single commodity networks as a direct corollary of Theorem 9.

▶ **Corollary 10.** *Consider a game $\Gamma = (\mathcal{G}, \mathcal{I}, \boldsymbol{c})$ with BPR-type cost and a single O/D pair. Then, there exists a DIOT $\boldsymbol{\tau}$ that satisfies* (17)*.*

Example 7 shows that the condition of Theorem 9 is only sufficient for the existence of a DIOT that satisfies (17).

References

1 Lihud Bai, Donald W. Hearn, and Siriphong Lawphongpanich. Decomposition techniques for the minimum toll revenue problem. *Networks*, 44:142–150, 2004. doi:10.1002/net.20024.

2 Lihud Bai, Donald W. Hearn, and Siriphong Lawphongpanich. A heuristic method for the minimum toll booth problem. *Journal of Global Optimization*, 48:533–548, 2010. doi:10.1007/s10898-010-9527-7.

3 Lihud Bai and Paul A. Rubin. Combinatorial Benders cuts for the minimum tollbooth problem. *Operations Research*, 57:1510–1522, 2009. doi:10.1287/opre.1090.0694.

4 Martin J. Beckmann, C. B. McGuire, and Christopher B. Winsten. *Studies in the Economics of Transportation*. Yale University Press, New Haven, CT, 1956.

5 Pia Bergendorff, Donald W. Hearn, and Motakuri V. Ramana. Congestion toll pricing of traffic networks. In P. Pardalos, D. Hearn, and W. Hager, editors, *Network Optimization*, volume 450 of *LNE*, pages 51–71. Springer, 1997. doi:10.1007/978-3-642-59179-2_4.

6 Umang Bhaskar, Katrina Ligett, Leonard J. Schulman, and Chaitanya Swamy. Achieving target equilibria in network routing games without knowing the latency functions. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 31–40. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.12.

7 Vittorio Bilò and Cosimo Vinci. Dynamic taxes for polynomial congestion games. In *Proceedings of the 17th ACM Conference on Economics and Computation (EC)*, pages 839–856, 2016. doi:10.1145/2940716.2940750.

8 Vincenzo Bonifaci, Mahyar Salek, and Guido Schäfer. Efficiency of restricted tolls in non-atomic network routing games. In G. Persiano, editor, *Proceedings of the 4th Symposium on Algorithmic Game Theory (SAGT)*, volume 6982 of *LNCS*, pages 302–313, 2011. doi:10.1007/978-3-642-24829-0_27.

9 Philip N. Brown and Jason R. Marden. Avoiding perverse incentives in affine congestion games. In *Proceedings of the 55th IEEE Conference on Decision and Control (CDC)*, pages 7010–7015, 2016. doi:10.1109/CDC.2016.7799349.

10 Philip N. Brown and Jason R. Marden. The robustness of marginal-cost taxes in affine congestion games. *IEEE Transactions on Automatic Control*, 62:3999–4004, 2017. doi:10.1109/TAC.2016.2619674.

11 Ioannis Caragiannis, Christos Kaklamanis, and Panagiotis Kanellopoulos. Taxes for linear atomic congestion games. In Y. Azar and T. Erlebach, editors, *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, pages 184–195. Springer, 2006. doi:10.1007/11841036_19.

12 Giorgos Christodoulou, Kurt Mehlhorn, and Evangelia Pyrga. Improving the price of anarchy for selfish routing via coordination mechanisms. *Algorithmica*, 69:619–640, 2014. doi:10.1007/s00453-013-9753-8.

13 Richard Cole, Yevgeniy Dodis, and Tim Roughgarden. Pricing network edges for heterogeneous selfish users. In *Proceedings of the 35th Annual Symposium on Theory of Computing (STOC)*, pages 521–530, 2003. doi:10.1145/780542.780618.

14 Richard Cole, Yevgeniy Dodis, and Tim Roughgarden. How much can taxes help selfish routing? *Journal of Computer and System Sciences*, 72:444–467, 2006. doi:10.1016/j.jcss.2005.09.010.

15 Riccardo Colini-Baldeschi, Max Klimm, and Marco Scarsini. Demand-independent optimal tolls. *arXiv*, 2017. arXiv:1708.02737.

16 Robert B. Dial. Minimal-revenue congestion pricing part I: A fast algorithm for the single-origin case. *Transportation Research, Part B*, 33:189–202, 1999. doi:10.1016/S0191-2615(98)00026-5.

17 Robert B. Dial. Minimal-revenue congestion pricing part II: An efficient algorithm for the general case. *Transportation Research, Part B*, 34:645–665, 2000. doi:10.1016/S0191-2615(99)00046-6.

18 Lisa Fleischer. Linear tolls suffice: New bounds and algorithms for tolls in single source networks. *Theoretical Computer Science*, 348:217–225, 2005. doi:10.1016/j.tcs.2005.09.014.

19 Lisa Fleischer, Kamal Jain, and Mohammad Mahdian. Tolls for heterogeneous selfish users in multicommodity networks and generalized congestion games. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 277–285, 2004. doi:10.1109/FOCS.2004.69.

20 Dimitris Fotakis, George Karakostas, and Stavros G. Kolliopoulos. On the existence of optimal taxes for network congestion games with heterogeneous users. In S. Kontogiannis, E. Koutsoupias, and P. G. Spirakis, editors, *Proceedings of the 3rd Symposium on Algorithmic Game Theory (SAGT)*, volume 6386 of *LNCS*, pages 162–173, 2010. doi:10.1007/978-3-642-16170-4_15.

21 Dimitris Fotakis and Paul G. Spirakis. Cost-balancing tolls for atomic network congestion games. *Internet Mathematics*, 5(4):343–364, 2008. doi:10.1080/15427951.2008.10129175.

22 Jose A. Gómez-Ibáñez and Kenneth A. Small. *Road Pricing for Congestion Management: A Survey of International Practice*, volume 210 of *NCHRP Synthesis*. National Academy Press, Washington, D.C., 1994.

23 Michael A. Hall. Properties of the equilibrium state in transportation networks. *Transportation Science*, 12:208–216, 1978. doi:10.1287/trsc.12.3.208.

24 Deren Han, Hong K. Lo, Jie Sun, and Hai Yang. The toll effect on price of anarchy when costs are nonlinear and asymmetric. *European Journal of Operational Research*, 186:300–316, 2008. doi:10.1016/j.ejor.2007.01.027.

25 Tobias Harks, Ingo Kleinert, Max Klimm, and Rolf H. Möhring. Computing network tolls with support constraints. *Networks*, 65:62–285, 2015. doi:10.1002/net.21604.

26 Donald W. Hearn and Motakuri V. Ramana. Solving congestion toll pricing models. In P. Marcotte and S. Nguyen, editors, *Equilibrium and advanced transportation modeling*, pages 109–124. Springer, New York, 1998. doi:10.1007/978-1-4615-5757-9_6.

27 Martin Hoefer, Lars Olbrich, and Alexander Skopalik. Taxing subnetworks. In C. Papadimitriou and S. Zhang, editors, *Proceedings of the 4th Workshop on Internet and Network Economics (WINE)*, volume 5385 of *LNCS*, pages 286–294, 2008.

28 George Karakostas and Stavros G. Kolliopoulos. Edge pricing of multicommodity networks for heterogeneous selfish users. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 268–276, 2004. doi:10.1109/FOCS.2004.26.

29 George Karakostas and Stavros G. Kolliopoulos. The efficiency of optimal taxes. In A. López-Ortiz and A. Hamel, editors, *Proceedings of the 1st Workshop on Combinatorial and Algorithmic Aspects of Networking (CAAN)*, volume 3405 of *LNCS*, pages 3–12. Springer, 2005. doi:10.1007/11527954_2.

30 George Karakostas and Stavros G. Kolliopoulos. Edge pricing of multicommodity networks for selfish users with elastic demands. *Algorithmica*, 53:225–249, 2009. doi:10.1007/s00453-008-9181-3.

31 Frank H. Knight. Some fallacies in the interpretation of social cost. *Quarterly Journal of Economics*, 38(4):582–606, 1924. doi:10.2307/1884592.

32 Torbjörn Larsson and Michael Patriksson. Side constrained traffic equilibrium models—analysis, computation and applications. *Transportation Research, Part B*, 33(4):233–264, 1999. doi:10.1016/S0191-2615(98)00024-1.

33 Reshef Meir and David Parkes. When are marginal congestion tolls optimal? In *Proceedings of the 9th Workshop on Agents in Traffic and Transportation (ATT)*, volume 1678 of *CEUR Workshop Proceedings*, 2016.

34 Arthur C. Pigou. *The Economics of Welfare*. Macmillan and Co., London, 1st edition, 1920.

35 Aaron Roth, Jonathan Ullman, and Zhiwei Steven Wu. Watch and learn: optimizing from revealed preferences feedback. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 949–962. ACM, 2016. doi:10.1145/2897518.2897579.

36 William H. Sandholm. Evolutionary implementation and congestion pricing. *Review of Economic Studies*, 69(3):667–689, 2002. doi:10.1111/1467-937X.t01-1-00026.

37 William H. Sandholm. Pigouvian pricing and stochastic evolutionary implementation. *Journal of Economic Theory*, 132(1):367–382, 2007. doi:10.1016/j.jet.2005.09.005.

38 Chandramani Singh. Marginal cost pricing for atomic network congestion games. Technical report, Department of Electrical Communication Engineering, Indian Institute of Science, 2008.

39 Kenneth A. Small and Jose A. Gómez-Ibáñez. Road pricing for congestion management: the transition from theory to policy. In Tae Hoon Oum, editor, *Transport Economics*, chapter 16, pages 373–403. Routledge, London, 1997.

40 U.S. Bureau of Public Roads. Traffic assignment manual. U.S. Department of Commerce, Urban Planning Division, 1964.

41 John Glen Wardrop. Some theoretical aspects of road traffic research. In *Proceedings of the Institute of Civil Engineers, Part II*, volume 1, pages 325–378, 1952.

42 Hai Yang and Hai-Jun Huang. The multi-class, multi-criteria traffic network equilibrium and systems optimum problem. *Transportation Research, Part B*, 38:1–15, 2004. doi:10.1016/S0191-2615(02)00074-7.

Greedy Algorithms for Online Survivable Network Design

Sina Dehghani
University of Maryland, College Park, MD 20742, USA

Soheil Ehsani
University of Maryland, College Park, MD 20742, USA

MohammadTaghi Hajiaghayi
University of Maryland, College Park, MD 20742, USA

Vahid Liaghat
Facebook, Building 25, 190 Jefferson Dr, Menlo Park, CA 94025, USA

Saeed Seddighin
University of Maryland, College Park, MD 20742, USA

Abstract

In an instance of the *network design* problem, we are given a graph $G = (V, E)$, an edge-cost function $c : E \to \mathbb{R}^{\geq 0}$, and a connectivity criterion. The goal is to find a minimum-cost subgraph H of G that meets the connectivity requirements. An important family of this class is the *survivable network design problem* (SNDP): given non-negative integers r_{uv} for each pair $u, v \in V$, the solution subgraph H should contain r_{uv} edge-disjoint paths for each pair u and v.

While this problem is known to admit good approximation algorithms in the offline case, the problem is much harder in the online setting. Gupta, Krishnaswamy, and Ravi [14] (STOC'09) are the first to consider the online survivable network design problem. They demonstrate an algorithm with competitive ratio of $O(k \log^3 n)$, where $k = \max_{u,v} r_{uv}$. Note that the competitive ratio of the algorithm by Gupta *et al.* grows linearly in k. Since then, an important open problem in the online community [22, 14] is whether the linear dependence on k can be reduced to a logarithmic dependency.

Consider an online greedy algorithm that connects every demand by adding a minimum cost set of edges to H. Surprisingly, we show that this greedy algorithm significantly improves the competitive ratio when a congestion of 2 is allowed on the edges or when the model is stochastic. While our algorithm is fairly simple, our analysis requires a deep understanding of k-connected graphs. In particular, we prove that the greedy algorithm is $O(\log^2 n \log k)$-competitive if one satisfies every demand between u and v by $r_{uv}/2$ edge-disjoint paths. The spirit of our result is similar to the work of Chuzhoy and Li [7] (FOCS'12), in which the authors give a polylogarithmic approximation algorithm for edge-disjoint paths with congestion 2.

Moreover, we study the greedy algorithm in the online stochastic setting. We consider the i.i.d. model, where each online demand is drawn from a single probability distribution, the unknown i.i.d. model, where every demand is drawn from a single but unknown probability distribution, and the *prophet* model in which online demands are drawn from (possibly) different probability distributions. Through a different analysis, we prove that a similar greedy algorithm is constant competitive for the i.i.d. and the prophet models. Also, the greedy algorithm is $O(\log n)$-competitive for the unknown i.i.d. model, which is almost tight due to the lower bound of [9] for single connectivity.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases survivable network design, online, greedy

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.152

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 152; pp. 152:1–152:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding Supported in part by NSF CAREER award CCF-1053605, NSF BIGDATA grant IIS-1546108, NSF AF:Medium grant CCF-1161365, DARPA GRAPHS/AFOSR grant FA9550-12-1-0423, and another DARPA SIMPLEX grant. Portions of this research were completed while the authors were visitors at the Simons Institute for the Theory of Computing.

Acknowledgements The authors would like to thank Seffi Naor for several fruitful discussions about the prophet versions of the online problems.

1 Introduction

In an instance of the *network design* problem, we are given a graph $G = (V, E)$, an edge-cost function $c : E \to \mathbb{R}^{\geq 0}$, and a connectivity criteria. The goal is to find a minimum-cost subgraph H of G that satisfies the connectivity requirements. An important family of this class is the *survivable network design problem* (SNDP): Given non-negative integers r_{uv} for each pair $u, v \in V$, the solution subgraph H should contain r_{uv} edge-disjoint paths for each pair u and v. SNDP arises in fault tolerance management and thus is of much interest in design community: the connectivity of nodes u and v in H is resilient to even $(r_{uv} - 1)$ edge failures. This problem clearly generalizes the *Steiner tree*[1] and *Steiner forest*[2] problems.

For a non-empty cut $S \subset V$, let $\delta(S)$ denote the set of edges with exactly one endpoint in S. SNDP falls in the general class of network design problems that can be characterized by *proper cut functions*. A function $f : 2^V \to \mathbb{Z}^{\geq 0}$ defined over cuts in the graph is *proper*, if it is symmetric ($f(S) = f(V \setminus S)$ for all $S \subset V$) and it satisfies maximality ($f(S \cup T) \leq \max\{f(S), f(T)\}$ for all $S \cap T = \phi$). For SNDP, one can choose $f(S) = \max_{u \in S, v \notin S} r_{uv}$ for every cut S. Given a proper function f over cuts in the graph, the goal is to find a minimum-cost subgraph H such that

$$|E(H) \cap \delta(S)| \geq f(S) \qquad \forall \textit{non-empty } S \subset V \ .$$

Over the past decades, the offline SNDP and proper cut functions have been extensively studied especially as an important testbed for primal-dual and iterative rounding methods (see e.g. [10, 11, 13, 18, 26, 27]). In this paper, we consider SNDP in the *online* setting: we are given a graph $G = (V, E)$ and an edge-cost function c in advance. We receive an online sequence of demands in the form of tuples $(u, v, r_{uv}) \in V \times V \times \mathbb{Z}^{\geq 0}$. We start with an empty subgraph H. Upon the arrival of a demand (u, v, r_{uv}), we need to immediately *augment* H such that there exist at least r_{uv} edge-disjoint paths between u and v in H. The goal is to minimize the cost of H. The *competitive ratio* of an algorithm is defined as the maximum ratio of the cost of its output and that of an optimal offline solution, over all possible input instances.

The online 1-connectivity problems, in which $r_{uv} \in \{0, 1\}$ for all pairs, have been extensively studied in the last decades. Imase and Waxman [17] (SIAM'91) were first to consider the edge-weighted Steiner tree problem. They used a dual-fitting argument to show that the natural greedy algorithm is $O(\log n)$-competitive where n denotes the number of vertices[3]. Their result is asymptotically tight. Later, Berman and Coulston [3] (STOC'97) and Awerbuch, Azar, and Bartal [2] (TCS'04) demonstrated an $O(\log n)$-competitive algorithm for

[1] In the Steiner tree problem, given a set of terminal nodes $T \subset V$, the goal is to find a minimum-cost subgraph connecting all terminals.

[2] In the Steiner forest problem, given a set of pairs of vertices $s_i, t_i \in V$, the goal is to find a minimum-cost subgraph in which every pair is connected.

[3] In fact, the competitive ratio is $O(\log \min\{n, \mathcal{D}\})$ where $\mathcal{D}$ is the number of demand requests. However,

the more general Steiner forest problem by designing an elegant online primal-dual technique. The latter also shows that the greedy algorithm achieves the competitive ratio of $O(\log^2 n)$ for Steiner forest. Indeed, due to the simplicity of greedy approaches, an important open problem is to settle the competitiveness of the greedy algorithm for Steiner forest. In the recent years, several primal-dual techniques are developed for solving node-weighted variants [1, 15, 22], and prize-collecting variants [25, 16] of 1-connectivity problems.

Gupta, Krishnaswamy, and Ravi [14] (SIAM'12) were first to consider the online survivable network design problem. They demonstrate an elegant algorithm with competitive ratio of $\tilde{O}(k \log^3 n)$, where $k = \max_{u,v} r_{uv}$. The crux of their analysis is to use distance-preserving tree-embeddings in an online setting. More precisely, they first pick a random distance-preserving spanning subtree $T \subseteq G$. They satisfy a connectivity demand r_{uv} by iteratively increasing the connectivity of u and v. In each iteration, they show that it is sufficient to use cycles that are formed by an edge $e = (a, b) \notin T$ and the $\{a, b\}$-path in T; hence, reducing the number of *options* for satisfying a connectivity demand. This would enable them to use a set cover approach to solve the problem in an online manner and achieve the first competitive algorithm for online SNDP.

Single-source SNDP is a variant of SNDP where all demands share the same endpoint. Naor, Panigrahi, and Singh [22] (FOCS'11) partially improve the results of Gupta et al. [14] by demonstrating a *bi-criteria* competitive algorithm for single-source SNDP using structural properties of a single-source optimal solution. A bi-criteria competitive ratio of (α, β) for SNDP implies that the solution produced by the online algorithm achieves a connectivity of $\lfloor \frac{r_i}{\beta} \rfloor$ for every demand σ_i and is at most a factor of α more expensive than the optimal offline solution for connectivity of r_i. The algorithm by Naor et al. achieves the competitive ratio of $(O(\frac{k \log n}{\epsilon}), 2 + \epsilon)$ for any $\epsilon > 0$. They also study and give bi-criteria algorithms for the vertex-connectivity problem.

The competitive ratio of algorithms by Gupta et al. and Naor et al. grow linearly in k. This seems to be inherent to their methods since they may need to solve a set-cover-like problem in each iteration of incrementing the connectivity of a demand; hence, losing a polylogarithmic factor in each iteration. One would need a new approach to break the linear dependency on k. Indeed, both factors of $O(\log^3 n)$ and $O(k)$ are not plausible in practice, and an important open problem in the online community [22, 14] is whether the linear dependency on k can be reduced to a logarithmic dependency.

We circumvent this problem within the class of greedy algorithms. This class has been a center of attention in many applications due to its simplicity. We would like to note that the previous algorithms for the problem are based on fairly complex techniques such tree-embedding and reductions to online set cover. Despite their theoretical provable guarantees, these approaches are not efficiently implementable for large networks and become extremely harder to analyze when more parameters of the system are involved, e.g. degree constraints. For these reasons, we are interested in the theory behind simple algorithms for the online SNDP. In this paper, we study both the cases with and without the presence of priori information about connectivity demands.

No Information Setting: for this traditional online setting, we show that the greedy approach is promising although the *classic greedy algorithm*[4] fails to give a competitive

to simplify the comparison with results for SNDP, in this paper we ignore the distinction between this factor and $O(\log n)$.

[4] Which augments the solution with the cheapest set of edges that satisfy full connectivity demands.

solution. In Section 2, we give a hardness instance for the classic greedy algorithm and present our modified versions of it which have polylogarithmic competitive ratio. In particular, we demonstrate a deterministic algorithm with a bi-criteria competitive ratio of $(O(\log^2 n \log_{1+\epsilon} k), 2+\epsilon)$ for any constant $\epsilon > 0$. For the single-rooted variant, the competitive ratio is $(O(\log n \log_{1+\epsilon} k), 2+\epsilon)$. Besides, our hardness instance shows a loss of $\Omega(n)$ on the weight criterion if satisfying full connectivities greedily.

Partial Information Setting: one of the recent trends in the study of online problems is to consider a stochastic model for the online demands. This, in particular, is to model the scenarios in which the algorithm designer has sufficient data available in hand to be able to make predictions, i.e. fit distributions, for future requests. In this paper, we consider an online stochastic model inspired by the well-known *prophet inequality* problem[5] in which different demands are drawn independently from different distributions. We call it the *prophet setting* not only to highlight the similarity between this online stochastic model and prophet inequalities but also to distinguish it from other online stochastic models[6].

In the second half of this paper, we study online SNDP in the prophet setting for both cases of known and unknown distributions. For the most general case in prophet setting in which the distributions are known and (possibly) different, we show that the classic greedy algorithm that satisfies full connectivity results in a constant approximation solution. Furthermore, we explore the connection between the *known i.i.d.* case and the general case and present a framework that transforms an algorithm for the former into one for the latter by loosing only a constant factor on the approximation ratio. Lastly but more interestingly, we prove that the classic greedy algorithm is $O(\log n)$ approximation[7] for the *unknown i.i.d.* case[8]. This result shows that a simple algorithm can significantly outperform the previous complex algorithms when the demands are drawn from the same distribution without even learning the underlying distribution!

1.1 Our Results and Techniques

Let $\sigma_i = (u_i, v_i, r_i)$ denote the i-th connectivity demand. Consider the following intuitive greedy approach. Upon the arrival of σ_i, we augment the solution subgraph H, by finding the minimum-cost set of edges whose addition to H creates r_i edge-disjoint paths between u_i and v_i. Awerbuch et al. [2] (TCS'04) show that if all the demands require 1-connectivity (i.e., $r_i = 1$ for every i), this algorithm achieves a competitive ratio of $O(\log^2 n)$. This leads to a natural question that whether greedy works for higher connectivity problems as well. However, we show an instance of online SND in Section 2, for which the greedy algorithm has a competitive ratio of $\Omega(n)$. Indeed, the connectivity demands in the instance are either zero or two, hence greedy is not competitive even for low connectivity demands. However, on the positive side, we show that *greedy-like* algorithms do surprisingly well in both the stochastic version of the problem and the case when a small congestion on the edges is acceptable.

---

[5] Given n distributions $D_1, \ldots, D_n$ on real numbers and an online sequence of random draws $X_i \sim D_i$, we have to make an immediate and irrevocable selection X_τ that maximizes the ratio $\mathbb{E}[X_\tau]/\mathbb{E}[\max_{1 \le i \le n} X_i]$.

[6] Such as "the (two stage) stochastic version" of Gupta et al [14].

[7] Which is almost tight

[8] When unknown, it is natural to assume the distributions are i.i.d. Otherwise, the algorithm can be easily tricked.

1.1.1 Allowing Small Congestion

We show that a greedy algorithm does surprisingly well, if we relax the connectivity requirements by a constant factor. Let α denote an arbitrary scale factor. We define an *α-scaled* variant of the greedy algorithm in which the goal is to find only $\lfloor \frac{r_i}{\alpha} \rfloor$ disjoint paths between the endpoints of σ_i. Our main result states that the scaled greedy algorithm is polylogarithmic competitive.

▶ **Theorem 1.** *For any constant $\epsilon > 0$, the $(2+\epsilon)$-scaled greedy algorithm is $(O(\log^2 n \log_{1+\epsilon} k), 2+\epsilon)$-competitive. For the single-source variant, the competitive ratio is $(O(\log n \log_{1+\epsilon} k), 2+\epsilon)$.*
Furthermore, for uniform SNDP, 2-scaled greedy is $(O(\log^2 n), 2)$-competitive.

We start by demonstrating a deep connection between the greedy method for SNDP and the *Steiner packing problems*. The Steiner packing problems are motivated by vast applications in VLSI-layout and have been used as an algorithmic toolkit in computer science. In the *Steiner tree packing* problem, we are given a graph $G = (V, E)$ and a set S of vertices and the goal is to find the *Steiner decomposition number* (SDN), the maximum number of edge-disjoint subgraphs that each connects the vertices of S. We note that a minimal connecting subgraph is a *Steiner tree* with respect to S. In the *Steiner forest packing* problem, we are given a set of demand pairs $u_i, v_i \in V$ and the goal is to find SDN, the maximum number of edge-disjoint subgraphs that in each the demand pairs are connected.

For simplicity, let us assume we have a *uniform* instance. In a uniform instance, we are given an integer k in advance and for any demand (u, v) we must k-connect u to v. Let opt denote the optimal SNDP solution, with the Steiner decomposition number q. In Section 2, we show that the $(\frac{k}{q})$-scaled greedy algorithm approximates opt up to logarithmic factors. Intuitively, every forest in the Steiner forest decomposition, gives us a path to satisfy a demand. Hence, we need to bound the overall cost of satisfying demands in all the q forests. The crux of our analysis is then to charge the cost of the scaled greedy to that of a parallel set of greedy algorithms that solve 1-connectivity instances on every forest. Finally, to get a polylogarithmic competitive algorithm, we need to find a universal lower bound on the SDN number q with respect to k.

It is shown that finding SDN is NP-hard and cannot be computed in polynomial time unless P=NP[6] (Algorithmica'06). Given that there exist q disjoint Steiner forests connecting a set of demands, it is straight forward to show the graph is q-connected on the demands. Therefore, a natural upper bound on SDN is the minimum connectivity of the endpoints of demands. For the case of spanning trees (Steiner tree with $S = V(G)$), it is proven that the above upper bound also provides a good approximation guarantee for the problem. In other words, any k-connected graph can be decomposed into $k/2$ edge-disjoint spanning trees[23]. This is also followed by a matching upper bound. The problem is much subtler when S does not encompass all vertices of the graph. The first lower bound for the Steiner tree packing problem was achieved by Petingi and Rodriguez[24] (CON'03) who proved every S-k-connected[9] has $\lfloor (2/3)^{(|V(G)-|S|)} k/2 \rfloor$ disjoint Steiner trees. This was later improved by Kriesell [20] (JCT'03), Jain, Mahdian, and Salavatipour [19] (SODA'03), Lau [21] (FOCS'04), and DeVos, McDonald, and Pivotto [8] (Man'13), the most recent of which shows for every S-$(5k+4)$-connected graph, we can find k edge-disjoint Steiner trees. However, the main conjecture is that, similar to the case of spanning trees, every S-k-connected graph admits a $k/2$-disjoint Steiner tree decomposition [20].

[9] A graph which is k-connected on a set of vertices S

For a set of demand pairs (u_i, v_i)'s, let $\mathcal{T}$ denote the set of Steiner forests that satisfy all the demands. In the *fractional Steiner forest packing* problem, the output is a fractional assignment x over $\mathcal{T}$ such that for every edge e, $\sum_{T\in\mathcal{T}:e\in T} x_T$ is not more than one. The goal is to find a fractional Steiner forest decomposition with the maximum total sum of weights in x. While the term fractional Steiner forest packing is not explicitly used in the previous work, it follows from the arguments of [4, 8] that the conjecture of Kriesell holds for the fractional variant.

▶ **Theorem 2** (proven in [4]). *Given a set of demand pairs (u_i, v_i), if G is k-connected for every demand pair, then the fractional Steiner decomposition number is at least $k/2$.*

Indeed, in Section 2, we use a dependent rounding method to show that the connection between SDN and the competitiveness of the greedy approach holds even for the stronger fractional variant of SDN. Hence, Theorem 2 implies that the 2-scaled greedy algorithm, achieves a polylogarithmic competitive ratio for the uniform SNDP. Finally, we prove that the scaled greedy is also competitive for the non-uniform variant if one is willing to lose an extra $O(\log k)$ factor in the competitive ratio (Theorem 1).

1.1.2 Online Stochastic SNDP

A single-source *uniform* instance of online SNDP is an instance in which for every demand σ_i, $u_i = u$, $r_i = k$ for some vertex $u \in V$ and integer k. For a non-uniform variant, let $k = \max_i r_i$. Let D be a given probability distribution over V. In i.i.d. SNDP, at each online step i, a random connectivity demand $\sigma_i = (u, v_i, k)$ arrives, where v_i is drawn independently at random from distribution D. We call the problem unknown distribution SNDP if the probability distribution D is not given in advance. Another interesting generalization of the i.i.d. model, which we call the prophet SNDP is defined as follows. In prophet SNDP, instead of only a single probability distribution D, we are given T probability distributions $D_1, \ldots, D_T$, such that the i-th demand is $\sigma_i = (u, v_i, k)$, where v_i is drawn independently at random from distribution D_i. In all three variants of the stochastic SNDP, the competitive-ratio is defined as the expected cost of an algorithm $\mathcal{A}$ over the expected cost of an optimal offline algorithm while the distributions are chosen by an adversary. More precisely let $E[A(\omega)]$ and $E[opt(\omega)]$ denote the expected cost of an algorithm $\mathcal{A}$ and the expected cost of an optimal offline algorithm for an online scenario ω, respectively. Thus the competitive-ratio of algorithm $\mathcal{A}$ is defined as follows.

$$cr(\mathcal{A}) := \max_{D} \frac{E_{\omega\sim D}[\mathcal{A}(\omega)]}{E_{\omega\sim D}[opt(\omega)]}.$$

We first provide an oblivious[10] greedy algorithm for the i.i.d. SNDP. This algorithm starts with a sampling of the demands and finding a 2-approximation solution for them using the algorithm of Jain [18]. Let us call this the backbone solution. Then, to satisfy each demand v, we simply connect it to the backbone using the cheapest set of edges and show that this greedy approach leads to a 4-competitive solution. The oblivious algorithm for the i.i.d. case is in fact a stepping stone that enables us to further analyze the greedy algorithm in the unknown distribution and different known distributions (prophet) cases.

▶ **Theorem 3.** *The oblivious greedy is 4-competitive algorithm for i.i.d. SNDP.*

[10] An oblivious algorithm connects a demand through a path which is independent of the rest of the demands

A similar result to Theorem 3 is also given in [5].

1.1.2.1 Unknown i.i.d.

Although computing the backbone solution is impossible for the unknown i.i.d. case, we take advantage of our analysis of the oblivious i.i.d. algorithm and show that the classic greedy algorithm is $O(\log n)$-approximation for unknown i.i.d. SNDP. The main idea is simple but tricky: due to the sampling nature of the backbone solution, for every $1 \leq k \leq T/2$ we can think of a solution for the first k demands as a backbone solution for the next k demands. Hence, we can exploit our analysis for the known i.i.d. case. This in conjunction with the submodularity of Steiner networks results in the desired competitive ratio. We note that the factor $O(\log n)$ is almost tight given the $\Omega(\frac{\log n}{\log \log n})$ lower bound of [9] for the 1-connectivity case.

▶ **Theorem 4.** *The classic greedy algorithm is $O(\log n)$-competitive for unknown i.i.d. SNDP.*

1.1.2.2 From Oblivious i.i.d. to Prophet

We show if there exists a competitive oblivious algorithm for an online problem in i.i.d. setting, we can obtain a competitive algorithm for the same problem in prophet setting. Roughly speaking, we show that we can combine different distributions in the prophet setting to obtain a single average distribution. Therefore, the i.i.d. oblivious algorithm for the average distribution does not incur more than a constant factor to the competitive ratio.

▶ **Theorem 5** (restated informally). *Given an oblivious α-competitive online algorithm for problem $\mathcal{P}$ in the i.i.d. setting, there exists an $\alpha\frac{2e}{e-1}(1+o(1))$-competitive online algorithm for $\mathcal{P}$ in prophet setting.*

▶ **Corollary 6.** *There exists a constant competitive algorithm for prophet SNDP.*

Using our framework, we can obtain competitive algorithms for many fundamental and classical problems in prophet setting. For instance, define $D_1, \ldots, D_T$ be T probability distributions over the elements of a set cover instance. Now let i-th demand of a set cover problem be an element randomly and independently drawn from distribution D_i. We call this problem the prophet set cover problem. Similarly one may define prophet facility location the same as the classical facility location problem, with the difference that the i-th demand is randomly drawn from a known distribution D_i. Garg et al. [9] provide oblivious online algorithms for i.i.d. facility location and i.i.d. vertex cover. Applying the above framework directly results in the following corollary.

▶ **Corollary 7.** *There exist constant competitive algorithms for prophet vertex cover and prophet facility location problems.*

Also, Grandoni et al. [12] provide an oblivious online algorithm for i.i.d. set cover. Hence, we have the following corollary.

▶ **Corollary 8.** *There exists an $O(\log n)$-competitive algorithm for prophet set cover.*

1.2 Further Related Work

Over the past decades, SNDP and proper cut functions have been an important testbed for primal-dual and iterative rounding methods. Goemans and Williamson [11] (SIAM'95) were first to consider the case of $\{0,1\}$-proper functions. They used a primal-dual method to obtain a 2-approximation algorithm for the problem; which later on got generalized to the celebrated moat-growing framework for solving connectivity problems. Klein and Ravi [26] (IPCO'93) considered the two-connectivity problem and the case of $\{0,2\}$-proper functions. They gave a primal-dual 3-approximation algorithm for the problem. Williamson, Goemans, Mihail, and Vazirani [27] (Combinatorica'95) were first to consider general proper functions. They too developed a primal-dual algorithm with approximation ratio $2k$, where $k = \max_S f(S)$. Subsequently, Goemans, Goldberg, Plotkin, Shmoys, Tardos, and Williamson [10] (SODA'94) presented a primal-dual $2H(k)$-approximation algorithm, where $H(k)$ is the k^{th} harmonic number. Finally, in his seminal work [18] (Combinatorica'01), Jain introduced the *iterative rounding* method by developing a 2-approximation algorithm for network design problems characterized by proper cut functions[11]. We refer the reader to [13] for a survey of results for (offline) network design problems.

2 Uniform SNDP

In this section we consider the uniform-connectivity version of the online survivable Steiner network design problem, in which all connectivity requirements are equal to a given number. For this problem we first give a very simple algorithm and then analyze it using Steiner packing tools. Further in the paper, we explain how to generalize our algorithm to make it work for inputs with non-uniform connectivity requirements.

In the online uniform-connectivity survivable Steiner forest problem we are given an offline graph $G = \langle V(G), E(G)\rangle$, an integer k, and an online stream of demands $S = (s_1,t_1),(s_2,t_2),\ldots$. Every time a demand (s_i,t_i) arrives we have to add some of the edges of G to our current solution H in order to make k edge-disjoint paths between s_i and t_i in H. The online uniform-connectivity survivable Steiner tree problem is a special case of the forest problem in which the second endpoints of all demands are fixed at some vertex *root*. The objective of the problems is to minimize the cost of the selected subgraph H according to a given cost function.

A simple approach to solve these problems is to choose edges based on the following greedy method: for every demand add a minimum-cost subset of edges that satisfies the k-connectivity between its endpoints. In this section we show that this algorithm is not competitive to the optimum offline solution. This is shown by Lemma 14 in which we give an instance graph and a series of demands for which the greedy algorithm gives a solution of cost $\Omega(n)$ times the cost of the optimum offline solution.

However, we show a modified version of the greedy algorithm can be a viable approach for these problems if we lose some factor on the connectivity requirement. This can be done by satisfying half of the required connectivity. In particular, for every demand we add a minimum-cost subset of the edges that makes the current solution $(k/2)$-connected between the endpoints of that demand. Let us call this algorithm *GA*. In this section we show the cost of the edges GA selects is poly-logarithmically competitive to the optimum offline solution that satisfies k-connectivity for every demand.

[11] Indeed, the results in [10] and [18] applies to the more general class of weakly or skew supermodular cut functions.

Algorithm 1: 2-scaled Greedy

Input: A graph G, an integer k, and an online stream of demands $(s_1, t_1), (s_2, t_2), \ldots$.

Output: A set H of edges such that every given demand (s_i, t_i) is connected through k edge-disjoint paths in H.

Offline Process:

1: Initialize $H = \varnothing$.

Online Scheme; assuming a demand (s_i, t_i) is arrived:

1: $P_i =$ A minimum-cost subset of edges, such that s_i is $k/2$-connected to t_i in $H \cup P_i$.

2: Update $H = H \cup P_i$.

▶ **Theorem 9.** *For the online survivable Steiner forest problem, the output of GA satisfies $(k/2)$-connectivity for every demand and its cost is $O(\log^2 n)$-competitive.*

▶ **Theorem 10.** *For the online survivable Steiner tree problem, the output of GA satisfies $(k/2)$-connectivity for every demand and its cost is $O(\log n)$-competitive.*

As a direct consequence of adding edges according to GA, the $(k/2)$-connectivity is guaranteed for every demand. To complete the proof of the theorems, we need to show that the cost of the solution produced by GA is upper bounded by a factor of $O(\log^2 n)$ for forests, and $O(\log n)$ for trees.

Let $c : E(G) \to \mathbb{R}^{\geq 0}$ be the cost function on the edges. With some abuse of notation, we also use $c(Y)$ for a subset of edges $Y \subseteq E(G)$ as the sum of the cost of the edges in Y. With this notation we can say at every step i GA chooses a subset of edges P_i that satisfies $(k/2)$-connectivity and minimizes $c(P_i)$.

The overall idea of the proofs is as follows. We take an optimum solution and charge every $c(P_i)$ to $c(L_i)$, where L_i is a set of edges chosen from the optimum solution. The way we define L_i's allows them to have overlapping edges, but we show that their total cost is limited by the desired poly-logarithmic factor of the cost of the optimum solution. More specifically, we charge $c(L_i)$ to the cost of a fractional routing Q_i between s_i and t_i. Every Q_i is itself a linear combination of routes on different Steiner forests of the optimum solution. The coefficients of this linear combination are achieved from an Steiner forest packing of the optimum solution. In this fashion, the problem boils down to finding an upper bound for the total cost of routings on each Steiner forest. In the following we formally prove every step in detail.

Let OPT be an optimum offline solution of the survivable Steiner forest problem on graph G, a stream of demands S, and the connectivity requirement k. Now we define L_i for every demand i as a minimum-cost set of edges in OPT that is $(k/2)$-connected between s_i and t_i assuming the endpoints of every previous demand are contracted. In particular, we call a set of edges a *pseudo-path* between s_i and t_i if there is a path between these vertices using those edges and the edges in $\{(s_j, t_j) | \forall j < i\}$. A *pseudo-routing* between s_i and t_i is hence a set of pseudo-paths between s_i and t_i. With these definitions, L_i is a minimum-cost pseudo-routing between s_i and t_i in OPT that consists of $k/2$ pseudo-paths. The following lemma shows the relation between the costs of L_i and P_i.

▶ **Lemma 11.** *For every demand i, $c(P_i) \leq c(L_i)$.*

Proof. Every time a demand i arrives, GA finds a set P_i with the minimum cost and adds it to H in order to satisfy $(k/2)$-connectivity between s_i and t_i. Note that the endpoints

of every demand $j < i$ are already connected with $k/2$ disjoint paths in H. Besides, L_i is a pseudo-routing between s_i and t_i which is $(k/2)$-connected between s_i and t_i if we contract the two endpoints of every previous demand. Therefore adding L_i to H makes H $(k/2)$-connected between s_i and t_i. Since GA finds a minimum-cost set of edges that satisfies $(k/2)$-connectivity in H, $c(P_i)$ never exceeds $c(L_i)$. ◀

In the remaining we show how to charge the total cost of L_i's to $c(OPT)$. As a property of an optimum solution, OPT contains k edge-disjoint paths between the endpoints of every demand $(s_i, t_i) \in S$. Therefore, according to Theorem 2 there exists a solution for the fractional Steiner forest packing of OPT and demand set S with value at least $k/2$. Let $\mathbf{z}$ be a Steiner forest packing of OPT with value $k/2$. In the following we use $\mathcal{F}_S(OPT)$ to denote the collection of all Steiner forests of OPT with respect to demand set S. The theorem states there exists a vector $\mathbf{z}$ such that

$$\sum_{F \in \mathcal{F}_S(OPT)} z_F = k/2 \tag{1}$$

$$\sum_{F \in \mathcal{F}_S(OPT): e \in F} z_F \leq 1 \qquad \forall e \in OPT \ . \tag{2}$$

Moreover, the following inequality holds for the summation of the costs of these forests.

▶ **Lemma 12.** $\sum_{F \in \mathcal{F}_S(OPT)} z_F.c(F) \leq c(OPT)$.

Proof. For each forest we replace its cost with the sum of the cost of its edges.

$$\begin{aligned}
\sum_{F \in \mathcal{F}_S(OPT)} z_F.c(F) &= \sum_{F \in \mathcal{F}_S(OPT)} z_F \sum_{e \in F} c(e) \\
&= \sum_{e \in OPT} \sum_{F \in \mathcal{F}_S(OPT): e \in F} z_F c(e) \\
&= \sum_{e \in OPT} c(e) \Big(\sum_{F \in \mathcal{F}_S(OPT): e \in F} z_F \Big) \ .
\end{aligned}$$

Now we use the fact that the load on every edge in the fractional Steiner forest packing is no more than 1.

$$\begin{aligned}
\sum_{F \in \mathcal{F}_S(OPT)} z_F.c(F) &\leq \sum_{e \in OPT} c(e) \qquad \textit{Inequality} \quad (2) \\
&= c(OPT) \ .
\end{aligned}$$

◀

Now for every forest $F \in \mathcal{F}_S(OPT)$ and every demand i we define $Q_i(F)$ as a minimum-cost pseudo-path between s_i to t_i in F. This definition allows using an edge $e \in F$ multiple times in $Q_i(F)$ of different demands. Note that $Q_i(F)$ can be considered as a fractional pseudo-routing between s_i and t_i with value z_F. Considering this for all forests in $\mathcal{F}_S(OPT)$, we achieve a fractional pseudo-routing between s_i and t_i that has a value of $k/2$. We use Q_i to refer to this fractional pseudo-routing and $c(Q_i) = \sum_{F \in \mathcal{F}_S(OPT)} z_F.c(Q_i(F))$ to refer to its cost.

For every demand i we have mentioned two different pseudo-routings between s_i and t_i in OPT with value $k/2$: an integral pseudo-routing L_i, and a fractional pseudo-routing Q_i. The following lemma shows the relation between the costs of these two.

▶ **Lemma 13.** *For every L_i and Q_i pseudo-paths defined as above, we have:*

$$c(L_i) \leq c(Q_i)$$

In the interest of space, we defer the proof of Lemma 13 to the full-version of the paper.

Finally for a particular $F \in \mathcal{F}_S(OPT)$ we show an upper bound for the sum of $c(Q_i(F))$ over all demands. First let us take a closer look at every $Q_i(F)$ on a particular F. Every time a new demand (s_i, t_i) arrives $Q_i(F)$ connects its endpoints through a pseudo-path in F. This can be generalized to an algorithm for the online single-connectivity Steiner forest problem that greedily connects the endpoints of every demand by fully buying a minimum-cost pseudo-path between s_i and t_i. This is very similar to the greedy algorithm proposed in [2]. Theorem 2.1 of that paper states that their greedy algorithm is $O(\log^2 n)$-competitive. The statement of that theorem is slightly different than Claim 2.1, but the same proof verifies the correctness of the claim.

▶ **Claim 2.1.** *For the online Steiner forest problem, the algorithm that connects every demand with a minimum-cost pseudo-path is $O(\log^2 n)$-competitive.*

Now we are ready to wrap up the proof of Theorem 9.

Proof of Theorem 9. Let ALG denote the output of GA. The cost of ALG is the sum of the cost of P_i's over all demands. Therefore, by applying lemmas 11 and 13 we have

$$\begin{aligned} c(ALG) &= \sum_{(s_i,t_i)\in S} c(P_i) \\ &\leq \sum_{(s_i,t_i)\in S} c(L_i) && \textit{Lemma } 11 \\ &\leq \sum_{(s_i,t_i)\in S} c(Q_i) && \textit{Lemma } 13 \end{aligned}$$

Now we replace $c(Q_i)$ with the weighted sum of $c(Q_i(F))$'s with respect to $\mathbf{z}$.

$$\begin{aligned} c(ALG) &\leq \sum_{(s_i,t_i)\in S} \sum_{F\in\mathcal{F}_S(OPT)} z_F.c(Q_i(F)) \\ &= \sum_{F\in\mathcal{F}_S(OPT)} z_F \sum_{(s_i,t_i)\in S} c(Q_i(F)) \qquad (3) \end{aligned}$$

By applying Claim 2.1 to Inequality (3) we achieve an $O(\log^2 n)$-competitive ratio for GA.

$$\begin{aligned} c(ALG) &\leq \sum_{F\in\mathcal{F}_S(OPT)} z_F \Big(O(\log^2 n) c(F) \Big) && \textit{Claim } 2.1 \\ &\leq O(\log^2 n) \sum_{F\in\mathcal{F}_S(OPT)} z_F.c(F) \\ &\leq O(\log^2 n) c(OPT) \ . && \textit{Lemma } 12 \end{aligned}$$

◀

Finally, for the survivable Steiner tree problem we show that GA is $O(\log n)$-competitive. In other words, if one endpoint of every demand is fixed at the root, then the output of GA is at most $O(\log n)$ times the optimum offline solution. To complete the proof of Theorem 10 we use a result from [22]. In that paper the authors prove a competitive ratio of $O(\log n)$ for the algorithm which satisfies every demand using a minimum-cost pseudo-path. The following claim is a restatement of their result.

▶ **Claim 2.2.** *For the online Steiner tree problem, the algorithm that satisfies each demand with a minimum-cost pseudo-path is $O(\log n)$-competitive.*

of Theorem 10. Note that the tree problem is a special case of the forest problem, hence Inequality (3) also holds for it. By applying Claim 2.2 to that inequality the proof is complete.

$$\begin{aligned} c(ALG) &\leq \sum_{F \in \mathcal{F}_S(OPT)} z_F \Big(O(\log n) c(F) \Big) && \textit{Claim} \quad 2.2 \\ &\leq O(\log n) \sum_{F \in \mathcal{F}_S(OPT)} z_F . c(F) \\ &\leq O(\log n) c(OPT) \ . && \textit{Lemma} \quad 12 \end{aligned}$$

◀

The following Lemma shows that there exists a graph G and a sequence of demands σ such that Greedy algorithm performs $\Omega(n)$ times worse than the optimal solution.

▶ **Lemma 14.** *The competitive ratio of the greedy algorithm for survivable Steiner network design is $\Omega(n)$, even if every connectivity requirement is exactly 2.*

Proof. First we provide an online instance of the survivable network design problem where every connectivity requirement is exactly 2 and show the greedy algorithm performs poorly in comparison with the optimal solution. We construct a graph G of size n as follows. For each $1 \leq i \leq n-1$, there exist two undirected edges from node i to node $i+1$ of weights 1 and $n - i - \epsilon$ for some small $\epsilon > 0$. There exist two undirected edges from node n to node 1 with weights 1 and $n - \epsilon$. Thus G is the union of two cycles of size n. We construct a set of demands S as follows. For each $1 \leq i \leq n-1$, let $(i, i+1)$ be the i'th demand in S.

Now we analyze the output of the greedy algorithm for the input instance. We claim that after satisfying demand i the greedy algorithm has selected both edges between j and $j+1$ for every $j \leq i$. We prove this claim by induction. For the base case, when the first demand arrives the greedy algorithm chooses both edges between nodes 1 and 2 which costs $n - \epsilon$. Now assume the greedy algorithm has selected every edge between j and $j+1$ for every $j < i$ before the arrival of the i'th demand. When the i'th demand arrives, the set of edges with minimum cost that provides two edge-disjoint paths from i to $i+1$ is the two edges between i and $i+1$ which costs $n - i - \epsilon$. Thus the total cost of the greedy algorithm at the end is $\frac{n(n-1)}{2} - \epsilon n$. However, the optimum offline solution chooses the cycle containing all edges of weight 1. Thus the competitive ratio of the greedy algorithm is $\Omega(n)$. ◀

References

1. Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM Journal on Computing*, 39(2):361–370, 2009.
2. Baruch Awerbuch, Yossi Azar, and Yair Bartal. On-line generalized steiner problem. *Theoretical Computer Science*, 324(2):313–324, 2004.
3. Piotr Berman and Chris Coulston. On-line algorithms for steiner tree problems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 344–353. ACM, 1997.
4. Robert Carr and Santosh Vempala. Randomized metarounding. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 58–62. ACM, 2000.
5. Chandra Chekuri and Nitish Korula. A graph reduction step preserving element-connectivity and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 254–265. Springer, 2009.

6 Joseph Cheriyan and Mohammad R Salavatipour. Hardness and approximation results for packing steiner trees. *Algorithmica*, 45(1):21–43, 2006.

7 Julia Chuzhoy and Shi Li. A polylogarithmic approximation algorithm for edge-disjoint paths with congestion 2. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 233–242. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.54.

8 Matt DeVos, Jessica McDonald, and Irene Pivotto. Packing steiner trees. *arXiv preprint arXiv:1307.7621*, 2013.

9 Naveen Garg, Anupam Gupta, Stefano Leonardi, and Piotr Sankowski. Stochastic analyses for online combinatorial optimization problems. In *SODA*, pages 942–951. Society for Industrial and Applied Mathematics, 2008.

10 Michel X Goemans, Andrew V Goldberg, Serge A Plotkin, David B Shmoys, Eva Tardos, and David P Williamson. Improved approximation algorithms for network design problems. In *SODA*, volume 94, pages 223–232, 1994.

11 Michel X Goemans and David P Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

12 Fabrizio Grandoni, Anupam Gupta, Stefano Leonardi, Pauli Miettinen, Piotr Sankowski, and Mohit Singh. Set covering with our eyes closed. In *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 347–356. IEEE, 2008.

13 Anupam Gupta and Jochen Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, 16(1):3–20, 2011.

14 Anupam Gupta, Ravishankar Krishnaswamy, and R Ravi. Online and stochastic survivable network design. *SIAM Journal on Computing*, 41(6):1649–1672, 2012.

15 Mohammad T Hajiaghayi, Vahid Liaghat, and Debmalya Panigrahi. Online node-weighted steiner forest and extensions via disk paintings. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 558–567. IEEE, 2013.

16 MohammadTaghi Hajiaghayi, Vahid Liaghat, and Debmalya Panigrahi. Near-optimal online algorithms for prize-collecting steiner problems. In *Automata, Languages, and Programming*, pages 576–587. Springer, 2014.

17 Makoto Imase and Bernard M Waxman. Dynamic steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991.

18 Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

19 Kamal Jain, Mohammad Mahdian, and Mohammad R Salavatipour. Packing steiner trees. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 266–274. Society for Industrial and Applied Mathematics, 2003.

20 Matthias Kriesell. Edge-disjoint trees containing some given vertices in a graph. *Journal of Combinatorial Theory, Series B*, 88(1):53–65, 2003.

21 Lap Chi Lau. An approximate max-steiner-tree-packing min-steiner-cut theorem. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 61–70. IEEE, 2004.

22 Joseph Naor, Debmalya Panigrahi, and Monika Singh. Online node-weighted steiner tree and related problems. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 210–219. IEEE, 2011.

23 CSJA Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 1(1):445–450, 1961.

24 Louis Petingi and J Rodriguez. Bounds on the maximum number of edge-disjoint steiner trees of a graph. *Congressus Numerantium*, pages 43–52, 2000.

25 Jiawei Qian and David P Williamson. An o (logn)-competitive algorithm for online constrained forest problems. In *Automata, Languages and Programming*, pages 37–48. Springer, 2011.

26 R Ravi and Philip Klein. When cycles collapse: A general approximation technique for constrained two-connectivity problems. In *3rd Conference on Integer Programming and Combinatorial Optimization*, pages 39–56, 1993.

27 David P Williamson, Michel X Goemans, Milena Mihail, and Vijay V Vazirani. A primal-dual approximation algorithm for generalized steiner network problems. *Combinatorica*, 15(3):435–454, 1995.

Algorithms for Noisy Broadcast with Erasures

Ofer Grossman
EECS Departent, MIT, Cambridge, MA, USA
ofer.grossman@gmail.com

Bernhard Haeupler
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
haeupler@cs.cmu.edu

Sidhanth Mohanty
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
sidhanth@cmu.edu

Abstract

The noisy broadcast model was first studied by [10] where an n-character input is distributed among n processors, so that each processor receives one input bit. Computation proceeds in rounds, where in each round each processor broadcasts a single character, and each reception is corrupted independently at random with some probability p. [10] gave an algorithm for all processors to learn the input in $O(\log \log n)$ rounds with high probability. Later, a matching lower bound of $\Omega(\log \log n)$ was given by [11].

We study a relaxed version of this model where each reception is erased and replaced with a '?' independently with probability p, so the processors have knowledge of whether a bit has been corrupted. In this relaxed model, we break past the lower bound of [11] and obtain an $O(\log^* n)$-round algorithm for all processors to learn the input with high probability. We also show an $O(1)$-round algorithm for the same problem when the alphabet size is $\Omega(\text{poly}(n))$.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases noisy broadcast, error correction, erasures, distributed computing with noise

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.153

1 Introduction

In recent years, it is becoming increasingly common for computational tasks to be performed by multiple processors in a distributed fashion. The communication channels of these networks may have imperfections, which introduces noise to the system.

A formal version of a noise model was proposed by [7]: There are n processors: $1, 2, \ldots, n$ and each processor is given a bit. In each round, every processor broadcasts a bit to all other processors. Every processor will receive the correct message with some probability, and may receive a different (corrupted) message independently with probability $p < 1/2$ (i.e., each reception gets corrupted with probability p). The goal is for the processors to collectively compute the XOR of all their inputs. An algorithm that takes $O(\log \log n)$ rounds for all processors to learn the full input (and hence the XOR as well) was found by [10]. A matching lower bound of $\Omega(\log \log n)$ rounds was proven by [11].

All of the prior works were concerned with *substitution errors*. In this paper, we study such networks in the presence of *erasure errors*, where instead of messages getting corrupted into other messages, instead messages may get dropped. Specifically, we study the following model: in a single round each processor can broadcast a single bit b to all other processors. For each ordered pair (i, j), independently with some probability p, the character that i

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 153; pp. 153:1–153:12


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

transmitted is not received by j and a '?' is received instead. In other words, there is a string $X \in \{0,1\}^n$ and processor i is given the ith bit of X, called x_i, and the goal is for each processor to learn X using as few rounds of communication as possible. We call our noise model the *erasure model*.

1.1 Our results

We show that for any alphabet, each processor can learn the inputs of all other processors with high probability within $O(\log^* n)$ rounds. At the high level, the algorithm involves recursively running the protocol on groups of size $\log n$, and having each group encode its input using a constant rate and constant relative distance error correcting code. Then, the group collectively transmits this encoded string within a constant number of rounds. It can be shown that with high probability every processor receives enough bits to decode the group's input. There are groups for which not enough processors learn the full string (i.e., the recursive call fails), and some technical steps are needed to handle these 'failed groups'. The protocol is described in full detail in section 2.

We note that in the presence of *substitution errors*, it was proven in [11] that $\Omega(\log \log n)$ rounds are required for all processors to learn the whole input. Since we show a $O(\log^*(n))$ algorithm for the problem in the presence of *erasure errors*, this shows a fundamental difference between substitution errors and erasure errros in the broadcast model.

We then show that when the alphabet is of polynomial size, there is an $O(1)$ round algorithm for every processor to learn the full input. The algorithm involves treating the alphabets as elements of a finite field $\mathbb{F}_q$, and simulating multiplying the input vector with an appropriate random matrix. Then , the processors receive a random system of linear equations which one can show has a unique solution with high probability.

We then show that any symmetric function of the input can be computed within a constant number of rounds via computing the Hamming weight.

1.2 Related Work

A related problem was studied in [10] where the broadcast model assumed was sequential, where in one round only one processor can broadcast a bit. Additionally, the noise model assumed was that of bit flips instead of erasures. That is, each transmitted bit is independently flipped with probability p on the receiving end. In their model, [10] shows that all the processors can learn the entire input within $O(\log \log n)$ rounds. However, it left open the question of whether a faster protocol was possible.

The model of [10] was studied further in [11] where a lower bound of $\Omega(n \log \log n)$ was proven for the total number of broadcasts, thereby establishing that Gallager's protocol is optimal up to constant factors. The lower bound is proved via a reduction to another model called the generalized noisy decision tree, which is a variant of the noisy decision tree model introduced in [9]. [11] also studies whether more efficient protocols exist when the processors only want to compute some specific function on the entire input and shows that the Hamming weight can be computed with constant probability within $O(n)$ broadcasts.

We note that it follows from the lower bound in [11] that in a variant of our model where one considers substitution errors instead of erasure errors, any protocol from which all the processors learn the entire input must take $\Omega(\log \log n)$ rounds. In light of this lower bound, our result of an $O(\log^* n)$ protocol is interesting, as it shows a fundamental difference between substitution and erasure errors in this broadcast model.

Recently, a work by Efremenko, Kol, and Saxena [6] showed that under a model where the processors can adaptively choose which processor will speak in each round, the lower bound of [11] breaks down.

Note that the work of Gallager [10] shows that in the substitution model where a single processor broadcasts to the rest in a round, any function can be computed within $O(n \log \log n)$ rounds. A work by Kushilevitz and Mansour [14] studies the question of which Boolean functions can be computed within $O(n \log \log n)$ broadcasts. They determine that threshold functions can be computed with constant probability within $O(n)$ broacasts.

A paper by Feige and Killian [8] studied a harsher noise model than [10], where an adversary can arbitrarily 'uncorrupt' arbitrary corrupted bits, causing the noise to lose structure. In this harsher model, they show an $O(\log^* n)$ round protocol to compute the OR of all input bits. Newman [16] studies another noise model where each bit transmitted is independently flipped with an unknown probability that is at most p and gives algorithms that use $O(n)$ broadcasts and $O(\log^* n)$ rounds for certain classes of Boolean functions, including OR, AND, and functions with linear size AC_0 formulas.

In [1], the authors show efficient protocols to handle errors in the UCAST model, in which instead of broadcasting bits, a processor can send a different message to each other processor. They also show efficient protocols to handle errors when the communication network has certain expansion properties. For general graphs of low degree, a protocol for handling errors was found in [17], which was later shown to be optimal in [3].

Our model in the absence of errors is known as the Broadcast Congested Clique, which is a computational model often studied in distributed computing (see for example, [5, 15, 4, 2, 12]. In this model, n processors each get a piece of the input, and they work together to compute some function of this shared input. Computation proceeds in rounds, where in each round each processor can broadcast a short message to all other processors. Our work can be interpreted as showing that when using messages of constant size, every protocol in the Broadcast Congested Clique can be made resilient to erasure errors with a blowup of only $O(\log^*(n))$. In the case where messages are of logarithmic size, we show the Broadcast Congested Clique can be made resilient to erasure errors with only a constant blowup.

1.3 Notation and conventions

In this section, we state some notational conventions we use. First, we describe the computational model (without erasures), and then we formally define the model we consider with erasures.

The Computational Model: In a setting with n processors, each processor is identified with a distinct number in $[n]$. Given a string X, which we denote using an upper case character, we write the ith bit as x_i, using the corresponding lower case character. To denote the substring of X starting at position i and ending in position j we write $X_{[i,j]}$. When we wish to compute some function of a n-bit string X using n processors, assume x_i is provided as input to processor i. In the description of algorithms, $\textsc{Algo}(x_1, \ldots, x_n)$ refers to an algorithm that runs on n processors where the ith processor is given x_i as input.

In all our algorithms, we assume that each broadcast is repeated γ times where γ is some appropriately chosen constant.

Formally, we have:

▶ **Definition 1.** We let the *noisy parallel broadcast model* be a model of computation where there are n processors $P_1, \ldots, P_n$, and P_i receives input bit x_i. In each round of computation, each processor can broadcast one bit to all other processors. Each reception is corrupted

with some constant probability $0 \leq p < 1$, in which case the character '?' is received instead of the bit which was sent.

In this paper, we study the complexity of computing certain functions in the above model. Specifically, for constant erasure probability p we show a bound of $O(\log^*(n))$ for computing any function, and a bound of $O(1)$ for symmetric functions.

As part of our algorithm we use error correcting codes, so we include standard results and notations for codes below: **Error Correcting Codes:** An error correcting code is described by functions $\mathsf{Enc} : \{0,1\}^n \to \{0,1\}^k$ and $\mathsf{Dec} : \{0,1\}^k \to \{0,1\}^n$ with the property that $\mathsf{Dec}(x)$ maps to the string y for which $\mathsf{Enc}(y)$ is the closest string to x in $\mathcal{C} = \text{Image}(\mathsf{Enc})$.

The *rate* of an error correcting code is defined as $\frac{k}{n}$ and the *relative distance* is defined as $\frac{\min_{x,y \in \mathcal{C}} d(x,y)}{k}$. The quantity $\frac{d(x,y)}{2}$ is referred to as the *decoding radius*.

We use the result of [13] that error correcting code families of constant rate and constant relative distance exist. In particular, for the sake of this paper, we assume the existence of an error correcting code family E with relative distance 0.25 and rate some absolute constant K.

2 An $O(\log^* n)$ algorithm for computing any function

We consider the following message-passing model. There are n processors, and in each round, every processor transmits a single bit b to all other processors. Each processor receives each bit independently and at random with probability $1 - p$. With probability p, the character '?' is received instead. If each processor starts with a single input bit, we ask how many rounds are required so that every processor knows all input bits with high probability. We show a bound of $O(\log^*(n))$ for this problem. Specifically, we will show:

▶ **Theorem 2.** *For every $0 \leq p < 1$, there is an algorithm in the noisy broadcast parallel erasure model that computes ID_n with high probability within $O(\log^*(n) \log \frac{1}{1-p})$ rounds.*

Without loss of generality, we assume that $p \leq 0.01$, since for any erasure probability $p < 1$, repeating each message $O(\log \frac{1}{1-p})$ times can be used to effectively lower the probability of receiving '?'. We describe our algorithm for the case where the alphabet $\Sigma = \{0,1\}$. The protocol generalizes to larger alphabets in a straightforward manner.

We describe a protocol for n processors with the guarantees: at the end of the protocol, all n processors can output the full string C with probability at least $1 - \frac{1}{n^5}$, and if the protocol fails (that is, there is some processor who cannot output the full string C), then all n processors can output '$\perp$' with probability at least $1 - \frac{1}{2^{7n}}$. For the rest of this section, we assume $n \geq n_0$ for a sufficiently large n_0.

We begin by describing algorithms for simpler subproblems.

▶ **Lemma 3.** *Let b_i be the input to processor i, and let the erasure probability p be .01. Then there is an $O(1)$-round algorithm and an absolute constant α such that all processors output the AND of all b_i with probability at least $1 - 2^{-\alpha n}$.*

Proof. **Algorithm:** The algorithm is as follows: in each round, a processor i broadcasts '0' either if $x_i = 0$ or if processor i has received at least one '0' in at least one of the previous rounds. Otherwise, processor i boradcasts 1. This is repeated for 100 rounds.

Processor j's output is the AND of all bits it received.

Analysis: First, note that if all of the $b_i = 1$, then all processors must output 1, no matter what messages were corrupted, since all received bits of all processors must be 1.

Algorithm 1: EQUALITYTEST$(S_1, \ldots, S_n)$.

Enc is the encoding function of a code $\mathcal{C}$ with relative distance 0.25 and constant rate K.

1. Transmit $(\mathsf{Enc}(S_i))_{[(i-1)K+1,iK]}$ over K rounds
2. Let $A_{t,i}$ be the K-bit string received from processor t and $A_i = A_{1,i}A_{2,i}\ldots A_{n,i}$. Set c_i to 1 if Hamming distance between A_i and $\mathsf{Enc}(S_i)$ is at most $0.06Kn$ and 0 otherwise
3. The processors run the AND protocol from Lemma 3 and output the AND of all c_i

Now, suppose there is an i for which $b_i = 0$. Let t be the number of processors that received the transmission of i in the first round. The probability that processor j receives only 1s in the second round is at most p^t.

We can use Hoeffding's inequality to obtain

$$\mathbf{Pr}\left[t < \frac{n}{2}\right] \leq e^{-\alpha' n}$$

for some constant α'. Thus, the probability that there is some j that received only 1's even if there is a processor with a 0 is at most $n(e^{-\alpha' n} + p^{n/2})$, bounded above by $e^{-\alpha n}$ for a constant α. ◀

We note that the above protocol does not work in the substitution model (the model where a message may be flipped with small probability, as opposed to being corrupted to a '?'). In fact, in [11] it was proven that computing the AND function with high probability in the substitution model requires $\Omega(\log \log n)$ rounds.

We next show an $O(1)$ round algorithm for EQUALITY TESTING. Each processor is given an n-bit string S_i as input, and the goal is for all processors to output 1 if all their inputs are equal and 0 otherwise with probability at least $1 - 2^{-\Omega(n)}$. Unless otherwise specified, each step of the algorithm is from the view of processor i. Roughly speaking, this step will be used in the main algorithm to verify that all processors end up with the same output string S.

▶ **Lemma 4.** *When the erasure probability $p \leq .01$, Algorithm 1 correctly solves* EQUALITY TESTING *with probability at least $1 - 2^{-\beta n}$ for some absolute constant β.*

Proof. Let A be the string collectively transmitted by all processors in Step 1. We know

$$\begin{aligned} d(\mathsf{Enc}(S_i), \mathsf{Enc}(S_j)) &\leq d(\mathsf{Enc}(S_i), A_i) + d(A_i, A) + d(A, A_j) + d(A_j, \mathsf{Enc}(S_j)) \\ &\leq 2(d(\mathsf{Enc}(S_i), A_i) + d(\mathsf{Enc}(S_j), A_j)) \end{aligned}$$

where the second inequality is because $d(A, A_i)$ is the number of '?'s received, and lower bounds $d(\mathsf{Enc}(S_i), A_i)$.

If both $d(\mathsf{Enc}(S_i), A_i)$ and $d(\mathsf{Enc}(S_j), A_j)$ are at most $0.06Kn$, then $d(\mathsf{Enc}(S_i), \mathsf{Enc}(S_j))$ is at most $0.24Kn$, but since they are codewords of a code with relative distance 0.25, $\mathsf{Enc}(S_i) = \mathsf{Enc}(S_j)$, implying $S_i = S_j$. So if there is a pair i, j with $S_i \neq S_j$, then either c_i or c_j must be 0. And then from Lemma 3, with probability at least $1 - e^{-\alpha n}$, the processors correctly detect that there is a c_i equal to 0.

On the other hand, if all the strings are indeed equal, then c_j is 0 only if processor j receives fewer than $0.94Kn$ bits. We upper bound the probability that this happens by using Chernoff bound along with a union bound over all processors.

$$n\mathbf{Pr}[\text{processor } i \text{ receives fewer than } 0.88Kn \text{ bits}] \leq ne^{-\alpha'' n} \leq e^{-\alpha' n}$$

where α' is some constant. We let $\beta = \min\{\alpha, \alpha'\}$. ◀

Let $X = x_1x_2 \dots x_n$ be the input string and processor i is given x_i and is required to output a tuple (X_i, s_i), where X_i an n-bit string and s either 1, indicating success or 0, indicating failure, with the goal of having all $X_i = X$ and all $s_i = 1$. We say that an algorithm on a group of processors **succeeded** if $X_i = X$ and $s_i = 1$ for all i, **failed with knowledge** if $r_i = 0$ for all i, and **failed without knowledge** otherwise. We describe an algorithm for this problem where each step is from the view of processor i unless otherwise specified. Recall that each broadcast is repeated γ times to effectively reduce the erasure probability p to be at most .01. For simplicity, we assume that n is a power of 2, and so $\log n$ is an integer. It is easy to generalize the algorithm to all values of n.

At the high level, the algorithm proceeds as follows. We partition the processors into $n/\log n$ sets of size $\log n$ each (Step 2a). Then, we recursively compute the input on each of these subsets. Now, some of these subsets will have succeeded, and some will have failed. For the ones that failed, we now recompute the input, but this time we add more processors to be "helper processors". That is, the processors which succeeded in the recursive calls will now be used to aid the processors who failed in the recursive call by sending messages on their behalf. This can be seen in Step 2g, where the processor sends x_{ℓ_i}, which is the input to a processor which failed on the recursive call. This idea of using successful processors to help others who failed helps ensure that within a constant number of tries, with high probability all input bits will be known.

We now prove the following proposition, from which Theorem 2 immediately follows.

▶ **Proposition 5.** *Algorithm 2 runs in $O(\log^* n)$ rounds, succeeds (i.e., each processor outputs $(X, 1)$, where X is the input to all processors) with probability at least $1 - \frac{1}{n^5}$ and fails without knowledge with probability at most $\frac{1}{2^{7n}}$.*

Proof. We list conditions under which the protocol definitely succeeds, and show all these conditions hold with probability at least $1 - \frac{1}{n^5}$. Define R as $r'_1r'_2 \dots r'_n$ from the output of Step 2a. Define M_s as all j such that $\frac{n(s-1)}{z} < j \leq \frac{ns}{z}$ where z is the number of 0's in R.

The protocol definitely succeeds if the following conditions hold:

1. All but at most $\frac{n}{\log^3 n}$ groups succeed in the recursive call of Step 2a.
2. No group fails without knowledge in the recursive call of Step 2a, and $R_i = R$ for all i.
3. For all j such that $(R)_j = 0$, for all i, processor i receives at least one transmission from a processor in $M_{\ell_j,i}$ in Step 2g where the ℓ_jth 0 in R occurs at $(R)_j$.
4. Each processor receives at least $0.88K \log n$ bits from each successful group in at least one transmission in Step 2d of the algorithm.

Indeed, for any j in a successful group, all processors correctly learn the input to processor j because Condition 4 is met. By Condition 2, for fixed s, $M_{s,i}$ is the same for all i since $M_{s,i}$ depends on R_i. For any j in a failed group, by Condition 2, $(R)_j = (R_i)_j = 0$, and by Condition 3, each processor receives at least one transmission of processor j's input in Step 2g and so all processors correctly learn the input to processor j.

We now proceed with showing a lower bound on the probability that all of these conditions hold.

We can see that Condition 1 holds with probability at least $1 - \frac{1}{n^6}$ since by Chernoff bounds, the number of failed groups exceeds $\frac{n}{\log^3 n}$ with probability at most $\frac{1}{n^6}$.

Now suppose Condition 1 holds. A group fails without knowledge with probability at most $\frac{1}{n^7}$ by the guarantees of the protocol. The probability that there exists a group that failed without knowledge, by the union bound, is therefore at most $\frac{1}{n^6}$. If no group failed without knowledge, the only way R_i cannot equal R is if there is a group $M_{j,i}$ that processor i did not receive a single bit from. The probability that processor i does not receive a single

Algorithm 2: LearnInput$(x_1, \ldots, x_n)$.

Enc is the encoding function of a code $\mathcal{C}$ with relative distance 0.25 and constant rate K

1. **Base Case:** If $n < 100$
   a. Transmit x_i repeatedly 100 times, and set string S_i as per

$$(S_i)_j = \begin{cases} b & \text{if } b \text{ was received in any transmission from } j \\ \text{random bit} & \text{if all transmissions from } j \text{ are `?'} \end{cases}$$

   and go to Step 3a.
2. **Recursive Step:**
   a. Recursively obtain $(X'_i, r'_i) = \text{LearnInput}\left(x_{\lfloor \frac{i}{\log n} \rfloor \log n+1}, \ldots, x_{\lfloor \frac{i}{\log n} \rfloor \log n+\log n}\right)$. We call this set of processors the *group* of i.
   b. Broadcast r'_i
   c. Set R_i by setting $(R_i)_j$ to 1 if only 1's were received from j's group (i.e., from the processors which j computed the recursive call with) and 0 otherwise, for each $j \in [n]$.
   d. Let $i' = i \bmod \log n$ and transmit $\mathsf{Enc}(X'_i)_{[(i'-1)K+1, i'K]}$ over the next K rounds.
   e. Let z_i be the number of zeros in R_i and let $j = \left\lceil \frac{iz_i}{n} \right\rceil$ and let ℓ_i be the index of the jth zero in R_i. Create set $M_{s,i}$ to be all t such that

$$\frac{n(s-1)}{z_i} < t \leq \frac{ns}{z_i}$$

   f. Transmit x_i.
   g. Broadcast what was received from ℓ_i, which is either '?' or x_{ℓ_i}. Let $M'_{j,i}$ be the set of characters received from $M_{j,i}$.
   h. Set X_i by setting $(X_i)_j$ to x_i if $j = i$, by decoding the bits received in Step 2d if $(R_i)_j = 1$ and at least $0.88K \log n$ bits were received from the group of j, to a random bit if $(R_i)_j = 1$ and fewer than $0.88K \log n$ bits were received from group j in Step 2d, and to $\mathbf{1}_{1 \in M'_{j,i}}$ if $(R_i)_j = 0$. Proceed to Step 3a.
3. **Verification of output**
   a. Obtain $v_i = \text{EqualityTest}(X_1, \ldots, X_n)$ and output (X_i, v_i).

bit from this group is $p^{\gamma \log n}$, which for appropriate γ is at most $\frac{1}{n^8}$. Thus, the probability that there is some i, j pair such that processor i does not receive a single bit from group j is at most $\frac{1}{n^6}$ by a union bound. So the probability that Condition 2 is not met (given that Condition 1 is met) is at most $\frac{2}{n^6}$.

Note that $R_i = R$ means $M_s = M_{s,i}$ for all i. It follows from Chernoff bounds that the number of processors in M_s that receive the bit transmitted by processor s is at least $\log n$ with probability at least $1 - \frac{1}{n^6}$. The probability that processor i does not receive any bits from processors in M_s in any of the repetitions of Step 2g is at most $p^{\gamma \log n}$, which can be made smaller than $\frac{1}{n^8}$ by setting γ to be large enough. Now by taking a union bound over all pairs (i, s) we can conclude that Condition 3 does not hold with probability at most $\frac{1}{n^6}$.

The probability that processor i receives fewer than $0.88K \log n$ bits from group j in all repetitions of Step 2d is at most $\frac{1}{n^{c\gamma}}$ for some constant c by Chernoff bounds. A union bound across all processor-group pairs tells us that Condition 4 does not hold with probability at most $\frac{1}{n^{c\gamma-2}}$ which can be made smaller than $\frac{1}{n^6}$ with large enough γ.

Algorithm 3: LearnInputLargeAlphabet$(x_1, \ldots, x_n)$.

Let F be a function that encodes subsets of $[6 \log n]$ as elements of $\mathbb{F}_q$

1. Let $k = \lfloor 6 \log n \rfloor$ and determine $B_i = \{\frac{nj}{k} + 1, \ldots, \frac{n(j+1)}{k}\}$, where j is chosen such that $i \in B_i$
2. Broadcast x_i for 10 rounds
3. For each t from 1 to 10 and for each processor in B_i from which an entry was received in round t of Step 2, choose the processor with probability $\frac{1}{2(1-p)}$ and choose i with probability $\frac{1}{2}$. Let $T_{t,i}$ be the set of chosen elements.
4. For the next 20 rounds, processor i transmits all the $\sum_{b \in T_{t,i}} x_b$ (where the x_b are added as elements of $\mathbb{F}_q$) and $F(T_{t,i})$
5. Output X_i consistent with all received pairs $\left(\sum_{b \in T_{t,i}} x_b, F(T_{t,i})\right)$. If there is more than one possibility for such an X_i, pick one at random.

Based on the bounds we obtained on the probability that each of Conditions 1, 2, 3, 4 don't hold, we can conclude that the probability that all the conditions hold is at least $1 - \frac{1}{n^5}$.

It remains to show that the probability that the processors failed without knowledge is at most 2^{-7n}. If there is X_i such that $X_i \neq X$, then it differs from X in some index j, which means $(X_i)_j \neq (X_j)_j$ by construction of X_j implying $X_i \neq X_j$. Thus, a failure without knowledge happens only if Step 3a fails, which happens with probabilty at most $e^{-\beta n}$, which can be made smaller than 2^{-7n} by choosing the number of repetitions γ to be a large enough constant.

The number of rounds this algorithm takes is given by $T(n)$, which satisfies the recurrence relation $T(n) = T(\log n) + L$ where L is a constant and with base case $T(100) = O(1)$, which solves to $T(n) = O(\log^* n)$. ◀

3 An $O(1)$ algorithm for large alphabets

For large alphabets, in the regime where the alphabet Σ is $\mathbb{F}_q$ and $q = \text{poly}(n)$, we give a constant round algorithm to have all processors learn the input X with probability at least $1 - \frac{1}{\text{poly}(n)}$. Unless otherwise specified, the algorithm is from the view of processor i. While our algorithm works for any q that is polynomial in n, for simplicity of exposition we assume $q \geq n^6$ and that q is a prime.

▶ **Theorem 6.** *With probability at least $1 - \frac{1}{\text{poly}(n)}$, after running Algorithm 3, all processors will know all other processors' inputs. Furthermore, the algorithm terminates within $O(1)$ rounds.*

As a first ingredient towards proving Theorem 6, we prove the following lemma.

▶ **Lemma 7.** *If A is a $5k \times k$ random binary matrix where each entry is i.i.d. generated by flipping a fair coin, then with probability at least $1 - e^{-0.4k}$, A is full rank.*

Proof. Suppose V is a subspace of $\mathbb{F}_q^k$ that is not equal to all of $\mathbb{F}_q^k$, then we can find standard basis vector e_i that is not in V. Then for any binary vector v, consider v' with the bit at the i-th coordinate flipped. Either v or v' is not in V, which means at least half of the binary vectors are not in V, which means each new vector has probability at least $\frac{1}{2}$ of not being in

V. If we let $V = \mathsf{span}\{\text{vectors drawn so far}\}$, then each draw has a probability at least $\frac{1}{2}$ of increasing the dimension. Suppose we flip $5k$ coins, the probability that the number of heads is at most k is an upper bound on the probability of the span of $5k$ randomly drawn vectors not being the whole space.

By Chernoff bounds, this probability is at most $e^{-0.4k}$. ◀

Proof of Theorem 6. Each $T_{t,i}$ is a uniformly random subset of input bits of set B_i. Let x_{B_i} be a k-dimensional vector of the inputs to processors in B_i, then the transmitted characters in Round 5 are of the form $(\langle a_{B_i}, x_{B_i}\rangle, F(T_{t,i}))$ where a_{B_i} is a random binary vector, and $F(T_{t,i})$ is an encoding of a_{B_i}. The transmitted characters can be viewed as elements in the vector Ax_{B_i}, where A is a matrix whose rows are the a_{B_i}. A single processor's output of x_{B_i} is given by sampling rows of the equation $Ay_{B_i} = Ax_{B_i}$ where y_{B_i} is indeterminate and solving for y_{B_i}. If the number of sampled rows is at least $5k$, then from Lemma 7 the probability that the sampled rows span $\mathbb{F}_q^k$ and hence give a unique solution to y_{B_i} is at least $1 - \frac{1}{n^{2.4}}$.

The probability that the number of sampled rows for a group is less than $5k$ can be upper bounded by $\frac{1}{n^5}$ using Chernoff bounds.

So by union bound over all group-processor pairs (i.e., all pairs (i, B_j)), we get a $\frac{1}{\mathrm{poly}(n)}$ upper bound on the failure probability. ◀

References

1. Noga Alon, Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Reliable communication over highly connected noisy networks. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 165–173. ACM, 2016.
2. Florent Becker, Antonio Fernandez Anta, Ivan Rapaport, and Eric Reémila. Brief announcement: A hierarchy of congested clique models, from broadcast to unicast. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 167–169. ACM, 2015.
3. Mark Braverman, Klim Efremenko, Ran Gelles, and Bernhard Haeupler. Constant-rate coding for multiparty interactive communication is impossible. *Journal of the ACM (JACM)*, 65(1):4, 2017.
4. Keren Censor-Hillel, Petteri Kaski, Janne H Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 143–152. ACM, 2015.
5. Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*, pages 367–376. ACM, 2014.
6. Klim Efremenko, Gillat Kol, and Raghuvansh Saxena. Interactive coding over the noisy broadcast channel. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 24, page 93, 2017.
7. A El Gamal. Open problems presented at the 1984 workshop on specific problems in communication and computation sponsored by bell communication research. *Open Problems in Communication and Computation*, 1987.
8. Uriel Feige and Joe Kilian. Finding or in a noisy broadcast network. *Information Processing Letters*, 73(1-2):69–75, 2000.
9. Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994.
10. Robert G Gallager. Finding parity in a simple broadcast network. *IEEE Transactions on Information Theory*, 34(2):176–180, 1988.

11 Navin Goyal, Guy Kindler, and Michael Saks. Lower bounds for the noisy broadcast problem. *SIAM Journal on Computing*, 37(6):1806–1841, 2008.

12 Tomasz Jurdzinski and Krzysztof Nowicki. Msf and connectivity in limited variants of the congested clique. *arXiv preprint arXiv:1703.02743*, 2017.

13 Jørn Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18(5):652–656, 1972.

14 Eyal Kushilevitz and Yishay Mansour. Computation in noisy radio networks. In *SODA*, volume 98, pages 236–243, 1998.

15 Pedro Montealegre and Ioan Todinca. Deterministic graph connectivity in the broadcast congested clique. *arXiv preprint arXiv:1602.04095*, 2016.

16 Ilan Newman. Computing in fault tolerance broadcast networks. In *Computational Complexity, 2004. Proceedings. 19th IEEE Annual Conference on*, pages 113–122. IEEE, 2004.

17 Sridhar Rajagopalan and Leonard Schulman. A coding theorem for distributed computation. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 790–799. ACM, 1994.

A An $O(1)$ protocol for computing any symmetric function

We show that any symmetric function can be computed within $O(1)$ rounds in the model. Symmetric functions are functions whose value doesn't change under permutation of the input bits. In other words, these functions only depend on the Hamming weight of the input string. Hence, an algorithm for every processor to learn the Hamming weight of the string leads to an algorithm to compute any symmetric function. Our algorithm is inspired by a similar algorithm (for a different model) of [11].

▶ **Theorem 8.** *There is an $O(1)$ round algorithm in the noisy broadcast parallel erasure model that computes Hamming Weight(X) with probability at least 0.75.*

Our algorithm proceeds in two phases:

1. Divide the interval $[0, n]$ into subintervals of length $c\sqrt{n}$ and find which interval the Hamming weight belongs to.
2. Figure out exactly which integer in the interval is the Hamming weight.

More precisely, the first step will give us three intervals, and we will show for at least two of these intervals, with high probability all processors will end up with the same interval. Then, we will run the second step (where we pinpoint the exact Hamming weight) on each of the three intervals, and take a majority vote to compute the final output.

We describe the first step below:

▶ **Lemma 9.** *With probability at least $1 - \exp(-\Omega(n))$, for at least two t in $\{1, 2, 3\}$, all $C_{i,t}$ outputted in Step 4 of Algorithm 4 are equal and correspond to an interval containing Hamming Weight(X).*

Proof. By Chernoff bounds, the probability that h_i deviates from the truth by $t\sqrt{n}$ is at most e^{-Ct^2} for an absolute constant C. This can be made smaller than 0.01 with appropriate choice of a constant t. Then for at least two values of s, h_i lies in the correct interval in $\mathcal{B}_s$ with probability at least 0.99. Without loss of generality, say this happens for $s = 0$ and $s = 1$. Using Chernoff bounds, we can show that for some constant c, with probability at least $1 - \exp(-\Omega(n))$, at least 0.95 fraction of the processors decode the correct interval in $\mathcal{B}_0$ and $\mathcal{B}_1$.

Algorithm 4: DETERMINEINTERVAL$(x_1, \ldots, x_n)$.

$A_1, A_2, \ldots, A_k$ are disjoint intervals of size $\approx 2t\sqrt{n}$ covering $[0, n]$, with t chosen later.
Let A_i be $\varnothing$ if $i < 1$ or $i > k$.
$B_i := A_i \cup A_{i+1} \cup A_{i+2}$.
$\mathcal{B}_s := \{B_i : i \equiv s \bmod 3\}$.
Enc is the encoding function of a code with relative distance 0.25 and constant rate K.

---

1. Transmit x_i
2. Compute $h_i := \frac{\text{number of 1's received}}{1-p}$
3. For $s = 0, 1, 2$:
   a. Find interval in $\mathcal{B}_s$ containing h_i, called I. I is encoded as a string s_I (of size $O(\log n)$).
   b. Let $i' = i \bmod \log n$ and transmit $\mathsf{Enc}(s_I)_{[K(i'-1)+1, Ki']}$ over K rounds
   c. $C_{i,s} := \begin{cases} s_I & \text{if at least } .88K \log n \text{ bits were received in Step 3b} \\ \text{decoded string} & \text{if fewer than } .88K \log n \text{ bits were received in Step 3b} \end{cases}$
4. Return $C_{i,0}, C_{i,1}$ and $C_{i,2}$.

And assuming at least 0.95 fraction of the processors decode the correct intervals in $\mathcal{B}_0$ and $\mathcal{B}_1$, we can show once again using Chernoff bounds and union bound, that the number of bits from the encoded string of the correct interval received by each processor is more than $0.9Kn$ with probability at least $1 - \exp(-\Omega(n))$, which means with exponentially high probability, every processor decodes the correct interval in $\mathcal{B}_0$ and $\mathcal{B}_1$. ◀

For the second step, our goal is the following: given that every processor knows an interval $[a, b]$ in which the Hamming weight of the input string lies, it can recover the value of the Hamming weight in $O(1)$ rounds.

▶ **Lemma 10.** *On running Algorithm 5, all processors return the Hamming weight s of X with probability at least* 0.9.

Proof. Define $\widehat{\theta}_s$ to be the fraction of β_i transmitted in Step 3 that are 1.

We can lower bound $\theta_{\ell+1} - \theta_\ell$ for $x \leq \ell < y$ by $\frac{c}{\sqrt{n}}$ where c is some constant [11, Lemma 41]. The probability that $|\theta_s - \widehat{\theta}_s|$ is at most $\frac{c}{8\sqrt{n}}$ can be made at least 0.99 with an appropriate choice of the number of repetitions γ. Similarly, we can ensure that $|\widehat{\theta}_s - \widehat{\theta}_{s,i}|$ is at most $\frac{c}{8\sqrt{n}}$ with probability at least 0.99.

By Chernoff bounds, the fraction of processors for which $|\widehat{\theta}_s - \widehat{\theta}_{s,i}| < \frac{c}{8\sqrt{n}}$ is at least 0.95 with probability at least $1 - \exp(-\Omega(n))$. Thus, conditioned on $|\theta_s - \widehat{\theta}_s| < \frac{c}{8\sqrt{n}}$, we have that for at least 0.95 of the processors, $|\theta_s - \widehat{\theta}_{s,i}| < \frac{c}{4\sqrt{n}}$. Further, the string $S_1 S_2 \ldots S_n$ transmitted in Step 6 with random erasures has distance less than the decoding radius of $\mathcal{C}$ of $\mathsf{Enc}(s)$ with probability at least $1 - \exp(-\Omega(n))$, in which case all processors can correctly output s.

Since the condition $|\theta_s - \widehat{\theta}_s| < \frac{c}{8\sqrt{n}}$ holds with probability at least 0.99, the required guarantees of the Lemma hold. ◀

Proof of Theorem 8. The processors run Algorithm 4 to obtain 3 candidate intervals I_1, I_2 and I_3, and with exponentially high probability, at least two of these candidate intervals

Algorithm 5: PINPOINTWEIGHT$(x_1, x_2, \ldots, x_n; [a, b])$.

$[a, b]$ is the interval of length up to $3\sqrt{n}$ where the Hamming weight is promised to lie
Enc is the encoding function of a code $\mathcal{C}$ with relative distance 0.25 that maps $\log n$ bit strings to $K \log n$ bit strings
Let θ_s be defined as the probability that when flipping s coins, each coming up heads with probability $1 - p$, at least $(1 - p)\left(\frac{a+b}{2}\right)$ come up heads.

---

1. Transmit x_i
   Let Y be the number of 1's received.
2. $\beta_i := \begin{cases} 1 & \text{if number of 1's received is greater than } (1-p)\left(\frac{a+b}{2}\right) \\ 0 & \text{otherwise} \end{cases}$
3. Transmit β_i
4. Let $\widehat{\theta}_{s,i}$ be the fraction of received bits from Step 3 that are 1 (i.e., the total number of 1's received, divided by the total number of 1's or 0's received).
5. $\widehat{s}_i = \arg\min_\ell |\theta_\ell - \widehat{\theta}_{s,i}|$
6. Let $i' = i \bmod \log n$ and transmit $\mathsf{Enc}(\widehat{s}_i)_{[K(i'-1)+1, Ki']}$ over K rounds
7. $\tilde{s}_i = \begin{cases} \text{decoded string} & \text{if at least } .88K \log n \text{ bits were received in Step 6} \\ \widehat{s}_i & \text{if fewer than } .88K \log n \text{ bits were received in Step 6} \end{cases}$

contain the Hamming weight. The processors run Algorithm 5 on each of the three intervals and processor i obtains outputs n_0, n_1 and n_2 respectively. With constant probability, at least two of n_0, n_1 and n_2 are the same and equal to the correct Hamming weight, and hence outputting the majority of the three matches the guarantee. ◀

Efficient Black-Box Reductions for Separable Cost Sharing

Tobias Harks
Universität Augsburg, Institut für Mathematik, Augsburg, Germany
tobias.harks@math.uni-augsburg.de

Martin Hoefer
Goethe-Universität Frankfurt am Main, Institut für Informatik, Frankfurt am Main, Germany
mhoefer@cs.uni-frankfurt.de

Anja Huber
Universität Augsburg, Institut für Mathematik, Augsburg, Germany
anja.huber@math.uni-augsburg.de

Manuel Surek
Universität Augsburg, Institut für Mathematik, Augsburg, Germany
manuel.surek@math.uni-augsburg.de

Abstract

In *cost sharing games with delays,* a set of agents jointly uses a finite subset of resources. Each resource has a fixed cost that has to be shared by the players, and each agent has a non-shareable player-specific delay for each resource. A prominent example is uncapacitated facility location (UFL), where facilities need to be opened (at a shareable cost) and clients want to connect to opened facilities. Each client pays a cost share and his non-shareable physical connection cost. Given any profile of subsets used by the agents, a *separable cost sharing protocol* determines cost shares that satisfy budget balance on every resource and separability over the resources. Moreover, a separable protocol guarantees existence of pure Nash equilibria in the induced strategic game for the agents.

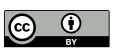
In this paper, we study separable cost sharing protocols in several general combinatorial domains. We provide black-box reductions to reduce the design of a separable cost sharing protocol to the design of an approximation algorithm for the underlying cost minimization problem. In this way, we obtain new separable cost sharing protocols in games based on arbitrary player-specific matroids, single-source connection games without delays, and connection games on n-series-parallel graphs with delays. All these reductions are efficiently computable – given an initial allocation profile, we obtain a profile of no larger cost and separable cost shares turning the profile into a pure Nash equilibrium. Hence, in these domains any approximation algorithm can be used to obtain a separable cost sharing protocol with a price of stability bounded by the approximation factor.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory, Theory of computation → Exact and approximate computation of equilibria, Theory of computation → Network games

Keywords and phrases Cost Sharing, Price of Stability, Matroids, Connection Games

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.154

Related Version A full version of the paper is available at [34], `https://arxiv.org/abs/1802.10351`.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 154; pp. 154:1–154:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding The research of the authors from Augsburg was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - HA 8041/1-1.

1 Introduction

Cost sharing is a fundamental task in networks with strategic agents and has attracted a large amount of interest in algorithmic game theory. Traditionally, cost sharing has been studied in a cooperative sense, i.e., in the form of cooperative games or mechanism design. Many of these approaches treat cost in a *non-separable* way and return a single, global cost share for each agent. In contrast, when agents jointly design a resource infrastructure in large networks, it is much more desirable to provide algorithms and protocols for *separable* cost sharing that specify which agent needs to pay how much to each resource. Here the natural approach are strategic cost sharing games with n players that use subsets of m resources. Each resource generates a cost depending on the subset of players allocating it. A protocol determines a cost share for each resource and each player using it. In addition to separability, there are further natural desiderata for such protocols, such as budget-balance (distribute exactly the arising cost of each resource) and existence of a pure Nash equilibrium (PNE), i.e., allow the resulting game to stabilize.

Perhaps the most prominent such protocol is the fair share protocol, in which the cost of each resource is allocated in equal shares to the players using it. This approach has been studied intensively (see our discussion below), but there are several significant drawbacks. Even in connection games on undirected networks, it can be PLS-hard to find a PNE [51] and the PoS (the total cost of the best Nash equilibrium compared to the cost of the optimal allocation) is not known to be constant. This contrasts the fact that there are polynomial time approximation algorithms with low approximation factors, see, e.g. [12]. For directed networks the PoS can even be as large as $\Omega(\log n)$ [4, 17].

In this paper, we study a slight generalization of cost sharing games, where every resource has a shareable cost component and a non-shareable player-specific delay component. The shareable cost needs to be shared by the players using it, the non-shareable player-specific delay represents, e.g., a physical delay and is thus unavoidable. This setting arises in several relevant scenarios, such as uncapacitated facility location (UFL) [37]. Here players share the monetary cost of opened facilities but additionally experience delays measured by the distance to the closest open facility. Another important example appears in network design, where players jointly buy edges of a graph to connect their terminals. Besides the monetary cost for buying edges, each player experiences player-specific delays on the chosen paths. In such a distributed network environment, it is not clear a priori if an optimal solution can be stable – i.e., if the shareable costs can be distributed among the players in a separable way so that players do not want to deviate from it. This question leads directly to the design of protocols that distribute the costs in order to induce stable and good-quality solutions of the resulting strategic game.

Our results are three polynomial-time black-box reductions for the price of stability (PoS) of separable cost sharing protocols in combinatorial resource allocation problems. Our domains represent broad generalizations of UFL – arbitrary, player-specific matroids, single-source connection games without delays, and connection games on undirected n-series-parallel graphs with delays. In each of these domains, we take as input an arbitrary profile and efficiently turn it into a profile having no larger cost and a sharing of the shareable costs such that it is a Nash equilibrium. Our protocols are polynomial-time in several ways. Firstly, the games we study are succinctly represented. In matroids, we assume that strategies

are represented implicitly via an independence oracle. For connection games on graphs, the strategies of each player are a set of paths, which is implicitly specified by terminal vertices of the player and the graph structure. The cost sharing protocol is represented by a strategy profile S and a sharing of the shareable costs arising in S on each resource. While in principle the protocol must specify a sharing of the costs for all of the other (possibly exponentially many) strategy profiles, one can do so implicitly by a simple lexicographic assignment rule. It guarantees that the profile S becomes a PNE. As such, starting from an arbitrary initial profile S', we can give in polynomial time the Nash equilibrium profile S, the cost shares for S, and the assignment rule for cost shares in the other profiles. Hence, if S' is polynomial-time computable, then both protocol and Nash equilibrium S are both polynomial-time computable and polynomial-space representable.

1.1 Our Results

We present several new polynomial-time black-box reductions for separable cost sharing protocols with small PoS. We study three domains that represent broad generalizations of the uncapacitated facility location problem. In each domain, we devise an efficient black-box reduction that takes as input an arbitrary strategy profile and computes a new profile of no larger cost together with a separable cost sharing protocol inducing the new profile as a PNE. Thus, *any* polynomial-time α-approximation of the social cost can be turned into a separable cost sharing protocol with PoS at most α.

Matroidal Set Systems. In Section 3 we provide a black-box reduction for matroidal set systems. Our results even apply to the broad class of *subadditive* cost functions that include fixed costs and discrete concave costs even with weighted players as a special case. Here we assume access to a value oracle for the subadditive cost function for each resource. Matroidal set systems with player-specific delays include uncapacitated facility location as a special case, since these correspond to matroid games, where each player has a uniform rank 1 matroid. For *metric* UFL, there is for instance a 1.488-approximation algorithm [45] using ideas of a previous 1.5-approximation algorithm [11]. This leads to a separable cost sharing protocol with PoS equal 1.488. Also, the existing hardness results for UFL carry over, meaning that for any polynomial-time computable, separable cost sharing protocol, the cost of an equilibrium that can be computed in polynomial time is bounded from below by the inapproximability bound. For metric UFL there is a lower bound of 1.46 [31].

Single-Source Connection Games with Fixed Costs. In Section 4 we consider cost sharing games on graphs, where the set systems correspond to paths connecting a global source with a player-specific terminal. We again provide a polynomial-time black-box reduction. Our result improves significantly over the existing Prim-Sharing [17] with a PoS of 2. We obtain separable protocols based on any approximation algorithm for Steiner tree, such as, e.g., the classic 1.55-approximation algorithm [49], or the celebrated recent 1.39-approximation algorithm [12]. Our black-box reduction continues to hold even for directed graphs, where we can use any algorithm for the Directed Steiner Tree problem [15], or games based on the (directed or undirected) Group Steiner Tree problem [16, 26]. All lower bounds on approximation hardness translate to the quality of polynomial-time computable equilibria of polynomial-time computable separable protocols.

Connection Games With Delays. Finally, in Section 5 we study multi-terminal connection games with delays and fixed costs. For directed graphs, an optimal Steiner forest is not enforceable by a separable cost sharing protocol, even for two players [17]. Very recently,

a similar result was shown even for two-player games on undirected graphs [35]. Thus, for general graphs, we cannot expect separable protocols with optimal or close-to-optimal equilibria, or (efficient) black-box reductions. We introduce a class of so-called n-series-parallel graphs, which allows to obtain a black-box reduction in polynomial time. The transformation directly implies that the n-series-parallel graphs always admit a separable cost sharing protocol inducing an optimal Steiner forest as an equilibrium.

The reduction also applies to discrete-concave cost functions and player-specific delays, however, we do not know if polynomial running time is guaranteed. n-series-parallel graphs have treewidth at most 2, thus, for fixed edge costs and no delays, it is possible to compute efficiently even an optimal Steiner forest [7]. Hence, in this case we obtain a separable protocol with PoS of 1 in polynomial time. We finally demonstrate that the specific setting of n-series-parallel graphs is in some sense necessary: Even for generalized series-parallel graphs we give a counterexample showing that a black-box reduction is impossible to achieve.

1.2 Preliminaries and Related Work

Cooperative cost sharing games have been studied over the last decades for a variety of combinatorial optimization problems, such as minimum spanning tree [10], Steiner tree [29,30, 46,52], facility location [28], vertex cover [22], and many more. Cooperative cost sharing games have interesting implications for (group-)strategyproof cost sharing mechanisms [41,42,47,48]. For Bayesian cost-sharing mechanisms there even exist efficient black-box reductions from algorithm to mechanism design [27]. A major difference to our work is that cooperative cost sharing is *not separable*.

The most prominent example of a *separable* cost sharing protocol is the *fair share protocol*, in which the cost of each resource is divided in equal shares among the players that use it. This protocol is also anonymous, and it implies that the resulting game is a congestion game [50]. It guarantees the smallest PoS within a class of anonymous protocols [17]. The fair share protocol has attracted a serious amount of research interest over the last decade [1, 4, 8, 33], especially the notorious open problem of a constant PoS for connection games in undirected graphs [9, 23, 25, 43, 44]. However, as a significant drawback - outside of the domain of undirected connection games - the PoS is often as large as $\Omega(\log n)$, e.g. [17, 32].

More general separable protocols have been studied mostly in terms of the price of anarchy, e.g., for scheduling (or matroid games) [6, 13, 19, 24, 53] or single-source network design with [20, 21] and without uncertainty [17]. The best result here is a price of anarchy (and stability) of 2 via Prim-Sharing [17], a protocol inspired by Prim's MST algorithm. A protocol with logarithmic PoS was shown for capacitated UFL games [37].

We observe that separable protocols with low PoS can be obtained using results for cost sharing games with so-called "arbitrary sharing", where effectively players pay the cost increment when changing their strategy (see [36] for a formal definition). A PNE for arbitrary sharing can be translated directly into a PNE for a suitable separable cost sharing protocol. A simple proof of Proposition 1 can be found in the full version of this paper [34].

▶ **Proposition 1.** *If for a cost sharing model, the non-cooperative game with arbitrary sharing has a PNE, then there is a separable cost sharing protocol with the same PNE.*

This implies existence of separable protocols with optimal PNE and PoS 1 for a variety of classes of games, including matroid games with uniform discrete-concave costs [36], uncapacitated facility location with fixed [14] and discrete-concave costs [39], connection games (single-source [5, 38] and other classes [2, 3, 40]) with fixed costs, and more. However, the large majority of these results are *inefficient*, i.e., there is no polynomial-time algorithm that computes the required optimal equilibrium.

Alternatively, one may resort to approximate equilibria in games with arbitrary sharing that are efficiently computable. The most prominent technique works via reducing costs by an additive value ε to ensure polynomial running time (put forward for single-source connection games in [5] and used in much of the follow-up work [2, 3, 14, 38]). This approach *does not translate* to separable protocols, since a player must eventually contribute to *all resources*. This is impossible for the model we consider here.

2 Separable Cost Sharing Protocols

We are given a finite set N of players and a finite set E of resources. Each player $i \in N$ is associated with a predefined family of subsets $\mathcal{S}_i \subseteq 2^E$ from which player i needs to pick at least one. The space of strategy profiles is denoted by $\mathcal{S} := \times_{i \in N} \mathcal{S}_i$. For $S \in \mathcal{S}$ we denote by $N_e(S) = \{i \in N : e \in S_i\}$ the set of players that use resource e. Every resource $e \in E$ has a fixed cost $c_e \geq 0, e \in E$ that is assumed to be *shareable* by the players. In addition to the shareable costs, there are *player-specific constant costs* $d_{i,e} \geq 0, i \in N, e \in E$ that are not shareable. If player i chooses subset S_i, then the player-specific costs $\sum_{e \in S_i} d_{i,e}$ must be paid completely by player i. The total cost of a profile S is defined as $C(S) = \sum_{e \in \cup_{i \in N} S_i} c_e + \sum_{i \in N} \sum_{e \in S_i} d_{i,e}$.

A *cost sharing protocol* assigns cost share functions $\xi_{i,e} : \mathcal{S} \to \mathbb{R}_{\geq 0}$ for all $i \in N$ and $e \in E$ and thus induces the strategic game $(N, \mathcal{S}, \xi)$. For a player i, her total private cost of strategy S_i in profile S is $\pi_i(S) := \sum_{e \in S_i} (\xi_{i,e}(S) + d_{i,e})$. We assume that every player picks a strategy in order to minimize her private cost. A prominent solution concept in non-cooperative game theory are pure Nash equilibria. Using standard notation in game theory, for a strategy profile $S \in \mathcal{S}$ we denote by $(S'_i, S_{-i}) := (S_1, \ldots, S_{i-1}, S'_i, S_{i+1}, \ldots, S_n) \in \mathcal{S}$ the profile that arises if only player i deviates to strategy $S'_i \in \mathcal{S}_i$. A profile is a *pure Nash equilibrium (PNE)* if for all $i \in N$ it holds $\pi_i(S) \leq \pi_i(S'_i, S_{-i})$ for all $S'_i \in \mathcal{S}_i$.

In order to be practically relevant, cost sharing protocols need to satisfy several desiderata. In this regard, *separable* cost sharing protocols are defined as follows [17].

▶ **Definition 2** (Cost Sharing Protocols and Enforceability)**.** A cost sharing protocol is

1. *stable* if it induces only games that admit at least one pure Nash equilibrium.
2. *budget balanced,* if for all $e \in E$ with $N_e(S) \neq \emptyset$

$$c_e = \sum_{i \in N_e(S)} \xi_{i,e}(S) \text{ and } \xi_{i,e}(S) = 0 \text{ for all } i \notin N_e(S).$$

3. *separable* if it is stable, budget-balanced and induces only games for which in any two profiles $S, S' \in \mathcal{S}$ for every resource $e \in E$,

$$N_e(S) = N_e(S') \Rightarrow \xi_{i,e}(S) = \xi_{i,e}(S') \text{ for all } i \in N_e(S).$$

4. *polynomial time computable,* if the cost sharing functions ξ can be computed in polynomial time in the encoding length of the cost sharing game.

We call a strategy profile S *enforceable*, if there is a separable protocol inducing S as a PNE.

Separability means that for any two profiles S, S' the cost shares on e are the same if the set of players using e remains unchanged. Still, separable protocols can assign cost share functions that are specifically tailored to a given congestion model, for example based on an optimal profile. In this paper, we are additionally interested in *polynomial-time computable protocols* that we introduce here.

3 Matroid Games

In this section, we consider matroid games. As usual in matroid theory, we will write $\mathcal{B}_i$ instead of $\mathcal{S}_i$, and $\mathcal{B}$ instead of $\mathcal{S}$, when considering matroid games. The tuple $\mathcal{M} = (N, E, \mathcal{B}, (c_e)_{e\in E}, (d_{i,e})_{e\in E, i\in N})$ is called a *matroid game* if $E = \bigcup_{i\in N} E_i$, and each set system $\mathcal{B}_i \subseteq 2^{E_i}$ forms the base set of some matroid $\mathcal{M}_i = (E_i, \mathcal{B}_i)$. While seemingly abstract, the class includes several prominent application domains, such as UFL games. In a UFL game, the resources are facilities (e.g. common transport hubs) and the players incur delay $d_{i,e}$ in addition to their cost shares for opening used facilities. Every player i chooses exactly one resource, that is $|B_i| = 1$ for all $B_i \in \mathcal{B}_i$ and $i \in N$ and hence $\mathcal{B}_i$ corresponds to a *uniform matroid of rank one*. Recall that every base B of a matroid $\mathcal{M}_i = (E_i, \mathcal{B}_i)$ has the same cardinality which we denote with rk_i (the rank of $\mathcal{M}_i$).

In the following, instead of fixed costs on the resources, we allow for *subadditive* cost functions $c_e : 2^N \to \mathbb{R}_{\geq 0}, e \in E$. c_e is called *subadditive*, if it satisfies (1) $c_e(S) \leq c_e(T)$ for all $S \subseteq T \subseteq N$, and (2) $c_e(S + \{i\}) \leq c_e(S) + c_e(\{i\})$ for all $S \subset N, i \in N$. Note that subadditive functions include fixed costs, but also discrete concave costs, where the nondecreasing cost $c_e : \mathbb{N} \to \mathbb{R}_{\geq 0}$ depends on the number of players using e and satisfies $c_e(x + \delta) - c_e(x) \geq c_e(y + \delta) - c_e(y) \forall x \leq y;\ x, y, \delta \in \mathbb{N}$. As such, in the discrete concave setting, all players have the same weight of 1, whereas subadditive costs also allow for different weights (as in weighted congestion games). We furthermore assume $c_e(\emptyset) = 0, e \in E$.

Let us denote the cost of the cheapest alternative of player i to resource e for profile $B \in \mathcal{B}$ by $\Delta_i^e(B) := \min_{f\in E, B_i+f-e\in\mathcal{B}_i} (c_e(B_i + f - e, B_{-i}) + d_{i,f})$. Here we use the simplified notations $B_i + f - e := B_i \cup \{f\} \setminus \{e\}$ and $c_e(B) := c_e(N_e(B))$. We recapitulate the following characterization of enforceable strategy profiles (obtained in [37], where also a cost sharing protocol inducing the enforceable profile as a PNE is given).[1]

▶ **Lemma 3.** *A profile $B = (B_1, \ldots, B_n)$ is enforceable iff the following two properties are satisfied. Note that* (D1) *implies that each summand $\Delta_i^e(B) - d_{i,e}$ in* (D2) *is nonnegative.*

$$d_{i,e} \leq \Delta_i^e(B) \text{ for all } i \in N, e \in B_i \tag{D1}$$

$$c_e(B) \leq \sum_{i\in N_e(B)} (\Delta_i^e(B) - d_{i,e}) \text{ for all } e \in E. \tag{D2}$$

▶ Remark. The characterization was used in [37] to prove that an optimal collection of bases is enforceable. This implies a PoS of 1 for a separable cost sharing protocol that relies on the optimal profile. As such, the protocol is not efficiently computable (unless $P = NP$).

In the following, we devise a black-box reduction in Algorithm 1. It takes as input an arbitrary collection of bases B and transforms them *in polynomial time* into an enforceable set of bases B' of no larger cost. We define for each $i \in N, e \in E$ a *virtual cost value* $v_i^e := c_e(\{i\}) + d_{i,e}$, and for each $B \in \mathcal{B}, i \in N, e \in E$ a *virtual deviation cost* $\bar{\Delta}_i^e(B) := \min_{f\in E, B_i+f-e\in\mathcal{B}_i} v_i^f$. The algorithm now iteratively checks whether (D1) and (D2) from Lemma 3 hold true (in fact it checks this condition for smaller values on the right hand side given by the virtual values), and if not, exchanges one element of some player. We show that the algorithm terminates with an enforceable profile after polynomially many steps.

▶ **Theorem 4.** *Let B be a strategy profile for a matroid congestion model with subadditive costs. There is an enforceable profile B' with $C(B') \leq C(B)$ that can be computed in at most*

[1] The original characterization in [37] was proven for weighted players and load-dependent non-decreasing cost functions but the proof also works for subadditive cost functions.

Algorithm 1: Transforming any profile B into an enforceable profile B'

Input: Congestion model $(N, \mathcal{B}, c, d)$ and profile $B \in \mathcal{B}$
Output: Enforceable profile B' with $C(B') \leq C(B)$.

1 Set $B' \leftarrow B$
2 **while** *there is* $e \in E$ *that satisfies at least one of the following conditions:*

$$d_{i,e} > \bar{\Delta}_i^e(B') \text{ for some } i \in N_e(B') \quad (1) \text{ or}$$

$$c_e(B') > \sum_{i \in N_e(B')} \left(\bar{\Delta}_i^e(B') - d_{i,e}\right) \quad (2) \textbf{ do}$$

3 **if** (1) *holds true for some* $i \in N_e(B')$ **then**
4 Let $f_i \in \arg \min_{\substack{f \in E \\ B_i' + f - e \in \mathcal{B}_i}} v_i^f$
5 Update $B_i' \leftarrow B_i' + f_i - e$
6 **else if** (2) *holds true* **then**
7 **while** (2) *holds true on* e **do**
8 Pick $i \in N_e(B')$ with $v_i^e > \bar{\Delta}_i^e(B')$
9 Let $f_i \in \arg \min_{\substack{f \in E \\ B_i' + f - e \in \mathcal{B}_i}} v_i^f$
10 Update $B_i' \leftarrow B_i' + f_i - e$

$n \cdot m \cdot rk(\mathcal{B})$ iterations of the outer while-loop in Algorithm 1, where $rk(\mathcal{B}) = \max_{i \in N} rk_i$. Furthermore, each such iteration needs $\mathcal{O}(n \cdot m \cdot rk(\mathcal{B}) \cdot Q + (m+n) \cdot Q')$ time, where Q and Q' denote the maximum complexities of the independence oracles for the players' strategies, and of the value oracles for the subadditive cost functions of the resources, respectively.

Proof. First, observe that if (D1) and (D2) from Lemma 3 hold true for smaller values $0 \leq \bar{\Delta}_i^e(B) \leq \Delta_i^e(B), i \in N, e \in E$, then the profile B is also enforceable. Hence, if the algorithm terminates, the resulting strategy profile B' will be enforceable.

To show that the algorithm is well-defined, we only need to check Line 8. By subadditivity we get $\sum_{i \in N_e(B')} c_e(\{i\}) > c_e(B')$. Thus, whenever $c_e(B') > \sum_{i \in N_e(B')} \left(\bar{\Delta}_i^e(B') - d_{i,e}\right)$, there is an $i \in N_e(B')$ with $c_e(\{i\}) + d_{i,e} > \bar{\Delta}_i^e(B')$.

We now bound the running time. Consider player i and the matroid bases $\mathcal{B}_i$. We interpret a basis $B_i \in \mathcal{B}_i$ as distributing exactly rk_i unit sized packets over the resources in E. This way, we can interpret the algorithm as iteratively moving packets away from those resources $e \in E$ for which either (1) or (2) holds true. We give each packet a unique ID $i_k, k = 1, \ldots, \mathrm{rk}_i$. For $B_i \in \mathcal{B}_i$, let e_{i_k} denote the resource on which packet i_k is located. We now analyse the two types of packet movements during the execution of the algorithm. For a packet movement executed in Line 5 of Algorithm 1, we have $d_{i,e} > \bar{\Delta}_i^e(B')$, thus, when packet i_k located on $e = e_{i_k}$ is moved to f_i, it holds that $v_i^{e_{i_k}} = v_i^e \geq d_{i,e} > \bar{\Delta}_i^e(B') = v_i^{f_i}$. For packet movements executed in Line 10, then by the choice of player $i \in N_e(B')$ (see Line 8) for the corresponding packet i_k it holds $v_i^{e_{i_k}} = v_i^e > \bar{\Delta}_i^e(B') = v_i^{f_i}$. In both cases we obtain $v_i^e > v_i^{f_i}$. Hence, every movement of a single packet i_k is in strictly decreasing order of virtual value of the resource. Note that the virtual cost value v_i^e does not depend on profile B. Thus, there are at most m different virtual cost values that a packet i_k of player i can experience, and thus packet i_k can move at most $m - 1$ times. The following is an upper bound on the

total number of packet movements for all players $\sum_{i \in N} \mathrm{rk}_i \cdot (m-1) \leq n \cdot m \cdot \mathrm{rk}(\mathcal{B})$. Thus, the number of iterations of the outer while-loop can be at most that value. It is straightforward to observe the stated complexity of one iteration of the while-loop.

It is left to argue that the final output B' has cost at most $C(B)$. We prove this inductively by the different types of packet movements. Consider first a packet movement of type (1). Let B and B' be the profiles before and after packet i_k has been moved from e to f_i, respectively. We obtain (by using subadditivity and assumption (1))

$$\begin{aligned} C(B') - C(B) &= (c_{f_i}(B') - c_{f_i}(B) + d_{i,f_i}) - (c_e(B) - c_e(B') + d_{i,e}) \\ &\leq c_{f_i}(\{i\}) + d_{i,f_i} - (c_e(B) - c_e(B') + d_{i,e}) \\ &= \bar{\Delta}_i^e(B) - d_{i,e} + (c_e(B') - c_e(B)) \leq \bar{\Delta}_i^e(B) - d_{i,e} \quad < \quad 0. \end{aligned}$$

Now consider packet movements of type (2). We treat all movements occurring in one run of the while loop in Line 7. Let B denote the profile before and B' after all these movements. Let $T_e(B) \subseteq N_e(B)$ denote the set of those players whose packet i_k on e is moved to f_i for $i \in T_e(B)$ during the while loop. Let $F_e(B) = \bigcup_{i \in T_e(B)} \{f_i\}$ and for $i \in T_e(B)$ define $T_{f_i}(B) = \{j \in T_e(B) \mid f_j = f_i\}$. We derive some useful observations. Before entering the while loop, it holds

$$c_e(B) > \sum_{i \in N_e(B)} \left(\bar{\Delta}_i^e(B) - d_{i,e}\right) = \sum_{i \in N_e(B) \setminus T_e(B)} \left(\bar{\Delta}_i^e(B) - d_{i,e}\right) + \sum_{i \in T_e(B)} \left(\bar{\Delta}_i^e(B) - d_{i,e}\right). \quad (3)$$

Moreover, after exiting the while loop it holds

$$c_e(B') \leq \sum_{i \in N_e(B) \setminus T_e(B)} \left(\bar{\Delta}_i^e(B) - d_{i,e}\right). \quad (4)$$

Thus, combining (3) and (4) we get

$$c_e(B) - c_e(B') > \sum_{i \in T_e(B)} \left(\bar{\Delta}_i^e(B) - d_{i,e}\right). \quad (5)$$

Putting everything together, we obtain

$$\begin{aligned} C(B') - C(B) &= \sum_{f_i \in F_e(B)} \left(c_{f_i}(B') - c_{f_i}(B)\right) + \sum_{i \in T_e(B)} d_{i,f_i} - \left(c_e(B) - c_e(B') + \sum_{i \in T_e(B)} d_{i,e}\right) \\ &\leq \sum_{f_i \in F_e(B)} \sum_{j \in T_{f_i}(B)} c_{f_i}(\{i\}) + \sum_{i \in T_e(B)} d_{i,f_i} - \left(c_e(B) - c_e(B') + \sum_{i \in T_e(B)} d_{i,e}\right) \\ &= \sum_{i \in T_e(B)} \bar{\Delta}_i^e(B) - \left(c_e(B) - c_e(B') + \sum_{i \in T_e(B)} d_{i,e}\right) \quad < \quad 0, \end{aligned}$$

where the first inequality follows from subadditivity, and the last inequality from (5). ◀

4 Single-Source Connection Games without Delays

In this section, we study connection games in an undirected graph $G = (V, E)$ with a common source vertex $s \in V$. Every player i wants to connect a player-specific terminal node $t_i \in V$ to s. Consequently, every strategy P_i of player i is an (s, t_i)-path in G. We denote the set of paths for player i by $\mathcal{P}_i$ and the set of profiles by $\mathcal{P}$. Furthermore we focus on fixed shareable costs $c_e \geq 0$ and no player-specific delays $d_{i,e} = 0$, for all $i \in N$, $e \in E$.

For single-source games with delays, a black-box reduction is impossible to achieve, since one can construct instances where no optimal solution is enforceable. To see this, take any

multi-source multi-terminal connection game and introduce a new auxiliary source vertex s. Then connect s to each s_i with an auxiliary edge e_i, which has cost $c_{e_i} = 0$ and delays $d_{i,e_i} = 0$, $d_{j,e_i} = M$ for all other players j (for some prohibitively large constant M). Now in any equilibrium and any optimal state of the resulting game, player i will choose an (s, t_i)-path which begins with edge e_i. Moreover, e_i does not generate additional cost for player i. As such, the optimal solutions, the Nash equilibria, and their total costs correspond exactly to the ones of the original multi-source game. For the general multi-source case, there are instances with no enforceable optimal solution (even for only two players, see [35]).

For connection games with fixed shareable costs and no delays, the state of the art for polynomial-time cost sharing protocols is the Prim-Sharing protocol with a PoS of 2 [17]. Moreover, it is known that an optimal *tree* profile (a profile in which the union of player paths constitute a tree) is always enforceable [5,17]. Our result for polynomial-time computation of cheap enforceable profiles, stated in Theorem 5, represents a significant generalization – for any (possibly non-enforceable) tree profile, Algorithm 2 (given in detail in [34]) computes an enforceable profile that can only be cheaper in terms of total cost. Moreover, by combining the result with existing approximation algorithms, we obtain a protocol with PoS of 1.39.

▶ **Theorem 5.** *Let P be a strategy profile for a single-source connection game with fixed costs. There is an enforceable profile P' with $C(P') \leq C(P)$ that can be computed by Algorithm 2 in polynomial time.*

It is straightforward that for fixed costs we can transform each profile P into a tree profile $\hat{P}$, in which the union of player paths constitute a tree T, without increasing the cost. Over the course of the algorithm, we adjust this tree and construct a cost sharing for it in a bottom-up fashion. The approach has similarities to an approach for obtaining approximate equilibria for single-source cost sharing games with arbitrary sharing [5]. However, our algorithm exploits crucial properties of separable protocols, thereby providing an exact Nash equilibrium and polynomial running time.

When designing a separable protocol based on a state $\hat{P}$, we can always assume that when a player i deviates unilaterally to one or more edges $e \in G \setminus \hat{P}_i$, she needs to pay all of c_e. As such, player i always picks a collection of shortest paths with respect to c_e between pairs of nodes on her current path $\hat{P}_i$. All these paths in G are concisely represented in the algorithm as "auxiliary edges". The algorithm initially sets up an auxiliary graph $\hat{G}$ given by T and the set of auxiliary edges based on $\hat{P}$. It adjusts the tree T by removing edges of T and adding auxiliary edges in a structured fashion.

We first show in the following Lemma 6 (for a proof see [34]) that this adjustment procedure improves the total cost of the tree, and that the final tree $\hat{T}$ is enforceable in $\hat{G}$. In the corresponding cost sharing, every auxiliary edge contained in $\hat{T}$ is completely paid for by a single player that uses it. In the subsequent proof of Theorem 5, we only need to show that for the auxiliary edges in $\hat{T}$, the edge costs of the corresponding shortest paths in G can be assigned to the players such that we obtain an equilibrium in G. The proof shows that the profile P' evolving in this way is enforceable in G and has cost no larger than P.

▶ **Lemma 6.** *Algorithm 2 computes a cost sharing of a feasible tree $\hat{T}$ in the graph $\hat{G}$. The total cost $C(\hat{T}) \leq C(T)$, every auxiliary edge in $\hat{T}$ is paid for by a single player, and the corresponding profile $\hat{P}$ is enforceable in $\hat{G}$.*

Proof of Theorem 5. The previous lemma shows that the algorithm computes a cost sharing of a tree $\hat{T}$ in $\hat{G}$, such that every player is happy with the path $\hat{P}_i$ and every auxiliary edge in $\hat{T}$ is paid for completely by a single player. We now transform $\hat{P}$ into P' by replacing each

auxiliary edge $e = (u, v) \in \hat{P}_i$ by the corresponding shortest path $P(u, v)$ in G. We denote by E_i the set of edges introduced in the shortest paths for auxiliary edges in $\hat{P}_i$. For the total cost of the resulting profile we have that $C(P') \leq C(\hat{P}) \leq C(P)$, since the sets E_i can overlap with each other or the non-auxiliary edges of $\hat{T}$.

We show that P' is enforceable by transforming the cost sharing constructed in function $\hat{c}$ into separable cost sharing functions as follows. Initially, set $\xi_{i,e}(P') = 0$ for all $e \in E$ and $i \in N$. Then, for each non-auxiliary edge $e \in \hat{T}$ we assign $\xi_{i,e}(P') = \hat{c}_e(i)$ if $e \in \hat{P}_i$ and $\xi_{i,e}(P') = 0$ otherwise. Finally, number players arbitrarily from 1 to n and proceed in that order. For player i, consider the edges in E_i. For every $e \in E_i$, if $\sum_{j<i} \xi_{j,e}(P') = 0$, then set $\xi_{i,e}(P') = c_e$.

This yields a budget-balanced assignment for state P'. As usual, if a player i deviates in P' from P'_i to P''_i, we can assume player i is assigned to pay the full cost c_e for every edge $e \in P''_i \setminus P'_i$. To show that there is no profitable deviation from P', we first consider a thought experiment, where every edge in E_i comes as a separate edge bought by player i. Then, clearly P' is enforceable – the cost of P'_i with ξ is exactly the same as the cost of $\hat{P}$ with $\hat{c}$ in $\hat{G}$. Moreover, any deviation P''_i can be interpreted as an (s, t_i)-path in $\hat{G}$ by replacing all subpaths consisting of non-auxiliary edges in P'' by the corresponding auxiliary edge of $\hat{G}$. As such, the cost of P''_i is exactly the same as the cost of the corresponding deviation in $\hat{G}$. Now, there is not a separate copy for every edge in E_i. The set E_i can overlap with other sets E_j and/or non-auxiliary edges. Then player i might not need to pay the full cost on some $e \in E_i$. Note, however, every edge for which player i pays less than c_e is present in P'_i as well. Hence, P''_i cannot improve over P'_i due to this property. ◀

The result continues to hold for various generalizations. For example, we can immediately apply the arguments in directed graphs, where every player i seeks to establish a directed path between t_i and s. Moreover, the proof can also be applied readily for a group-connection game, where each player wants to establish a directed path to s from *at least one node of a set* $V_i \subset V$. For this game, we simply add a separate super-terminal t_i for every player i and draw a directed edge of cost 0 from t_i to every node in V_i.

▶ **Corollary 7.** *Let P be a strategy profile for a single-source group-connection game in directed graphs with fixed costs. There is an enforceable profile P' with $C(P') \leq C(P)$ that can be computed by Algorithm 2 in polynomial time.*

5 Connection Games and Graph Structure

In this section, we consider connection games played in undirected graphs $G = (V, E)$ with player-specific source-terminal pairs. Each player $i \in N$ has a source-terminal-pair (s_i, t_i). Consequently, every strategy P_i of player i is an (s_i, t_i)-path in G. We denote the set of paths for player i by $\mathcal{P}_i$.

Note that we can assume w.l.o.g. that $(G, (s_1, t_1), \ldots, (s_n, t_n))$ is *irredundant*, meaning that each edge and each vertex of G is contained in at least one (s_i, t_i)-path for some player $i \in N$ (nodes and edges not used by any player can easily be recognized (and then deleted) by Algorithm Irredundant in [34]; adapted from Algorithm 1 in [18]).
For the special case without delays, enforceability was characterized via an LP in [35]. We

can directly adapt this characterization as follows. For a strategy profile $P = (P_1, \dots, P_n)$:

$$\text{LP}(P) \quad \max \sum_{i \in N, e \in P_i} \xi_{i,e} \text{ s.t.: } \sum_{i \in N_e(P)} \xi_{i,e} \leq c_e \forall e \in E \text{ with } N_e(P) \neq \emptyset,$$
$$\sum_{e \in P_i \setminus P_i'} (\xi_{i,e} + d_{i,e}) \leq \sum_{e \in P_i' \setminus P_i} (c_e + d_{i,e}) \ \forall P_i' \in \mathcal{P}_i \ \forall i \in N \qquad \text{(NE)}$$
$$\xi_{i,e} \geq 0 \ \forall e \in P_i \ \forall i \in N$$

▶ **Theorem 8.** *P is enforceable iff there is an optimal solution $(\xi_{i,e})_{i \in N, e \in P_i}$ for LP(P) with*
$$\textstyle\sum_{i \in N_e(P)} \xi_{i,e} = c_e \ \forall e \in E \text{ with } N_e(P) \neq \emptyset. \qquad \text{(BB)}$$

Given an optimal solution $(\xi_{i,e})_{i \in N, e \in P_i}$ for LP(P) with the property (BB), the profile P becomes a PNE in the game induced by ξ, which assigns for each $i \in N$ and $e \in E$ and each strategy profile $P' = (P_1', \dots, P_n')$ the following cost shares (these cost shares resemble those introduced in [53]; the proof that P is a PNE can be directly adapted from [35]):

$$\xi_{i,e}(P') = \begin{cases} \xi_{i,e}, & \text{if } i \in S_e(P') = S_e(P), \\ c_e, & \text{if } i \in (S_e(P') \setminus S_e(P)) \text{ and } i = \min(S_e(P') \setminus S_e(P)), \\ c_e, & \text{if } i \in S_e(P') \subsetneq S_e(P) \text{ and } i = \min S_e(P'), \\ 0, & \text{else.} \end{cases}$$

We now introduce a subclass of generalized series-parallel graphs for which we design a polynomial time black-box reduction.

▶ **Definition 9** (*n*-series-parallel graph)**.** An irredundant graph $(G, (s_1, t_1), \dots, (s_n, t_n))$ is *n-series-parallel* if, for all $i \in N$, the subgraph G_i (induced by $\mathcal{P}_i$) is created by a sequence of series and/or parallel operations starting from the edge $s_i - t_i$. For an edge $e = u - v$, a series operation replaces it by a new vertex w and two edges $u - w, w - v$; a parallel operation adds to $e = u - v$ a parallel edge $e' = u - v$.

▶ **Theorem 10.** *If $(G, (s_1, t_1), \dots, (s_n, t_n))$ is n-series-parallel, the following holds:*

(1) *Given an arbitrary strategy profile P, an enforceable strategy profile P' with cost $C(P') \leq C(P)$, and corresponding cost share functions ξ, can be computed in polynomial time.*

(2) *For all cost functions c, d, every optimal strategy profile of $(G, (s_1, t_1), \dots, (s_n, t_n), c, d)$ is enforceable.*

(3) *For all edge costs c, an optimal Steiner forest of $(G, (s_1, t_1), \dots, (s_n, t_n), c)$ can be computed in polynomial time.*

Proof Sketch for Theorem 10. We first describe how to compute, given an arbitrary profile $P = (P_1, \dots, P_n)$, an enforceable strategy profile with cost at most $C(P)$. Assume that P is not enforceable, and let $(\xi_{i,e})_{i \in N, e \in P_i}$ be an optimal solution for LP(P). In the following, we denote the variables $(\xi_{i,e})_{i \in N, e \in P_i}$ as *cost shares*, although they do not correspond to a budget-balanced cost sharing protocol (since P is not enforceable). There is at least one edge f which is not completely paid, i.e. $\sum_{i \in N_f(P)} \xi_{i,f} < c_f$ holds. The optimality of the cost shares implies that each player $i \in N_f(P)$ has an alternative path P_i' with $f \notin P_i'$ and equality in the corresponding LP(P)-inequality (so-called *tight alternative* of player i that substitutes f). Furthermore, if the path P_i of player i contains more than one edge which is not completely paid, there is a tight alternative P_i' for player i which substitutes all non-paid edges. Figure 1 illustrates this for the case that P_i (straight edges) contains the three non-paid edges f, g, h, and player i substitutes them by using P_i' (thick edges).

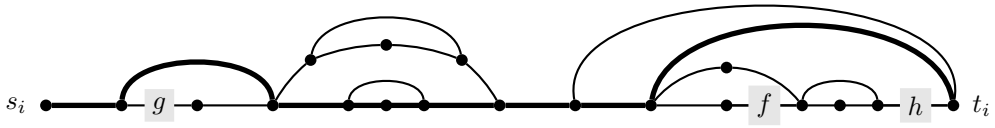


Figure 1 Illustration for substitution of nonpaid edges.

The high-level idea of the algorithm now is that all players with unpaid edges in their paths deviate from those edges by using tight alternatives until we reach a strategy profile in which all edges are completely paid. This strategy profile will then be enforceable and cheaper than P. In the following we explain this in more detail. Let $P' = (P'_1, \dots, P'_n)$ be the strategy profile which results from P if all players with unpaid edges in their paths substitute all these edges by a tight alternative path (as described above). Furthermore we define cost shares (again not nec. budget-balanced) for P' (for each player $i \in N, e \in P'_i$):

$$\xi_{i,e}(P') = \begin{cases} \xi_{i,e}, & \text{for } e \in P'_i \cap P_i, \\ c_e, & \text{for } e \in P'_i \setminus P_i. \end{cases}$$

If $\sum_{i \in N_e(P')} \xi_{i,e}(P') \geq c_e$ holds for all edges e with $N_e(P') \neq \emptyset$, the profile P' has the desired properties: The cost of P' is strictly smaller than P since the private costs of the players remain unchanged if they use tight alternatives. Furthermore the cost shares $(\xi_{i,e}(P'))_{i \in N, e \in P'_i}$ induce a feasible solution of $\mathrm{LP}(P')$ with (BB); thus P' is enforceable (we possibly need to decrease some cost shares to get a feasible solution of $\mathrm{LP}(P')$).

It remains to consider the case that there is at least one edge f which is not completely paid, i.e. for which $\sum_{i \in N_f(P')} \xi_{i,f}(P') < c_f$ holds. We can show that all users of nonpaid edges again have tight alternatives which substitute those edges, therefore we can again update the strategy profile (resulting in P'') and the corresponding cost shares. If now all edges are completely paid, P'' has the desired properties. Proceeding in this manner, we can show that one finally reaches a strategy profile for which all edges are completely paid; thus it is enforceable and cheaper than the profile P. Algorithm n-SePa in [34] summarizes the described procedure. To complete the proof of the first statement of Theorem 10, it remains to show that P' and ξ can be computed in polynomial time, i.e. Algorithm n-SePa has polynomial running time. This follows from two facts. First, the number of strategy profiles that we have to consider until we reach the desired strategy profile is bounded by $|P|$, the number of edges in the union of the paths $P_1, \dots, P_n$. Second, we can solve $\mathrm{LP}(P)$ in polynomial time. To this end we show that, for every player i, we do not need to consider all paths $P'_i \in \mathcal{P}_i$ in (NE) of $\mathrm{LP}(P)$, which can be exponentially many paths, but only a set of *alternatives* $\mathcal{A}_i$ of polynomial cardinality. Algorithm Alternatives(i) in [34] computes this set of alternatives. This completes the proof sketch for the first statement of Theorem 10.

Our analysis of the cost of P' immediately implies that every optimal strategy profile has to be enforceable: Otherwise Algorithm n-SePa computes a strategy profile with strictly smaller cost; contradiction. Thus the second statement of Theorem 10 holds.

The third statement follows from [7] where it is shown that an optimal Steiner forest can be computed in polynomial time if the underlying graph has treewidth at most 2. We show that every n-series-parallel graph is generalized series-parallel, and since these graphs have treewidth at most 2, the desired result follows. For a full proof of Theorem 10, see [34]. ◀

▶ Remark. The first two results of Theorem 10 can be generalized to nonnegative, nondecreasing and discrete-concave shareable edge cost functions. However, we do not know whether or not polynomial running time can be guaranteed.

We now demonstrate in Theorem 11 (for a proof see [34]) that the assumption of n-series-parallel graphs is in some sense well justified. Recall that a *generalized series-parallel graph* is created by a sequence of series, parallel, and/or add operations starting from a single edge, where an *add* operation adds a new vertex w and connects it to an existing vertex v by the edge $w - v$. The proofs of Theorem 10 and Theorem 11 show that the n-series-parallel graphs form a proper subclass of the generalized series-parallel graphs.

▶ **Theorem 11.** *For $n \geq 3$ players, there is a generalized series-parallel graph with fixed edge costs and no player-specific delays, so that the unique optimal Steiner forest is not enforceable. Therefore, a black-box reduction as for n-series-parallel graphs is impossible for generalized series-parallel graphs (even without player-specific delays).*

References

1 Nir Andelman, Michal Feldman, and Yishay Mansour. Strong price of anarchy. *Games Econom. Behav.*, 65(2):289–317, 2009.

2 Elliot Anshelevich and Bugra Caskurlu. Exact and approximate equilibria for optimal group network formation. *Theor. Comput. Sci.*, 412(39):5298–5314, 2011.

3 Elliot Anshelevich and Bugra Caskurlu. Price of stability in survivable network design. *Theory Comput. Syst.*, 49(1):98–138, 2011.

4 Elliot Anshelevich, Anirban Dasgupta, Jon Kleinberg, Tim Roughgarden, Éva Tardos, and Tom Wexler. The price of stability for network design with fair cost allocation. *SIAM J. Comput.*, 38(4):1602–1623, 2008. doi:10.1137/070680096.

5 Elliot Anshelevich, Anirban Dasgupta, Éva Tardos, and Tom Wexler. Near-optimal network design with selfish agents. *Theory of Computing*, 4(1):77–109, 2008.

6 Guy Avni and Tami Tamir. Cost-sharing scheduling games on restricted unrelated machines. *Theor. Comput. Sci.*, 646:26–39, 2016.

7 Mohammadhossein Bateni, Mohammadtaghi Hajiaghayi, and Dániel Marx. Approximation schemes for steiner forest on planar graphs and graphs of bounded treewidth. *JACM*, 58(5):21:1–21:37, 2011.

8 Vittorio Bilò, Angelo Fanelli, Michele Flammini, and Luca Moscardelli. When ignorance helps: Graphical multicast cost sharing games. *Theoretical Computer Science*, 411(3):660–671, 2010.

9 Vittorio Bilò, Michele Flammini, and Luca Moscardelli. The price of stability for undirected broadcast network design with fair cost allocation is constant. In *Proc. 54th Symp. Foundations of Computer Science (FOCS)*, pages 638–647, 2013.

10 C. Bird. On cost allocation for a spanning tree: A game theoretic approach. *Networks*, 6:335–350, 1976.

11 Jaroslaw Byrka and Karen Aardal. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. *SIAM Journal on Computing*, 39(6):2212–2231, 2010.

12 Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. Steiner tree approximation via iterative randomized rounding. *J. ACM*, 60(1):6:1–6:33, 2013.

13 Ioannis Caragiannis, Vasilis Gkatzelis, and Cosimo Vinci. Coordination mechanisms, cost-sharing, and approximation algorithms for scheduling. In Nikhil R. Devanur and Pinyan Lu, editors, *Web and Internet Economics*, pages 74–87, 2017.

14 Jean Cardinal and Martin Hoefer. Non-cooperative facility location and covering games. *Theor. Comput. Sci.*, 411:1855–1876, March 2010.

15 Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, and Sudipto Guha. Approximation algorithms for directed Steiner problems. *J. Algorithms*, 33(1):192–200, 1999.

16 Chandra Chekuri and Martin Pál. A recursive greedy algorithm for walks in directed graphs. In *Proc. 46th Symp. Foundations of Computer Science (FOCS)*, pages 245–253, 2005.

17 Ho-Lin Chen, Tim Roughgarden, and Gregory Valiant. Designing network protocols for good equilibria. *SIAM J. Comput.*, 39(5):1799–1832, 2010.

18 Xujin Chen, Zhuo Diao, and Xiaodong Hu. Network characterizations for excluding Braess's paradox. *Theory Comput. Syst.*, 59(4):747–780, 2016.

19 Giorgos Christodoulou, Vasilis Gkatzelis, and Alkmini Sgouritsa. Cost-sharing methods for scheduling games under uncertainty. In *Proc. 18th ACM Conf. Economics and Computation (EC)*, pages 441–458, 2017.

20 Giorgos Christodoulou, Stefano Leonardi, and Alkmini Sgouritsa. Designing cost-sharing methods for bayesian games. In *Proc. 9th Intl. Symp. Algorithmic Game Theory (SAGT)*, pages 327–339, 2016.

21 Giorgos Christodoulou and Alkmini Sgouritsa. Designing networks with good equilibria under uncertainty. In *Proc. 27th Symp. Discrete Algorithms (SODA)*, pages 72–89, 2016.

22 Xiaotie Deng, Toshihide Ibaraki, and Hiroshi Nagamochi. Algorithmic aspects of the core of combinatorial optimization games. *Math. Oper. Res.*, 24(3):751–766, 1999.

23 Yann Disser, Andreas Feldmann, Max Klimm, and Matús Mihalák. Improving the H_k-bound on the price of stability in undirected shapley network design games. *Theoret. Comput. Sci.*, 562:557–564, 2015.

24 Michal Feldman and Tami Tamir. Conflicting congestion effects in resource allocation games. *Oper. Res.*, 60(3):529–540, 2012.

25 Amos Fiat, Haim Kaplan, Meital Levy, Svetlana Olonetzky, and Ronen Shabo. On the price of stability for designing undirected networks with fair cost allocations. In *Proc. 33rd Intl. Coll. Automata, Languages and Programming (ICALP)*, volume 1, pages 608–618, 2006.

26 Naveen Garg, Goran Konjevod, and R Ravi. A polylogarithmic approximation algorithm for the Group Steiner tree problem. *J. Algorithms*, 37:66–84, 2000.

27 Konstantinos Georgiou and Chaitanya Swamy. Black-box reductions for cost-sharing mechanism design. In *Proc. 23rd Symp. Discrete Algorithms (SODA)*, pages 896–913, 2012.

28 Michel X. Goemans and Martin Skutella. Cooperative facility location games. *J. Algorithms*, 50(2):194–214, 2004.

29 Daniel Granot and Gur Huberman. On minimum cost spanning tree games. *Math. Prog.*, 21:1–18, 1981.

30 Daniel Granot and Michael Maschler. Spanning network games. *Internat. J. Game Theory*, 27:467–500, 1998.

31 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *J. Algorithms*, 31:228–248, 1999.

32 Thomas Dueholm Hansen and Orestis Telelis. On pure and (approximate) strong equilibria of facility location games. In Christos Papadimitriou and Shuzhong Zhang, editors, *Internet and Network Economics*, pages 490–497, 2008.

33 Thomas Dueholm Hansen and Orestis Telelis. Improved bounds for facility location games with fair cost allocation. In *Proc. 3rd Intl. Conf. Combinatorial Optimization and Applications (COCOA)*, pages 174–185, 2009.

34 Tobias Harks, Martin Hoefer, Anja Huber, and Manuel Surek. Efficient black-box reductions for separable cost sharing, 2018. `arXiv:1802.10351 [cs.GT]`.

35 Tobias Harks, Anja Huber, and Manuel Surek. A characterization of undirected graphs admitting optimal cost shares. In Nikhil R. Devanur and Pinyan Lu, editors, *Web and Internet Economics*, pages 237–251, Cham, 2017. Springer International Publishing.

36 Tobias Harks and Britta Peis. Resource buying games. *Algorithmica*, 70(3):493–512, 2014.

37 Tobias Harks and Philipp von Falkenhausen. Optimal cost sharing for capacitated facility location games. *European Journal of Operational Research*, 239(1):187–198, 2014.

38 Martin Hoefer. Non-cooperative tree creation. *Algorithmica*, 53:104–131, 2009.

39 Martin Hoefer. Competitive cost sharing with economies of scale. *Algorithmica*, 60:743–765, 2011.

40 Martin Hoefer. Strategic cooperation in cost sharing games. *Internat. J. Game Theory*, 42(1):29–53, 2013.

41 Kamal Jain and Vijay Vazirani. Applications of approximation algorithms to cooperative games. In *Proc. 33rd Symp. Theory of Computing (STOC)*, pages 364–372, 2001.

42 Jochen Könemann, Stefano Leonardi, Guido Schäfer, and Stefan van Zwam. A group-strategyproof cost sharing mechanism for the steiner forest game. *SIAM J. Comput.*, 37(5):1319–1341, 2008.

43 Euiwoong Lee and Katrina Ligett. Improved bounds on the price of stability in network cost sharing games. In *Proc. 14th Conf. Electronic Commerce (EC)*, pages 607–620, 2013.

44 Jian Li. An o(log(n)/log(log(n))) upper bound on the price of stability for undirected shapley network design games. *Inform. Process. Lett.*, 109(15):876–878, 2009.

45 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Inf. Comput.*, 222:45–58, 2013.

46 Nimrod Megiddo. Cost allocation for Steiner trees. *Networks*, 8(1):1–6, 1978.

47 Hervé Moulin and Scott Shenker. Strategyproof sharing of submodular costs: budget balance versus efficiency. *Econom. Theory*, 18(3):511–533, 2001.

48 Martin Pál and Éva Tardos. Group strategyproof mechanisms via primal-dual algorithms. In *FOCS*, pages 584–593, 2003.

49 Gabriel Robins and Alexander Zelikovsky. Tighter bounds for graph Steiner tree approximation. *SIAM J. Disc. Math.*, 19(1):122–134, 2005.

50 Robert Rosenthal. A class of games possessing pure-strategy Nash equilibria. *Internat. J. Game Theory*, 2:65–67, 1973.

51 Vasilis Syrgkanis. The complexity of equilibria in cost sharing games. In A. Saberi, editor, *Proc. 6th Internat. Workshop on Internet and Network Econom.*, LNCS, pages 366–377, 2010.

52 Arie Tamir. On the core of network synthesis games. *Math. Prog.*, 50:123–135, 1991.

53 Phlipp von Falkenhausen and Tobias Harks. Optimal cost sharing for resource selection games. *Math. Oper. Res.*, 38(1):184–208, 2013.

Price of Anarchy for Mechanisms with Risk-Averse Agents

Thomas Kesselheim
University of Bonn, Institute of Computer Science, Bonn, Germany
thomas.kesselheim@uni-bonn.de

Bojana Kodric
MPI for Informatics and Saarland University, Saarbrücken, Germany
bojana@mpi-inf.mpg.de

Abstract

We study the price of anarchy of mechanisms in the presence of risk-averse agents. Previous work has focused on agents with quasilinear utilities, possibly with a budget. Our model subsumes this as a special case but also captures that agents might be less sensitive to payments than in the risk-neutral model. We show that many positive price-of-anarchy results proved in the smoothness framework continue to hold in the more general risk-averse setting. A sufficient condition is that agents can never end up with negative quasilinear utility after playing an undominated strategy. This is true, e.g., for first-price and second-price auctions. For all-pay auctions, similar results do not hold: We show that there are Bayes-Nash equilibria with arbitrarily bad social welfare compared to the optimum.

2012 ACM Subject Classification Theory of computation → Algorithmic mechanism design, Theory of computation → Quality of equilibria

Keywords and phrases Mechanism Design, Price of Anarchy, Risk Aversion, Smoothness

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.155

Related Version A full version of the paper is available at [12], `https://arxiv.org/abs/1804.09468`.

Funding This work is supported by DFG through Cluster of Excellence MMCI.

1 Introduction

Many practical, "simple" auction mechanisms are not incentive compatible, making it beneficial for agents to behave strategically. A standard example is the first-price auction, in which one item is sold to one of n agents. Each of these agents is asked to report a valuation; the item is given to the agent reporting the highest value, who then has to pay what he/she reported. A common way to understand the effects of strategic behavior is to study resulting equilibria and to bound the *price of anarchy*. That is, one compares the social welfare that is achieved at the (worst) equilibrium of the induced game to the maximum possible welfare. Typical equilibrium concepts are Bayes-Nash equilibria and (coarse) correlated equilibria, which extend mixed Nash equilibria toward incomplete information or learning settings respectively. A key assumption in these analyses is that agents are *risk neutral*: Agents are assumed to maximize their expected quasilinear utility, which is defined to be the difference of the value associated to the outcome and payment imposed to the agent. So, an agent having a value of 1 for an item would be indifferent between getting this item with probability 10% for free and getting it all the time, paying 0.9.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 155; pp. 155:1–155:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

However, there are many reasons to believe that agents are not risk neutral. For instance, in the above example the agent might favor the certain outcome to the uncertain one. Therefore, in this paper, we ask the question: *What "simple" auction mechanisms preserve good performance guarantees in the presence of risk-averse agents?*

The standard model of risk aversion in economics (see, e.g., [16]) is to apply a (weakly) concave function to the quasilinear term. That is, if agent i's outcome is x_i and his payment is p_i, his utility is given as $u_i(x_i, p_i) = h_i(v_i(x_i) - p_i)$, where $h_i \colon \mathbb{R} \to \mathbb{R}$ is a weakly concave, monotone function. That is, for $y, y' \in \mathbb{R}$ and for all $\lambda \in [0, 1]$, it holds that $h_i(\lambda y + (1-\lambda)y') \geq \lambda h_i(y) + (1-\lambda)h_i(y')$. Agent i is risk neutral if and only if h_i is a linear function. If the function is strictly concave, this has the effect that, by Jensen's inequality, the utility for fixed x_i and p_i is higher than for a randomized x_i and p_i with the same expected $v_i(x_i) - p_i$.

We compare outcomes based on their social welfare, which is defined to be the sum of utilities of all involved parties including the auctioneer. That is, it is the sum of agents' utilities and their payments $\mathrm{SW}(\mathbf{x}, \mathbf{p}) = \sum_i u_i(x_i, p_i) + \sum_i p_i$. In the quasilinear setting this definition of social welfare coincides with the sum of values $\sum_i v_i(x_i)$. With risk-averse utilities they usually differ. However, all our results bound the sum of values and therefore also hold for this benchmark.

We assume that the mechanisms are oblivious to the h_i-functions and work like in the quasilinear model. Only the individual agent's perception changes. This makes it necessary to normalize the h_i-functions because otherwise they could be on different scales, e.g., if $h_1(y) = y$ and $h_2(y) = 1000 \cdot y$, which would be impossible for the mechanism to cope with without additional information. Therefore, we will assume that $u_i(\mathbf{x}, p_i) = v_i(\mathbf{x})$ if $p_i = 0$ and that $u_i(\mathbf{x}, p_i) = 0$ if $p_i = v_i$. That is, for the two cases that p_i is either 0 or the full value, the utility matches exactly the quasilinear one. However, due to risk aversion, the agents might be less sensitive to payments.[1]

1.1 Our Contribution

We give bounds on the price of anarchy for Bayes-Nash and (coarse) correlated equilibria of mechanisms in the presence of risk-averse agents. Our positive results are stated within the smoothness framework, which was introduced by [24]. We use the version that is tailored to quasilinear utilities by [26], which we extend to mechanism settings with general utilities (for a formal definition see Section 4). Our main positive result states that the loss of performance compared to the quasilinear setting is bounded by a constant if a slightly stronger smoothness condition is fulfilled.

▶ **Main Result 1.** *Given a mechanism with price of anarchy α in the quasilinear model provable via smoothness such that the deviation guarantees non-negative utility, then this mechanism has price of anarchy at most 2α in the risk-averse model.*

This result relies on the fact that the deviation action to establish smoothness guarantees agents non-negative utility. A sufficient condition is that all undominated strategies never have negative utility. First-price and second-price auctions satisfy this condition, we thus get constant price-of-anarchy bounds for both of these auction formats.

[1] We note here that this will not in turn allow the mechanism to arrive at huge utility gains, as compared to the quasilinear model, for example, by increasing payments arbitrarily. Indeed, Lemma 1 in Section 3 will show that the difference between the two optima is bounded by at most a multiplicative factor of 2.

In an all-pay auction every positive bid can lead to negative utility. Therefore, the positive result does not apply. As a matter of fact, this is not a coincidence because, as we show, equilibria can be arbitrarily bad.

▶ **Main Result 2.** *The single-item all-pay auction has unbounded price of anarchy for Bayes-Nash equilibria, even with only three agents.*

This means that although equilibria of first-price and all-pay auctions have very similar properties with quasilinear utilities, in the risk-averse setting they differ a lot. We feel that this to some extent matches the intuition that agents should be more reluctant to participate in an all-pay auction compared to a first-price auction.

In our construction for proving Main Result 2, we give a symmetric Bayes-Nash equilibrium for two agents. The equilibrium is designed in such a way that a third agent of much higher value would lose with some probability with every possible bid. Losing in an all-pay auction means that the agent has to pay without getting anything, resulting in negative utility. In the quasilinear setting, this negative contribution to the utility would be compensated by respective positive amounts when winning. For the risk-averse agent in our example, this is not true. Because of the risk of negative utility, he prefers to opt out of the auction entirely.

We also consider a different model of aversion to uncertainty, in which solution concepts are modified. Instead of evaluating a distribution over utilities in terms of their expectation, agents evaluate them based on the expectation minus a second-order term. We find that this model has entirely different consequences on the price of anarchy. For example, the all-pay auction has a constant price of anarchy in correlated and Bayes-Nash equilibria, whereas the second-price auction can have an unbounded price of anarchy in correlated equilibria.

1.2 Related Work

Studying the impact of risk-averseness is a regularly reoccurring theme in the literature. A proposal to distinguish between money and the utility of money, and to model risk aversion by a utility function that is concave first appeared in [1]. The expected utility theory, which basically states that the agent's behavior can be captured by a utility function and the agent behaves as a maximizer of the expectation of his utility, was postulated in [27]. This theory does not capture models that are standardly used in portfolio theory, "expectation minus variance" or "expectation minus standard deviation" [14], the latter of which we also consider in Section 7.

In the context of mechanisms, one usually models risk aversion by concave utility functions. One research direction in this area is to understand the effects of risk aversion on a given mechanism. For example, [6] studies symmetric equilibria in all-pay auctions with a homogenous population of risk-averse players. Due to symmetry and homogeneity, in this case, equilibria are fully efficient, that is, the price of anarchy is 1. In [18] a similar analysis for auctions with a buyout option is performed; [10] considers customers with heterogeneous risk attitudes in mechanisms for cloud resources. In [4] it is shown that for certain classes of mechanisms the correlated equilibrium is unique and has a specific structure depending on the respective valuations but independent of the actual utility function. One consequence of this result is that risk aversion does not influence the allocation outcome or the revenue.

Another direction is to design mechanisms for the risk-averse setting. For example, the optimal revenue is higher because buyers are less sensitive to payments. In a number of papers, mechanisms for revenue maximization are proposed [19, 17, 25, 11, 2, 7]. Furthermore, randomized mechanisms that are *truthful in expectation* lose their incentive properties if agents

are not risk neutral. Black-box transformations from truthful-in-expectation mechanisms into ones that fulfill stronger properties are given in [3] and [8].

Studying the effects of risk aversion also has a long history in game theory, where different models of agents' attitudes towards risk are analyzed. One major question is, for example, if equilibria still exist and if they can be computed [5, 9]. Price of anarchy analyses have so far only been carried out for congestion games. Tight bounds on the price of anarchy for atomic congestion games with affine cost functions under a range of risk-averse decision models are given in [23].

The smoothness framework was introduced by [24]. Among others, [26] tailored it to the quasilinear case of mechanisms. It is important to remark here that our approach is different from the one taken by [20]. They use the smoothness framework to prove generalized price of anarchy bounds for nonatomic congestion games in which players have biased utility functions. They assume that players are playing the "wrong game" and their point of comparison is the "true" optimal social welfare, meaning that the biases only determine the equilibira but do not affect the social welfare. We take the utility functions as they are, including the risk aversion, to evaluate social welfare in equilibria and also to determine the optimum, which makes our models incomparable.

For precise relation of von Neumann-Morgenstern preferences to mean-variance preferences, see for instance [15]. Mean-variance preferences were explored for congestion games in [21, 22], while [13] studies the bidding behavior in an all-pay auction depending on the level of variance-averseness.

2 Preliminaries

2.1 Setting

We consider the following setting: There is a set N of n players and $\mathcal{X}$ is the set of possible outcomes. Each player i has a utility function $u_i^{\theta_i}$, which is parameterized by her type $\theta_i \in \Theta_i$. Given a type θ_i, an outcome $\mathbf{x} \in \mathcal{X}$, and a payment $p_i \geq 0$, her utility is $u_i^{\theta_i}(\mathbf{x}, p_i)$. The traditionally most studied case are quasilinear utilities, in which types are valuation functions $v_i \in \mathcal{V}_i$, $v_i \colon \mathcal{X} \to \mathbb{R}$ and $u_i^{v_i}(\mathbf{x}, p_i) = v_i(\mathbf{x}) - p_i$. Throughout this paper, we will refer to quasilinear utilities by $\hat{u}_i^{v_i}$.

For fixed utility functions and types, the social welfare of an outcome $\mathbf{x} \in \mathcal{X}$ and payments $(p_i)_{i \in N}$ is defined as $\mathrm{SW}^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{p}) := \sum_{i \in N} u_i^{\theta_i}(\mathbf{x}, p_i) + \sum_{i \in N} p_i$. In the quasilinear case, this simplifies to $\sum_{i \in N} v_i(\mathbf{x})$. Unless noted otherwise, by $\mathrm{OPT}(\boldsymbol{\theta})$, we will refer to the optimal social welfare under type profile $\boldsymbol{\theta}$, i.e., $\max_{\mathbf{x},\mathbf{p}} \mathrm{SW}^{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{p})$.

A *mechanism* is a triple $(\mathcal{A}, X, P)$, where for each player i, there is a set of actions $\mathcal{A}_i$ and $\mathcal{A} = \times_i \mathcal{A}_i$ is the set of action profiles, $X \colon \mathcal{A} \to \mathcal{X}$ is an allocation function that maps actions to outcomes and $P \colon \mathcal{A} \to \mathbb{R}_+^n$ is a payment function that maps actions to payments p_i for each player i. Given an action profile $\mathbf{a} \in \mathcal{A}$, we will use the short-hand notation $u_i^{\theta_i}(\mathbf{a})$ to denote $u_i^{\theta_i}(X(\mathbf{a}), p_i)$.

2.2 Solution Concepts

In the setting of *complete information*, the type profile $\boldsymbol{\theta}$ is fixed. We consider (coarse) correlated equilibria, which generalize Nash equilibria and are the outcome of (no-regret) learning dynamics. A *correlated equilibrium (CE)* is a distribution $\mathbf{a}$ over action profiles from $\mathcal{A}$ such that for every player i and every strategy a_i in the support of $\mathbf{a}$ and every action

$a_i' \in \mathcal{A}_i$, player i does not benefit from switching to a_i' whenever he was playing a_i. Formally,

$$\mathbb{E}_{\mathbf{a}_{-i}|a_i}[u_i(\mathbf{a})] \geq \mathbb{E}_{\mathbf{a}_{-i}|a_i}[u_i(a_i', \mathbf{a}_{-i})], \forall a_i' \in \mathcal{A}_i, \forall i \ .$$

In *incomplete information*, the type of each player is drawn from a distribution F_i over her type space Θ_i. The distributions are common knowledge and the draws are independent among players. The solution concept we consider in this setting is the *Bayes-Nash Equilibrium*. Here, the strategy of each player is now a (possibly randomized) function $s_i \colon \Theta_i \to \mathcal{A}_i$. The equilibrium is a distribution over these functions s_i such that each player maximizes her expected utility conditional on her private information. Formally,

$$\mathbb{E}_{\boldsymbol{\theta}_{-i}|\theta_i}[u_i^{\theta_i}(\mathbf{s}(\boldsymbol{\theta}))] \geq \mathbb{E}_{\boldsymbol{\theta}_{-i}|\theta_i}[u_i^{\theta_i}(a_i, \mathbf{s}_{-i}(\boldsymbol{\theta}_{-i}))], \forall a_i \in \mathcal{A}_i, \forall \theta_i \in \Theta_i, \forall i \ .$$

The measure of efficiency is the expected social welfare over the types of the players: given a strategy profile $\mathbf{s} \colon \times_i \Theta_i \to \times_i \mathcal{A}_i$, we consider $\mathbb{E}_{\boldsymbol{\theta}}[\mathrm{SW}^{\boldsymbol{\theta}}(\mathbf{s}(\boldsymbol{\theta}))]$. We compare the efficiency of our solution concept with respect to the expected optimal social welfare $\mathbb{E}_{\boldsymbol{\theta}}[\mathrm{OPT}(\boldsymbol{\theta})]$.

The *price of anarchy (PoA)* with respect to an equilibrium concept is the worst possible ratio between the optimal expected welfare and the expected welfare at equilibrium, that is

$$\mathrm{PoA} = \max_{F} \max_{D \in EQ(F)} \frac{\mathbb{E}_{\boldsymbol{\theta} \sim F}[\mathrm{OPT}(\boldsymbol{\theta})]}{\mathbb{E}_{\boldsymbol{\theta} \sim F, \mathbf{a} \sim D}[\mathrm{SW}^{\boldsymbol{\theta}}(\mathbf{a})]} \ ,$$

where by $F = F_1 \times \cdots \times F_n$ we denote the product distribution of the players' type distributions and by $EQ(F)$ the set of all equilibria, which are probability distributions over action profiles.

We assume that players always have the possibility of not participating, hence any rational outcome has non-negative utility in expectation over the non-available information and the randomness of other players and the mechanism.

3 Modeling Risk Aversion

When modeling risk aversion, one wants to capture the fact that a random payoff (lottery) X is less preferred than a deterministic one of value $\mathbf{E}[X]$. The standard approach is, therefore, to apply a concave non-decreasing function $h \colon \mathbb{R} \to \mathbb{R}$ to X and consider $h(X)$ instead. By Jensen's inequality, we now know $\mathbf{E}[h(X)] \leq h(\mathbf{E}[X])$.

In the case of mechanism design, the utility of a risk-neutral agent is defined as the quasilinear utility $v_i(\mathbf{x}) - p_i$. That is, if an agent has a value of 1 for an item and has to pay 0.9 for it, then the resulting utility is 0.1. The expected utility is identical if the agent only gets the item with probability 0.1 for free. To capture the effect that the agent prefers the certain outcome to the uncertain one, we again apply a concave function $h_i \colon \mathbb{R} \to \mathbb{R}$ to the quasilinear term $v_i(\mathbf{x}) - p_i$. We then consider utility functions $u_i(\mathbf{x}, p_i) = h_i(v_i(\mathbf{x}) - p_i)$ in the setting described in Section 2. Note that the mechanisms we consider do not know the h_i-functions. They work as if all utility functions were quasilinear.

We want to compare outcomes based on their social welfare. We use the definition of social welfare being the sum of utilities of all involved parties including the auctioneer. That is, $\mathrm{SW}(\mathbf{x}, \mathbf{p}) = \sum_{i \in N} u_i(\mathbf{x}, p_i) + \sum_{i \in N} p_i$. It is impossible for any mechanism to choose good outcomes for this benchmark if the h_i-function are arbitrary and unknown. Therefore, we assume that utility functions are normalized so that the utility matches the quasilinear one for $p_i = 0$ and $p_i = v_i(\mathbf{x})$ (see Figure 1). In more detail, we assume the following *normalized risk-averse utilities*:

1. $u_i^{v_i}(\mathbf{x}, p_i) \geq u_i^{v_i}(\mathbf{x}, p_i')$ if $p_i \leq p_i'$ (monotonicity)
2. $u_i^{v_i}(\mathbf{x}, p_i) = 0$ if $p_i = v_i(\mathbf{x})$ (normalization at $p_i = v_i(\mathbf{x})$)

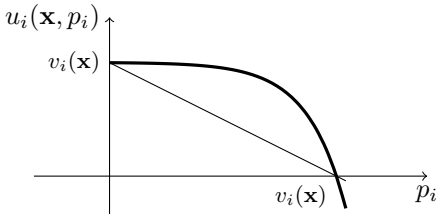


Figure 1 Normalized risk-averse utility function (bold) and quasilinear utility function for a fixed allocation $\mathbf{x}$ and varying payment p_i.

3. $u_i^{v_i}(\mathbf{x}, p_i) = v_i(\mathbf{x})$ if $p_i = 0$ (normalization at $p_i = 0$)
4. $u_i^{v_i}(\mathbf{x}, p_i) \geq v_i(\mathbf{x}) - p_i$ if $0 \leq p_i \leq v_i(\mathbf{x})$ and $u_i^{v_i}(\mathbf{x}, p_i) \leq v_i(\mathbf{x}) - p_i$ otherwise (relaxed concavity)

Assumption 4 is a relaxed version of concavity that suffices our needs for the positive results. Our negative results always fulfill concavity.

As an effect of normalization, the optimal social welfare of the risk-averse setting can be bounded in terms of the optimal sum of values, which coincides with the social welfare for quasilinear utilities.

▶ **Lemma 1.** *Given valuation functions $(v_i)_{i\in N}$ and normalized risk-averse utilities $(u_i^{v_i})_{i\in N}$, let OPT denote the optimal social welfare with respect to utilities $(u_i^{v_i})_{i\in N}$ and $\widehat{OPT}$ denote the optimal social welfare with respect to quasilinear utilities $(\hat{u}_i^{v_i})_{i\in N}$. Then, $OPT \leq 2\widehat{OPT}$.*

Proof. Let $\mathbf{x}$, $\mathbf{p}$ denote the outcome and payment profile that maximizes the social welfare $\sum_{i\in N} u_i^{v_i}(\mathbf{x}, p_i) + \sum_{i\in N} p_i$. Consider a fixed player i. If $0 \leq p_i \leq v_i(\mathbf{x})$, then by monotonicity of $u_i^{v_i}(\mathbf{x}, \cdot)$ and Assumption 3, $u_i^{v_i}(\mathbf{x}, p_i) + p_i \leq u_i^{v_i}(\mathbf{x}, 0) + p_i \leq 2v_i(\mathbf{x})$. If $p_i > v_i(\mathbf{x})$, then we know from Assumption 4 that $u_i^{v_i}(\mathbf{x}, p_i) + p_i \leq v_i(\mathbf{x})$. So, always, $u_i^{v_i}(\mathbf{x}, p_i) + p_i \leq 2v_i(\mathbf{x})$.

By taking the sum over all players, we get $\mathrm{OPT} = \sum_{i\in N} u_i^{v_i}(\mathbf{x}, p_i) + \sum_{i\in N} p_i \leq \sum_{i\in N} 2v_i(\mathbf{x}) \leq 2\widehat{\mathrm{OPT}}$. ◀

As a consequence, the optimal social welfare changes only within a factor of 2 by risk aversion and we may as well take $\widehat{\mathrm{OPT}}$ as our point of comparison. A VCG mechanism, for example, is still incentive compatible under risk-averse utilities but optimizes the wrong objective function. Lemma 1 shows that it is still a constant-factor approximation to optimal social welfare. However, in simple mechanisms, the agents' strategic behavior may or may not change drastically under risk aversion, depending on the mechanism. This way, equilbria and outcomes can possibly be very different.

4 Smoothness Beyond Quasilinear Utilities

Most of our positive results rely on the *smoothness* framework. It was introduced by [24] for general games. There are multiple adaptations to the quasilinear mechanism-design setting. We will use the one by [26]. As our utility functions will not be quasilinear, in this section we first observe that the framework can be extended to general utility functions. Note that throughout this section, the exact definition of $\mathrm{OPT}(\boldsymbol{\theta})$ is irrelevant. Therefore, it can be set to the optimal social welfare but also to weaker benchmarks depending on the setting.

▶ **Definition 2** (Smooth Mechanism)**.** A mechanism M is (λ, μ)-smooth with respect to utility functions $(u_i^{\theta_i})_{\theta_i\in\Theta_i, i\in N}$ for $\lambda, \mu \geq 0$, if for any type profile $\boldsymbol{\theta} \in \times_i \Theta_i$ and for any

action profile $\mathbf{a}$ there exists a randomized action $a_i^*(\boldsymbol{\theta}, a_i)$ for each player i, such that $\sum_i u_i^{\theta_i}(a_i^*(\boldsymbol{\theta}, a_i), \mathbf{a}_{-i}) \geq \lambda \mathrm{OPT}(\boldsymbol{\theta}) - \mu \sum_i p_i(\mathbf{a})$. We denote by $u_i^{\theta_i}(\mathbf{a})$ the expected utility of a player if $\mathbf{a}$ is a vector of randomized strategies.

Mechanism smoothness implies bounds on the price of anarchy. The following theorem and its proof are analogous to the theorems in [26], the proof is therefore deferred to the full version [12]. In cases where the deviation required by smoothness does not depend on a_i, the results extend to coarse correlated equilibria. The important point is that the respective bounds mostly do not depend on the assumption of quasilinearity.

▶ **Theorem 3.** *If a mechanism M is (λ, μ)-smooth w.r.t. utility functions $(u_i^{\theta_i})_{\theta_i \in \Theta_i, i \in N}$, then any Correlated Equilibrium in the full information setting and any Bayes-Nash Equilibrium in the Bayesian setting achieves efficiency of at least $\frac{\lambda}{\max\{1,\mu\}}$ of $OPT(\boldsymbol{\theta})$ or of $\mathbb{E}_{\boldsymbol{\theta}}[OPT(\boldsymbol{\theta})]$, respectively.*

In the standard single-item setting, one item is auctioned among n players, with their valuations and actions (bids) both being real numbers. In the common auction formats, the item is given to the bidder with the highest bid.

In a *first-price auction*, the winner has to pay her bid; the other players do not pay anything. It is $(1 - 1/e, 1)$-smooth w.r.t. quasilinear utility functions. In an *all-pay auction*, all players have to pay their bid. It is $(1/2, 1)$-smooth w.r.t. quasilinear utility functions. These smoothness results were given by [26]. They also show that simultaneous and sequential compositions of smooth mechanisms are again smooth.

What is remarkable here is that first-price and all-pay auctions achieve nearly the same welfare guarantees. We will show that in the risk-averse setting this is not true. While the first-price auction almost preserves its constant price of anarchy, the all-pay auction has an unbounded price of anarchy, even with only three players.

5 Quasilinear Smoothness Often Implies Risk-Averse Smoothness (Main Result 1)

Our main positive result is that many price-of-anarchy guarantees that are proved via smoothness in the quasilinear setting transfer to the risk-averse one. First, we consider mechanisms that are (λ, μ)-smooth with respect to quasilinear utility functions. We show that if the deviation strategy $\mathbf{a}^*$ that is used to establish smoothness ensures non-negative utility, then the price-of-anarchy bound extends to risk-averse settings at a multiplicative constant loss.

▶ **Theorem 4.** *If a mechanism is (λ, μ)-smooth w.r.t. quasilinear utility functions $(\hat{u}_i^{v_i})_{i \in N, v_i \in \mathcal{V}_i}$ and the actions in the support of the smoothness deviations satisfy $\hat{u}_i(a_i^*, \mathbf{a}_{-i}) \geq 0, \forall \mathbf{a}_{-i}, \forall i$, then any Correlated Equilibrium in the full information setting and any Bayes-Nash Equilibrium in the Bayesian setting achieves efficiency at least $\frac{\lambda}{2 \cdot \max\{1,\mu\}}$ of the expected optimal even in the presence of risk averse bidders.*

Using Theorem 3, it suffices to prove the following lemma.

▶ **Lemma 5.** *If a mechanism is (λ, μ)-smooth w.r.t. quasilinear utility functions $(\hat{u}_i^{v_i})_{i \in N, v_i \in \mathcal{V}_i}$ and the actions in the support of the smoothness deviations satisfy*

$$\hat{u}_i(a_i^*, \mathbf{a}_{-i}) \geq 0, \forall \mathbf{a}_{-i}, \forall i \ , \tag{1}$$

then the mechanism is $(\lambda/2, \mu)$-smooth with respect to any normalized risk-averse utility functions $(u_i^{v_i})_{i \in N, v_i \in \mathcal{V}_i}$.

Proof. We start from an arbitrary action profile $\mathbf{a}$ and want to satisfy Definition 2. Since there exist smoothness deviations s.t. $\hat{u}_i(a_i^*, \mathbf{a}_{-i}) = v_i(\mathbf{x}(a_i^*, \mathbf{a}_{-i})) - p_i \geq 0, \forall \mathbf{a}_{-i}, \forall i$, we know from property 4 of the risk aversion definition that $u_i^{v_i}(a_i^*, \mathbf{a}_{-i}) \geq \hat{u}_i^{v_i}(a_i^*, \mathbf{a}_{-i})$. Therefore,

$$\sum_i u_i^{v_i}(a_i^*, \mathbf{a}_{-i}) \geq \sum_i \hat{u}_i^{v_i}(a_i^*, \mathbf{a}_{-i}) \geq \lambda \widehat{\mathrm{OPT}} - \mu \sum_i p_i(\mathbf{a}) \geq \frac{\lambda}{2}\mathrm{OPT} - \mu \sum_i p_i(\mathbf{a}) \ ,$$

where the last inequality follows from Lemma 1. ◀

Note that in order for (1) to hold, it is sufficient if all undominated strategies guarantee non-negative quasilinear utility. For example, in a first-price auction, the only undominated bids are the ones from 0 to v_i. Regardless of the other players' bids, these can never result in negative utilities.

▶ **Corollary 6.** *Under normalized risk-averse utilities, the first-price auction has a constant price of anarchy for correlated and Bayes-Nash equilibria.*

We in addition note here that the first part of Property 4 of the normalization assumption, $u_i^{v_i}(\mathbf{x}, p_i) \geq v_i(\mathbf{x}) - p_i, 0 \leq p_i \leq v_i(\mathbf{x})$, is not crucial for obtaining a result similar to Theorem 4. Indeed, a relaxation of the form $u_i^{v_i}(\mathbf{x}, p_i) \geq C \cdot (v_i(\mathbf{x}) - p_i), 0 \leq p_i \leq v_i(\mathbf{x})$ for $0 < C < 1$, C constant, would incur a loss of at most a factor of C in the efficiency bound of Theorem 4. More details can be found in the full version [12].

For second-price auctions and their generalizations, for example, the just stated theorems do not suffice to prove guarantees on the quality of equilibria. One in addition needs a no-overbidding assumption and this is further taken care of in the framework of *weak smoothness*, also introduced in [26]. We defer all definitions and results that deal with weak smoothness, including the extension from quasilinear to general utility functions and risk-averse utilities yielding a constant loss as compared to the quasilinear case, to the full version [12].

We also consider the setting where players have hard *budget constraints*. Note that in this case the players' preferences are not quasilinear already in the risk neutral case. Informally, we show that if a mechanism is (λ, μ)-smooth w.r.t. quasilinear utility functions, then the loss of performance in the budgeted setting is bounded by a constant, even in the presence of risk-averse agents. All details can be found in the full version [12].

6 Unbounded Price of Anarchy for All-Pay Auctions (Main Result 2)

From the previous section, we infer that the constant price-of-anarchy bounds for first-price and second-price auctions immediately extend to the risk-averse setting. This is not true for all-pay auctions; by definition there is no non-trivial bid that always ensures non-negative utility. Indeed, as we show in this section, the price of anarchy is unbounded in the presence of risk-averse players. Missing calculation details can be found in the full version [12].

▶ **Theorem 7.** *In an all-pay auction with risk-averse players, the PoA is unbounded.*

The general idea is to construct a Bayes-Nash equilibrium with two players that very rarely have high values and only then bid high values. We then add a third player who always has a high value. However, as the first two players bid high values occasionally, there is no possible bid that ensures he will surely win. This means, any bid has a small probability of not getting the item but having to pay. Risk-averse players are more inclined to avoid this kind of lotteries. In particular, making our third player risk-averse enough, he prefers the sure zero utility of not participating to any way of bidding that always comes with a small probability of negative utility.

Proof of Theorem 7. We consider two (mildly) risk-averse players who both have the same valuation distributions and a third (very) risk-averse player with a constant value. For a large number $M > 5$, the first two players have values v_1 and v_2 drawn independently from distributions with density functions of value $2 \cdot (1 - (M-1) \cdot \varepsilon)$ on the interval $[1/2, 1)$ and value ε on the interval $[1, M]$, where $\varepsilon = 1/M^2$. The third player always has value $1/3 \cdot \ln(M/2)$ for winning.

We will construct a symmetric pure Bayes-Nash equilibrium involving only the first two players. It will be designed such that for the third player it is a best response to always bid 0, i.e., to opt out of the mechanism and never win the item. So, the combination of these strategies will be a pure Bayes-Nash equilibrium for all three players.

Note that the social welfare of any equilibrium of this form is upper-bounded by the optimal social welfare that can be achieved by the first two bidders. By Lemma 1, it is bounded by

$$\mathbb{E}[\mathrm{SW}] \leq 2 \cdot \mathbb{E}[\max\{v_1, v_2\}] \leq 2 \cdot \mathbb{E}[v_1 + v_2] = 2 \cdot (\mathbb{E}[v_1] + \mathbb{E}[v_2]) = 4 \cdot \mathbb{E}[v_1] \leq 4 \ .$$

Furthermore, the third player's value $v_3 = 1/3 \cdot \ln(M/2)$ is a lower bound to the optimal social welfare in the construction containing all three players. So, as pointwise $\mathrm{OPT}(v) \geq 1/3 \cdot \ln(M/2)$, where $v = (v_1, v_2, v_3) \in \mathcal{V}$ denotes the valuation profile, this implies that the price of anarchy can be arbitrarily high.

We define the utility functions by setting

$$u_i^{v_i}(b_i) = \begin{cases} \frac{h_i(v_i - b_i)}{h(v_i)} \cdot v_i & \text{if } b_i \text{ is the winning bid} \\ \frac{h_i(-b_i)}{h(v_i)} \cdot v_i & \text{otherwise} \end{cases} \tag{2}$$

For the first two players, we use $h_i(x) := 1 - e^{-x}$, $i \in \{1, 2\}$, so in particular increasing and concave. For the third player, we set $h_i(x) = x$ for $x \geq 0$ and $h_i(x) = C \cdot x$ for $x < 0$, where $C = (16 \cdot \frac{1}{3} \cdot \ln M/2) \cdot M^2 \geq 1$. Again this function is increasing and concave[2]. Note that the utility functions also satisfy normalizations at $p_i = v_i(\mathbf{x})$ and at $p_i = 0$. We see that in this example risk aversion has the effect of heavily penalizing payments without winning the auction.

▶ **Claim 8.** *With the third player not participating, it is a symmetric pure Bayes-Nash equilibrium for the first two players to play according to bidding function* $\beta \colon \mathcal{V}_i \to \mathbb{R}_+, i \in \{1, 2\}$, *such that*

$$\beta(x) = \int_{\frac{1}{2}}^{x} \frac{f(t)(e^t - 1)}{F(t) + (1 - F(t))e^t} dt \ , \tag{3}$$

where F *denotes the cumulative distribution function of the value and* f *denotes its density.*

Proof. We will argue that playing according to β is always the unique best response if the other player is playing according to β, too. Due to symmetry reasons, it is enough to argue about the first player.

Let us fix player 1's value $v_1 = x$ and consider the function $g \colon \mathbb{R}_{\geq 0} \to \mathbb{R}$ that is defined by $g(y) = \mathbf{E}[u_1^x(b_1 = y, b_2 = \beta(v_2), b_3 = 0)]$. We claim that g is indeed maximized at $y = \beta(x)$.

[2] Its slope is not an absolute constant. This is indeed necessary because the price of anarchy can be bounded in terms of the slopes of the h_i-functions as we show in the full version [12].

We have[3]

$$\begin{aligned} g(y) &= \Pr[\beta(v_2) \le y] \cdot \frac{h_1(x-y)}{h_1(x)} \cdot x + (1 - \Pr[\beta(v_2) \le y]) \cdot \frac{h_1(-y)}{h_1(x)} \cdot x \\ &= \frac{x}{h_1(x)} \left[F(\beta^{-1}(y)) \Big(h_1(x-y) - h_1(-y) \Big) + h_1(-y) \right] \\ &= x e^y F(\beta^{-1}(y)) + \frac{x(1-e^y)}{1-e^{-x}} \ . \end{aligned}$$

The first derivative of this function is given by

$$g'(y) = x e^y F(\beta^{-1}(y)) + x e^y \frac{d}{dy} F(\beta^{-1}(y)) - \frac{x}{1-e^{-x}} e^y \ .$$

The inverse function theorem implies $\frac{d}{dy} F(\beta^{-1}(y)) = \frac{f(\beta^{-1}(y))}{\beta'(\beta^{-1}(y))}$. Furthermore, as $\beta'(t) = \frac{f(t)(e^t-1)}{F(t)+(1-F(t))e^t}$, we get for all t that $\frac{f(t)}{\beta'(t)} = \frac{F(t)+(1-F(t))e^t}{e^t-1} = (1-F(t)) + \frac{1}{e^t-1}$. This simplifies $g'(y)$ to

$$g'(y) = x e^y + \frac{x e^y}{e^{\beta^{-1}(y)} - 1} - \frac{x e^y}{1-e^{-x}} = \frac{x e^y}{(1-e^{-x})(e^{\beta^{-1}(y)}-1)} \left(1 - e^{-x+\beta^{-1}(y)} \right) \ .$$

Notice that the factor $\frac{x e^y}{(1-e^{-x})(e^{\beta^{-1}(y)}-1)}$ is always positive. Therefore, we observe that $g'(y) = 0$ if and only if $e^{-x+\beta^{-1}(y)} = 1$, which is equivalent to $y = \beta(x)$. Furthermore, $g'(y) > 0$ for $y < \beta(x)$ and $g'(y) < 0$ for $y > \beta(x)$. This means that $y = \beta(x)$ has to be the (unique) global maximum of $g(y)$. ◀

▶ **Claim 9.** *If the first two players are bidding according to (3), then it is a best response for the third player to always bid* 0*.*

Proof. We now show that the very risk-averse third player with valuation $1/3 \cdot \ln(M/2)$ will indeed bid 0 because every bid $b_3' > 0$ will cause negative expected utility.

We distinguish two cases. For values of $b_3' > \frac{1}{16}$, we use that with a small probability one of the two remaining players has a valuation of at least $M-1$, which leads to negative utility. For $b_3' \le \frac{1}{16}$ on the other hand, he loses so often that his expected utility is again negative.

Let us first assume that the third player bids b_3' with $\frac{1}{16} < b_3' \le v_3$. In this case, with probability more than ε one of the first two players has value of at least $M-1$. The bid of this player with $v_i \ge M-1$ can be estimated as follows

$$\begin{aligned} \beta(v_i) \ge \beta(M-1) &\ge \int_{M/2}^{M-1} \frac{f(t)(e^t-1)}{1+(1-F(t))e^t} dt = \int_{M/2}^{M-1} \frac{\varepsilon(e^t-1)}{1+\varepsilon(M-t)e^t} dt \\ &\ge \frac{1}{2}(1-e^{-\frac{M}{2}}) \int_{M/2}^{M-1} \frac{\varepsilon e^t}{\varepsilon(M-t)e^t} dt = \frac{1}{2}(1-e^{-\frac{M}{2}}) \ln(M/2) > \frac{1}{3}\ln(M/2) \ , \end{aligned}$$

which means that by bidding b_3' the third player loses with probability of at least $\varepsilon = 1/M^2$. For the expected utility, this implies

$$\mathbb{E}[u_3(b_3', \mathbf{b}_{-3})] \le (1-\varepsilon)(v_3 - b_3') + \varepsilon(-C \cdot b_3') < \frac{1}{3}\ln\frac{M}{2} - \frac{1}{16} \cdot 16 \cdot \frac{1}{3}\ln\frac{M}{2} = 0 \ .$$

[3] Note that the first step assumes tie breaking in favor of player 1. This is irrelevant for the future steps as the involved probability distributions are continuous.

In the case where the third player bids b'_3, $0 < b'_3 \leq \frac{1}{16}$, we need to be a bit more careful with estimating the winning probability. We first give a lower bound on the bidding function of the first player for $v_1 < 1$

$$\beta(v_1) \geq \int_{1/2}^{v_1} \frac{2(1-\frac{M-1}{M^2})(e^t-1)}{2(t-\frac{1}{2})(1-\frac{M-1}{M^2})+(1-2(t-\frac{1}{2})(1-\frac{M-1}{M^2}))\cdot e^t} dt > \frac{1}{4}\left(v_1 - \frac{1}{2}\right) \ .$$

Since for $v_1 \geq 1$, $\beta(v_1) > \frac{1}{16}$ with probability 1, this implies that with b'_3, the third player has a winning probability of at most

$$\Pr[\beta(v_1) \leq b'_3] \leq \Pr\left[\frac{1}{4}\left(v_1 - \frac{1}{2}\right) \leq b'_3\right] = \Pr\left[v_1 \leq 4b'_3 + \frac{1}{2}\right] < 2 \cdot 4b'_3 \ .$$

Now, having in mind that $C = (16v_3) \cdot M^2 \geq 32 \cdot v_3$, the utility can be estimated as follows

$$\begin{aligned}\mathbb{E}[u_3(b'_3, \mathbf{b}_{-3})] &\leq \Pr[\beta(v_1) \leq b'_3] \cdot (v_3 - b'_3) - \Pr[\beta(v_1) > b'_3] \cdot 32 \cdot v_3 \cdot b'_3 \\ &< 8b'_3\left(-v_3 - \frac{1}{16}\right) < 0 \ .\end{aligned}$$

So also in this case, the expected utility is negative. ◀

Combining the two claims, we have constructed a class of distributions and Bayes-Nash equilibria with unbounded price of anarchy. ◀

As a final remark, we note that the first two bidders occasionally bid high only due to risk aversion. In a symmetric Bayes-Nash equilibrium of the all-pay auction in the quasilinear setting, all bids are always bounded by the expected value of a player (see full version [12]). Therefore, such an equilibrium would not work as a point of departure.

7 Variance-Aversion Model

In this section, we consider a different model that tries to capture the effect that agents prefer certain outcomes to uncertain ones. It is inspired by similar models in game theory and penalizes variance of random variables. Rather than reflecting the aversion in the utility functions, it is modeled by adapting the solution concept.

In the usual definition of equilibria involving randomization, the utility of a randomized strategy profile is set to be the expectation over the pure strategies. The definition we consider here is modified by subtracting the respective standard deviation. For a player i, the utility of a randomized strategy profile $\mathbf{a}$ is given as $u_i^{v_i}(\mathbf{a}) = \mathbb{E}_{\mathbf{b}\sim\mathbf{a}}[\hat{u}_i^{v_i}(\mathbf{b})] - \gamma\sqrt{\mathrm{Var}[\hat{u}_i^{v_i}(\mathbf{b})]}$, so a player's utility for an action profile is his expected quasilinear utility for this profile minus the standard deviation multiplied by a parameter γ that determines the degree of variance-averseness, $0 \leq \gamma \leq 1$. As already mentioned, $\hat{u}_i(\mathbf{a})$ denotes the quasilinear utility of player i for the action profile $\mathbf{a}$.

Bayes-Nash Equilibria and correlated equilibria can be defined the same way as before, always replacing expectations by the difference of expectation and standard deviation. The formal definition for $s(\mathbf{v})$ being a Bayes-Nash equilibrium in this setting is that $\forall i \in N$, $\forall v_i \in \Theta_i$, $a_i \in \mathcal{A}_i$,

$$\begin{aligned}&\mathbb{E}_{\mathbf{v}_{-i}}[\hat{u}_i^{v_i}(s(\mathbf{v})) \mid v_i] - \gamma\sqrt{\mathrm{Var}[\hat{u}_i^{v_i}(s(\mathbf{v})) \mid v_i]} \\ &\qquad \geq \mathbb{E}_{\mathbf{v}_{-i}}[\hat{u}_i^{v_i}(a_i, s_{-i}(\mathbf{v}_{-i})) \mid v_i] - \gamma\sqrt{\mathrm{Var}[\hat{u}_i^{v_i}(a_i, s_{-i}(\mathbf{v}_{-i})) \mid v_i]} \ .\end{aligned}$$

Note that we again evaluate social welfare as agents perceive it. That is, for a randomized strategy profile $\mathbf{a}$, we set $\mathrm{SW}^{\mathbf{v}}(\mathbf{a}) = \sum_i u_i^{v_i}(\mathbf{a}) + \sum_i p_i(\mathbf{a})$.

Our first result shows that first-price and notably also all-pay auctions have a constant price of anarchy in this setting. Note that even though the proof looks a lot like smoothness proofs, it is not possible to phrase it within the smoothness framework, since here we are dealing with a different solution concept.

▶ **Theorem 10.** *Bayes-Nash Equilibria and Correlated Equilibria of the first-price and all-pay auction have a constant price of anarchy in this model.*

Proof. For simplicity, we will show the claim only for Bayes-Nash equilibria. The proof for correlated equilibria works the same way with minor modifications to the notation.

Assume $\mathbf{b}$ is a Bayes-Nash equilibrium. We claim that $\mathbb{E}_{\mathbf{v}}[\mathrm{SW}^{\mathbf{v}}(\mathbf{b})] \geq \frac{1}{16} \cdot \mathbb{E}_{\mathbf{v}}[\mathrm{OPT}]$, where OPT denotes the value of social welfare in the allocation that maximizes it, i.e. maximized sum of utility and payments of the agents.

Consider a fixed player j and a fixed valuation v_j. Let $q = \Pr[\max_{i \neq j} b_i \leq \frac{1}{4} \cdot v_j]$ denote the probability that no other player's bid exceeds $\frac{1}{4} \cdot v_j$.

Assume first that $q \leq \frac{3}{4}$. Then, because the total social welfare lower bounded by the payments $\mathbb{E}_{\mathbf{v}_{-j}|v_j}[\mathrm{SW}^{\mathbf{v}}(\mathbf{b})] \geq (1-q)\frac{1}{4}v_j \geq \frac{1}{16}v_j$.

On the other hand, if $q \geq \frac{3}{4}$, since $\mathbb{E}_{\mathbf{v}_{-j}|v_j}[\mathrm{SW}^{\mathbf{v}}(\mathbf{b})] \geq \mathbb{E}_{\mathbf{v}_{-j}|v_j}\left[u_j^{v_j}(\frac{v_j}{4}, b_{-j})\right]$,

$$\mathbb{E}_{\mathbf{v}_{-j}|v_j}[\mathrm{SW}^{\mathbf{v}}(\mathbf{b})] \geq v_j q - \frac{1}{4}v_j - \gamma v_j \sqrt{q(1-q)} \geq \left(\frac{2-\gamma\sqrt{3}}{4}\right) v_j \geq \frac{1}{16}v_j \ ,$$

where the first inequality is in fact an equality for the all-pay auction.

From here, by taking the expectation over v_j and by weighing the right hand side by the probability that OPT takes a particular agent, the theorem follows. ◀

This is contrasted by a correlated equilibrium with 0 social welfare in a setting with positive values. Indeed, for the special case of $\lambda = 1$, we see that the variance-averse model further differs from the risk-averse model described in previous sections.

▶ **Observation 1.** *The PoA for CE of second price auctions is unbounded if $\gamma = 1$.*

The proof can be found in the full version [12]. This is not only a difference between smoothness and weak smoothness. Our final result is a mechanism that is (λ, μ)-smooth for constant λ and μ but has unbounded price of anarchy.

▶ **Theorem 11.** *For any constant $\gamma > 0$ there is a mechanism that is (λ, μ)-smooth with respect to quasilinear utility functions for constant λ and μ but has unbounded price of anarchy in the variance-aversion model.*

Proof. Consider a setting with two items and two players, who have unit-demand valuation functions such that $\frac{1}{c}v_{i,1} \leq v_{i,2} \leq cv_{i,1}$ for constant $c \geq 1$. The players' possible actions are to either report one of the two items as preferred or to opt out entirely. Our mechanism first assigns player 1 her (claimed) favorite item, then assigns player 2 the remaining one unless she opts out. There are no payments. Obviously, this mechanism is $(\frac{1}{c}, 0)$-smooth because the allocation is within a $\frac{1}{c}$-factor of the optimal allocation by construction.

We will now construct a mixed Nash equilibrium of bad welfare. To this end, let $v_{1,1} = v_{1,2} = \varepsilon$ for some small $\varepsilon > 0$. This makes player 1 indifferent between items 1 and 2. In particular, it is a best response to ask for item 1 with probability $\frac{q-1}{q}$ and for item 2 with

probability $\frac{1}{q}$. We note at this point that in a Bayes-Nash equilibrium we could make this respective action the unique best response by having random types.

For player 2, we set $v_{2,1} = c$, $v_{2,2} = 1$. She has the choice of participating or opting out. Opting out implies utility 0, whereas participating implies

$$u_2(\mathbf{a}) = \frac{c+q-1}{q} - \gamma\sqrt{\frac{(c-1)^2(q-1)}{q^2}} = \frac{(c-1)(1-\gamma\sqrt{q-1})}{q} + 1$$

Now, if we set $q = c-1$, then $u_2(\mathbf{a}) = 2 - \gamma\sqrt{c-2}$ which is negative for $c > \frac{4}{\gamma^2} + 2$. We further set $c = \frac{4}{\gamma^2} + 3$. That is, player 2 prefers to opt out. This outcome has social welfare ε whereas the optimal social welfare is c. ◀

Note that this last example shows that variance-averseness yields very strange preferences for lotteries. In our example, the variance-averse player prefers not to participate although any outcome in the (free) lottery has positive value.

References

1. Daniel Bernoulli. Exposition of a new theory on the measurement of risk. *Commentaries of the Imperial Academy of Science of Saint Petersburg*, 1738. Republished Econometrica 22(1), 23–36 (1954).
2. Anand Bhalgat, Tanmoy Chakraborty, and Sanjeev Khanna. Mechanism design for a risk averse seller. In *Proc. 8th Intl. Workshop Internet & Network Economics (WINE)*, pages 198–211, 2012.
3. Shaddin Dughmi and Yuval Peres. Mechanisms for risk averse agents, without loss. *CoRR*, abs/1206.2957, 2012.
4. Paul Dütting, Thomas Kesselheim, and Éva Tardos. Mechanism with unique learnable equilibria. In *Proc. 15th Conf. Econom. Comput. (EC)*, pages 877–894, 2014.
5. Amos Fiat and Christos H. Papadimitriou. When the players are not expectation maximizers. In *Proc. 3rd Intl. Symp. Algorithmic Game Theory (SAGT)*, pages 1–14, 2010.
6. Gadi Fibich, Arieh Gavious, and Aner Sela. All-pay auctions with risk-averse players. *Int. J. Game Theory*, 34(4):583–599, 2006.
7. Hu Fu, Jason Hartline, and Darrell Hoy. Prior-independent auctions for risk-averse agents. In *Proc. 14th Conf. Electr. Commerce (EC)*, pages 471–488. ACM, 2013.
8. Martin Hoefer, Thomas Kesselheim, and Berthold Vöcking. Truthfulness and stochastic dominance with monetary transfers. *ACM Trans. Econom. Comput.*, 4(2):11:1–11:18, 2016.
9. Darrell Hoy. The concavity of atomic splittable congestion games with non-linear utility functions. In *EC 2012, Workshop on Risk Aversion in Algorithmic Game Theory and Mechanism Design*, 2012.
10. Darrell Hoy, Nicole Immorlica, and Brendan Lucier. On-demand or spot? selling the cloud to risk-averse customers. In *Proc. 12th Intl. Conf. Web and Internet Economics (WINE)*, pages 73–86, 2016.
11. Audrey Hu, Steven A Matthews, and Liang Zou. Risk aversion and optimal reserve prices in first-and second-price auctions. *J. Econom. Theory*, 145(3):1188–1202, 2010.
12. Thomas Kesselheim and Bojana Kodric. Price of anarchy for mechanisms with risk-averse agents. *CoRR*, abs/1804.09468, 2018. URL: `http://arxiv.org/abs/1804.09468`.
13. Bettina S Klose and Paul Schweinzer. Auctioning risk: The all-pay auction under mean-variance preferences. Technical report, University of Zurich, 2014. `doi:10.5167/uzh-67043`.
14. Harry M Markowitz. *Portfolio selection: efficient diversification of investments*, volume 16. Yale university press, 1968.

15 Harry M. Markowitz. Mean-variance approximations to expected utility. *Europ. J. Oper. Res.*, 234(2):346–355, 2014.

16 Andreu Mas-Colell, Michael Dennis Whinston, Jerry R Green, et al. *Microeconomic theory*, volume 1. Oxford university press New York, 1995.

17 Eric Maskin and John Riley. Optimal auctions with risk averse buyers. *Econometrica*, pages 1473–1518, 1984.

18 Timothy Mathews and Brett Katzman. The role of varying risk attitudes in an auction with a buyout option. *J. Econom. Theory*, 27(3):597–613, 2006.

19 Steven A Matthews. Selling to risk averse buyers with unobservable tastes. *J. Econom. Theory*, 30(2):370–400, 1983.

20 Reshef Meir and David C. Parkes. Playing the wrong game: Smoothness bounds for congestion games with behavioral biases. *SIGMETRICS Performance Evaluation Review*, 43(3):67–70, 2015.

21 Evdokia Nikolova and Nicolás E. Stier Moses. A mean-risk model for the traffic assignment problem with stochastic travel times. *Oper. Res.*, 62(2):366–382, 2014.

22 Evdokia Nikolova and Nicolás E. Stier Moses. The burden of risk aversion in mean-risk selfish routing. In *Proc. 16th Conf. Econom. Comput. (EC)*, pages 489–506, 2015.

23 Georgios Piliouras, Evdokia Nikolova, and Jeff S. Shamma. Risk sensitivity of price of anarchy under uncertainty. *ACM Trans. Econom. Comput.*, 5(1):5:1–5:27, 2016.

24 Tim Roughgarden. Intrinsic robustness of the price of anarchy. In *Proc. 41st Symp. Theory of Computing (STOC)*, pages 513–522, 2009.

25 Mukund Sundararajan and Qiqi Yan. Robust mechanisms for risk-averse sellers. In *Proc. 11th Conf. Econom. Comput. (EC)*, pages 139–148, 2010.

26 Vasilis Syrgkanis and Éva Tardos. Composable and efficient mechanisms. In *Proc. 45th Symp. Theory of Computing (STOC)*, pages 211–220, 2013.

27 John von Neumann and Oskar Morgenstern. Theory of games and economic behavior. Princeton, New Jersey: Princeton University Press. XVIII, 625 p., 1944.

Polynomial Counting in Anonymous Dynamic Networks with Applications to Anonymous Dynamic Algebraic Computations

Dariusz R. Kowalski[1]
Computer Science Department, University of Liverpool, Liverpool, UK
D.Kowalski@liverpool.ac.uk

Miguel A. Mosteiro[2]
Computer Science Department, Pace University, New York, NY, USA
mmosteiro@pace.edu

Abstract

Starting with Michail, Chatzigiannakis, and Spirakis work [20], the problem of *Counting* the number of nodes in Anonymous Dynamic Networks has attracted a lot of attention. The problem is challenging because nodes are indistinguishable (they lack identifiers and execute the same program) and the topology may change arbitrarily from round to round of communication, as long as the network is connected in each round. The problem is central in distributed computing as the number of participants is frequently needed to make important decisions, such as termination, agreement, synchronization, and many others. A variety of algorithms built on top of *mass-distribution* techniques have been presented, analyzed, and also experimentally evaluated; some of them assumed additional knowledge of network characteristics, such as bounded degree or given upper bound on the network size. However, the question of whether Counting can be solved deterministically in sub-exponential time remained open. In this work, we answer this question positively by presenting Methodical Counting, which runs in polynomial time and requires no knowledge of network characteristics. Moreover, we also show how to extend Methodical Counting to compute the sum of input values and more complex functions without extra cost. Our analysis leverages previous work on random walks in evolving graphs, combined with carefully chosen alarms in the algorithm that control the process and its parameters. To the best of our knowledge, our Counting algorithm and its extensions to other algebraic and Boolean functions are the first that can be implemented in practice with worst-case guarantees.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

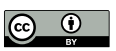
Keywords and phrases Anonymous Dynamic Networks, Counting, Boolean functions, distributed algorithms, deterministic algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.156

Related Version A full version of the paper is available at `https://arxiv.org/abs/1707.04282`.

Acknowledgements We thank Michal Koucký and Alessia Milani for useful discussions.

[1] Supported by Polish National Science Center grant UMO-2015/17/B/ST6/01897.
[2] Research work partially supported by Pace University Scholarly Research Award P1258 and the UK Royal Society Grant IES\R3\170293.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 156; pp. 156:1–156:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

How much information can one derive deterministically and distributedly from an arbitrarily evolving connected graph in polynomial time? Can we learn its size, or compute some simple Boolean functions, on its (distributed) input? In this work, we answer this question, posed in [20], in the affirmative. Specifically, we address first the problem of *Counting* the number of nodes of an Anonymous Dynamic Network (ADN) (and later extend to other algebraic functions) with a distributed and deterministic protocol. After a polynomial number of rounds of communication all nodes running our protocols obtain the result and stop. Our protocols resemble mass-distribution algorithms, however, our method is novel and quite complex as it has to deal with lack of node identities, lack of knowledge of network parameters, and adversarial topology changes.

The problem has been thoroughly studied [20, 10, 11, 12, 9, 21, 6] because Counting is central for distributed computing. Indeed, more complex tasks need the network size to make various decisions on state agreement, synchronization, termination, and others (e.g. [14, 15]). However, Anonymous Dynamic Networks pose a particularly challenging scenario. On one hand, nodes are indistinguishable from each other. For instance, they may lack identifiers or their number may be so massive that keeping record of them is not feasible. On the other hand, the topology of the network is highly dynamic. Indeed, the subsets of nodes that may communicate with each other may change all the time. All these features make ADN a valid model for anonymous ad hoc communication and computation.

In such a restrictive scenario, finding a way of providing theoretical guarantees of deterministic polynomial time has been elusive. Previous papers have either weaken the objective (e.g., computing only upper bound, only stochastic guarantees, etc.), assumed availability of network information (e.g., maximum number of neighbors, size upper bound, etc.), relied on a stronger model of communication, or provided only superpolynomial time guarantees.

Methodical Counting uses no information about the network. After completing its execution, all nodes obtain the exact size of the network and stop. Moreover, they stop all at the same time, allowing the algorithm to be concatenated with other computations.

Our algorithm is based on nodes continuously sharing some magnitude, which we call *potential*,[3] resembling *mass-distribution* and *push-pull* algorithms. Unlike previous algorithms, in Methodical Counting carefully and periodically (i.e. , "methodically") some potential is removed from the network, rather than greedily doing so continuously. This approach is combined with another methodological innovation testing whether the candidate value (for the network size) is within some polynomial range of the actual network size. This complex strategy yields an algorithm in which the progress in mass-distribution can be analyzed as a sequence of parametrized Markov chains (even though the algorithm itself is purely deterministic) enhanced by mass drift and alarms controlling the process and its parameters. Our analysis approach opens the path to study more complex tasks in Anonymous Dynamic Networks applying similar techniques.

Finally, we also present a variety of extensions of Methodical Counting to compute more complex functions. Most notably, we present an extension that, concurrently with finding the network size, computes the sum of input values held at each node without asymptotic time overhead. Having a method to compute the sum and network size, more complex computations are possible in polynomial time as well. Indeed, we also describe how to compute a variety of algebraic and Boolean functions.

[3] In previous related works this quantity, used in a different way, was termed *energy*. We steer away from such denomination to avoid confusion with node energy supply.

To the best of our knowledge, ours are the first algorithms for anonymous dynamic Counting and other algebraic computations that can be implemented in practice with worst-case polynomial-time guarantees.

Roadmap. The rest of the paper is organized as follows. Model and notation are detailed in Section 2. We overview previous work in Section 3 and present our results in Section 4. Section 5 includes the details of Methodical Counting, and we prove its correctness and running time in Section 6. Extensions to other functions are presented in Section 7.

2 Model, Problem, and Notation

The Counting Problem

The definition of the problem is simple. An algorithm solves the Counting Problem if, after completing its execution, all nodes have obtained the exact size of the network and stop.

Anonymous Dynamic Networks

The following model is customary in the Anonymous Dynamic Networks literature. We consider a network composed by a set V of $n > 1$ network *nodes* with processing and communication capabilities. It was shown in [20] that Counting cannot be solved in Anonymous Networks without the availability of at least one distinguished node in the network. Thus, we assume the presence of such node called *leader*. Aside from the leader, we assume that all other nodes are indistinguishable from each other. That is, we do not assume the availability of labels or identifiers, and all non-leader nodes execute exactly the same program.

Each pair of nodes that are able to communicate define a communication *link*, and the set of links is called the *topology* of the network. The nodes in a communication link are called *neighbors*. The event of sending a message to neighbors is called a *broadcast* or *transmission*. Nodes and links are reliable, in the sense that no communication or node failures occur. Hence, a broadcasted message is received by all neighbors. Moreover, links are *symmetric*, that is, if node a is able to send a message to node b, then b is able to send a message to a.

Without loss of generality, we discretize time in *rounds*. In any given round, a node may broadcast a message, receive all messages from broadcasting neighbors, and carry out some computations, in that order. Time needed for computations is assumed negligible.

The set of links among nodes may change from round to round, and nodes have no way of knowing which were the neighbors they had before. These topology changes are arbitrary, limited only to maintain the network connected in each round. That is, at any given round the topology is such that there is a *path*, i.e., a sequence of links, between each pair of nodes, but the set of links may change arbitrarily from round to round. This adversarial model of dynamics was called 1-*interval connectivity* in [19].

The following notation will be used. The maximum number of neighbors that any node may have at any given time, called the *dynamic maximum degree*, is denoted as Δ. An upper bound on Δ is denoted as $d_{\max}$. The maximum length of a shortest path between any pair of nodes at any given time is called the *dynamic diameter* and it is denoted as D. The maximum length of an opportunistic shortest path between any pair of nodes over many time slots is called the *chronopath* [13] and it is denoted as $\mathcal{D}$.

Table 1 Comparison of Counting protocols for Anonymous Dynamic Networks.

algorithm	needs		computes	stops?	complexity	
	size upper bound N	dynamic maximum degree u.b. $d_{\max}$			time	space
Degree Counting [20]		✓	$O(d_{\max}^n)$	✓	$O(n)$	
Conscious [10]	✓	✓	n	✓	$O(e^{N^2}N^3) \Rightarrow$ $O(e^{d_{\max}^{2n}} d_{\max}^{3n})$ using [20]	
Unconscious [10]			n	No	No theoretical bounds	
$\mathcal{A}_{\mathcal{O}^P}$ [11]		Oracle for each node	n	Eventually	Unknown	
EXT [9]			n	✓	$O(n^{n+4})$	EXPSPACE
Incremental Counting [21]		✓	n	✓	$O\left(n\,(2d_{\max})^{n+1}\,\frac{\ln n}{\ln d_{\max}}\right)$	
Methodical Counting [This work]			n	✓	$O(n^5 \ln^2 n)$	PSPACE

3 Previous Work

A comprehensive overview of work related to Anonymous Dynamic Networks can be found in a survey by Casteigts et al. [5] and references in the papers cited here. The directly related work overviewed in comparison with our results is summarized in Table 1.

With respect to lower bounds, it was proved in [8] that at least $\Omega(\log n)$ rounds are needed, even if D is constant. Also, $\Omega(\mathcal{D})$ is a lower bound since at least one node needs to hear about all other nodes to obtain the right count.

Counting and *Naming* was already studied in [20] for dynamic and static networks, showing that it is impossible to solve Counting without the presence of a distinguished node, even if the network is static. The Counting protocol requires knowledge of an upper bound on Δ, and obtains only an upper bound, which may be as bad as exponential.

Conscious Counting [10] computes the exact count, but it needs to start from an upper bound, and it takes exponential time only if the size upper bound is tight. In the same work and follow-up papers [11, 12], more challenging scenarios where Δ is unknown are studied, but protocols either do not terminate [10], or the protocol is terminated heuristically [12]. In experiments [12], such heuristic was found to perform well on dense topologies, but for other topologies the error rate was high. Another protocol in [11] is shown to terminate eventually, without running-time guarantees and under the assumption of having for each node an estimate of the number of neighbors in each round. In [20] it was conjectured that some knowledge of the network such as the latter would be necessary, but the conjecture was disproved later in [9]. On the other hand the protocol in [9] requires exponential space.

Incremental Counting, presented recently in [21], reduced exponentially the best-known running time guarantees. The protocol obtains the exact count, all nodes terminate simultaneously, the topology dynamics is only limited to 1-interval connectivity, it only requires polynomial space, and it only requires knowledge of of an upper bound ($d_{\max}$) on the dynamic maximum degree. The running time is still exponential, but reducing from doubly-exponential was an important step towards understanding the complexity of Counting.

In a follow-up paper [6], Incremental Counting was tested experimentally showing a promising polynomial behavior. The study was conducted on pessimistic inputs designed to slow the convergence, such as bounded-degree trees rooted at the leader uniformly chosen at random for each round, and a single path starting at the leader with non-leader nodes permuted uniformly at random for each round. The protocol was also tested on static versions of the inputs mentioned, classic random graphs, and networks where some disconnection is allowed. The results exposed important observations. Indeed, even for topologies that stretch the dynamic diameter, the running times obtained are below Δn^3. It was also observed that random graphs, as used in previous experimental studies [12], reduce the convergence time, and therefore are not a good choice to indicate worst-case behavior. These experiments showed good behavior even for networks that sometimes are disconnected, indicating that more relaxed models of dynamics, such as (α, β)-connectivity [13, 16], are worth to study. All in all, the experiments in [6] showed that Incremental Counting behaves well in a variety of pessimistic inputs, but not having a proof of what a worst-case input looks like, and being the experiments restricted to a range of values of n far from the expected massive size of an Anonymous Dynamic Network, a theoretical proof of polynomial time remained an open problem even from a practical perspective.

In a recent manuscript [2] a polynomial Counting algorithm is presented relying on the availability of an algorithm to compute average with polynomial convergence time. Such average computation is modeled as a Markov chain with underlying doubly-stochastic matrix, which requires topology information within two hops (cf. [23]). In our model of Anonymous Dynamic Network, such information is not available, and gathering it may not be possible due to possible topology changes from round to round.

Other studies also dealing with the time complexity of information gathering exist [7, 3, 24, 4, 22], but include in their model additional assumptions, such as the network having the same topology frequently enough.

4 Our Contributions

We present a deterministic distributed algorithm, which we call Methodical Counting, to compute the number of nodes in an Anonymous Dynamic Network. As opposed to previous works, our algorithm does not require any knowledge of network characteristics, such as dynamic maximum degree or an upper bound on the size. After $O(n^5 \ln^2 n)$ communication rounds of running Methodical Counting, all nodes obtain the network size and stop at the same round. To the best of our knowledge, this is the first polynomial deterministic Counting algorithm in the pure model of Anonymous Dynamic Network.

Our algorithm distributes potential in a mass-distribution fashion resembling previous works for Counting. The main novelty in our approach is that the leader participates in the process as any other node, removing potential only after it has accumulated enough. This approach allowed us to leverage previous work on random walks in evolving graphs. For this approach to work, we combine it with testing whether the candidate value for the network size is polynomially close to the actual value. Our approach also opens the path to study more complex computations in Anonymous Dynamic Networks using the same analysis.

Finally, we also present extensions of Methodical Counting to compute more complex functions. Most notably, we show how to modify Methodical Counting to compute the sum of input values held by nodes at the same time than counting. Having an algorithm to compute the network size and the sum of input values, we also show how to compute other algebraic and Boolean functions.

Algorithm 1 METHODICAL COUNTING algorithm for the leader. N is the set of neighbors of the leader in the current round. The parameters d, p, r and τ are as defined in Theorem 6.

```
1:  procedure COUNT
2:      ρ ← 0                                          // accumulator of consumed potential
3:      Φ ← 0                                          // current potential
4:      k ← 2                                          // current estimate
5:      status ← normal                                // status=normal|alarm|done
6:      while status ≠ done do                         // iterating epochs
7:          for phase = 1 to p do                      // iterating phases
8:              for round = 1 to r do                  // iterating rounds
9:                  Broadcast ⟨Φ, status⟩ and Receive ⟨Φ_i, status_i⟩, ∀i ∈ N
10:                 if status = normal and |N| ≤ d−1 and ∀i ∈ N : status_i = normal then
11:                     Φ ← Φ + Σ_{i∈N} Φ_i/d − |N|Φ/d     // update potential
12:                 else                                   // k is wrong
13:                     status ← alarm
14:                     Φ ← 1
                /* r rounds completed */
15:             if phase = 1 and Φ > τ then                // k is wrong
16:                 status ← alarm
17:                 Φ ← 1
18:             if status = normal then                    // prepare for next phase
19:                 ρ ← ρ + Φ
20:                 Φ ← 0
            /* p phases completed */
21:         if status = normal and k − 1 − 1/k ≤ ρ ≤ k − 1 then   // the size is k
22:             status ← done
23:         else                                           // prepare for next epoch
24:             ρ ← 0
25:             Φ ← 0
26:             k ← k + 1
27:             status ← normal
28:         for round = 1 to k do                          // disseminate termination
29:             Broadcast ⟨status⟩ and Receive ⟨status_i⟩, ∀i ∈ N
            /* epoch completed */
30:     return k
```

5 Methodical Counting

In this section we present METHODICAL COUNTING. First, we give the intuition of the algorithm, the details can be found in Algorithms 1 and 2. (References to algorithm lines are given as $\langle algorithm\#\rangle.\langle line\#\rangle$.)

Initially, the leader is assigned a potential of 0 and all the other nodes are assigned a potential of 1. Then, the algorithm is composed by epochs, each of which is divided into phases composed by rounds of communication. Epoch k corresponds to a size estimate k that is iteratively increased from epoch to epoch until the correct value n is found. Each epoch is divided into p phases. The purpose of each phase is for the leader to collect as much potential as possible from the other nodes in a mass-distribution fashion as follows.

Algorithm 2 METHODICAL COUNTING algorithm for each non-leader node i. N is the set of neighbors of i in the current round. The parameters d, p, r and τ are as defined in Theorem 6.

```
1:  procedure COUNT
2:      Φ ← 1                                                    // current potential
3:      k ← 2                                                    // current estimate
4:      status ← normal                                          // status=normal|alarm|done
5:      while status ≠ done do                                   // iterating epochs
6:          for phase = 1 to p do                                // iterating phases
7:              for round = 1 to r do                            // iterating rounds
8:                  Broadcast ⟨Φ, status⟩ and Receive ⟨Φ_i, status_i⟩, ∀i ∈ N
9:                  if status = normal and |N| ≤ d−1 and ∀i ∈ N : status_i = normal then
10:                     Φ ← Φ + Σ_{i∈N} Φ_i/d − |N|Φ/d           // update potential
11:                 else                                         // k is wrong
12:                     status ← alarm
13:                     Φ ← 1
               /* r rounds completed */
14:             if phase = 1 and Φ > τ then                      // k is wrong
15:                 status ← alarm
16:                 Φ ← 1
           /* p phases completed */
17:         for round = 1 to k do                                // disseminate termination
18:             Broadcast ⟨status⟩ and Receive ⟨status_i⟩, ∀i ∈ N
19:             if ∃i ∈ N : status_i = done then
20:                 status ← done
21:         if status ≠ done then
22:             k ← k + 1
23:             status ← normal
           /* epoch completed */
24:     return k
```

Each phase is composed by r rounds of communication. In each round, each node[4] broadcasts its potential and receives the potential of all its neighbors. Each node keeps only a fraction $1/d$ of the potentials received. The parameters p, r, and d are functions of k. The specific functions needed to guarantee correctness and saught efficiency are defined in Theorem 6. This varying way of distributing potential is different from previous approaches using mass distribution. After communication, each node updates its own potential accordingly (cf. Lines 1.11 and 2.10). That is, it adds a fraction $1/d$ of the potentials received, and subtracts a fraction $1/d$ of the potential broadcasted times the number of potentials received. Then, a new round starts.

At the end of each phase the leader "consumes" its potential. That is, it increases an internal accumulator ρ with its current potential, which is zeroed for starting the next phase (cf. Lines 1.19 and 1.20). A node stops the update of potential described, raises its potential to 1, and broadcasts an alarm in each round until the end of the epoch if any of the following happens: 1) at the end of the first phase its potential is above some threshold τ as defined in

[4] As opposed to previous work, in METHODICAL COUNTING the leader also follows this procedure.

Theorem 6 (cf. Lines 1.15 and 2.14), 2) at any round it receives more than $d-1$ messages (cf. Lines 1.12 and 2.11), or 3) at any round it receives an alarm (cf. Lines 1.12 and 2.11). The alarm for case 1) allows the leader to detect that the estimate is wrong when $k^{1+\epsilon} < n$ for some $\epsilon > 0$ (Lemmas 4 and 5), the alarm for case 2) allows the leader to detect that d is too small and hence the estimate is wrong, and the alarm for case 3) allows dissemination of all alarms. In the alarm status the potential is set to 1 to facilitate the analysis, but it is not strictly needed by the algorithm.

At the end of each epoch, the leader checks the value of ρ. If $k - 1 - 1/k \leq \rho \leq k - 1$ the current estimate is correct and the leader changes its status to "done" (cf. Line 1.21). Otherwise, all its variables are reset to start a new epoch with the next estimate (cf. Line 1.23). Before starting a new epoch the network is flooded with the status of the leader for k rounds (cf. Lines 1.28 and 2.17). If $k = n$, the leader initiates message "done" and the k rounds are enough for all the nodes to receive the "done" status and after completing the k rounds stop. Otherwise, nodes will not receive the "done" status and after completing the k rounds they start a new epoch.

6 Analysis

In this section we analyze Methodical Counting. References to algorithm lines are given as $\langle algorithm\# \rangle.\langle line\# \rangle$. We use standard notations $\mathbf{I}$ for the unit vector, and L_p for the norm of vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ as $||\mathbf{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$, for any $p \geq 1$. Only for the analysis, nodes are labeled as $0, 1, 2, \ldots, n-1$, where the leader has label 0. The potential of a node i at the beginning of round s of phase t is denoted as $\Phi_{s,t}[i]$, the potential of all nodes is denoted as a vector $\mathbf{\Phi}_{s,t}$, and the aggregated potential is then $||\mathbf{\Phi}_{s,t}||_1$. The subindices s, t, or both are omitted sometimes for clarity. We will refer to the potential right after the last round of a phase as $\mathbf{\Phi}_{r+1}$. Round $r+1$ does not exist in the algorithm, but we use this notation to distinguish between the potential right before the leader consumes its own potential (cf. Line 1.23) and the potential at the beginning of the first round of the next phase.

First, we provide a broad description of our analysis of Methodical Counting. Consider the vector of potentials $\mathbf{\Phi}_i$ held by nodes at the beginning of any given phase i. The way that potentials are updated in each round (cf. Lines 1.11 and 2.10) is equivalent to the progression of a d-lazy random walk on the evolving graph underlying the network topology [1], where the initial vector of potentials is equivalent to an initial distribution $\mathbf{\Pi}_i$ on the overall potential $||\mathbf{\Phi}_i||_1$ and the probability of choosing a specific neighbor is $1/d$. For instance, the initial vector of potentials $\mathbf{\Phi}_0 = \langle 0, 1, 1, \ldots \rangle$, corresponds to a distribution $\mathbf{\Pi}_0 = \langle 0, 1/(n-1), 1/(n-1), \ldots \rangle$ on the initial $||\mathbf{\Phi}_0||_1 = n - 1$.

Note that our Methodical Counting is not a simple "derandomization" of the lazy random walk on evolving graphs. First, in the Anonymous Dynamic Network model neighbors cannot be distinguished, and even their number is unknown at transmission time (only at receiving time the node learns the number of its neighbors). Second, due to unknown network parameters, it may happen in an execution of Methodical Counting that the total potential received could be bigger than 1. Third, our algorithm does not know a priori when to terminate and provide result even with some reasonable accuracy, as the formulas on mixing and cover time of lazy random walks depend on (a priori unknown) number of nodes n. Nevertheless, we can still use some results obtained in the context of analogous lazy random walks in order to prove useful properties of parts of algorithm Methodical Counting, namely, some parts in which parameters are temporarily fixed and the number of received messages does not exceed parameter d.

It was shown in [1] that random walks on d-regular explorable evolving graphs have a uniform stationary distribution, and bounds on the mixing and cover time were proved as well. Moreover, it was observed that those properties hold even if the graph is not regular and d is only an upper bound on the degree.[5]

Thus, for the cases where d is an upper bound on the number of neighboring nodes, we analyze the evolution of potentials within each phase leveraging previous work on random walks on evolving graphs. Specifically, we use the following result which is an extension of Corollary 14 in [1].

▶ **Theorem 1.** *(Corollary 14 in [1].) After t rounds of a $d_{\max}$-lazy random walk on an evolving graph with n nodes, dynamic diameter D, upper bound on maximum degree $d_{\max}$, and initial distribution $\mathbf{\Pi}_0$, the following holds:*

$$\left|\left|\mathbf{\Pi}_t - \frac{\mathbf{I}}{n}\right|\right|_2^2 \leq \left(1 - \frac{1}{d_{\max} D n}\right)^t \left|\left|\mathbf{\Pi}_0 - \frac{\mathbf{I}}{n}\right|\right|_2^2 .$$

In between phases the leader "consumes" its potential, effectively changing the distribution at that point. Then, a new phase starts.

In Methodical Counting, given that d is a function of the estimate k, if the estimate is low, there may be inputs for which d is not an upper bound on the number of neighbors. We show in our analysis that in those cases the leader detects the error and after some time all nodes increase the estimate.

First, we prove correctness when $k = n$ in the following lemma. The proof, left to the full version of this paper, is based on upper bounding the potential left in the system after running the algorithm.

▶ **Lemma 2.** *If $d \geq k$ and $k = n$, after running the* Methodical Counting *protocol for $p \geq \frac{k}{1-1/k} \ln(k(k-1))$ phases, each of $r \geq 4dk^2 \ln k$ rounds, the potential ρ consumed by the leader is $k - 1 - 1/k \leq \rho \leq k - 1$.*

Lemma 2 shows that if $\rho > k-1$ or $\rho < k-1-1/k$ we know that the estimate k is wrong, but the complementary case, that is, $k-1-1/k \leq \rho \leq k-1$, may occur even if the estimate is $k < n$ and hence the error has to be detected by other means. To prove correctness in that case, we show first that if $k < n \leq k^{1+\epsilon}$ for some $\epsilon > 0$ the leader must consume $\rho > k-1$ potential if the protocol is run long enough. To ensure that $d \geq \Delta + 1$, we restrict $d \geq k^{1+\epsilon}$. The proof of the following lemma, based again on upper bounding the potential left in the system after running the algorithm, is left to the full version of this paper.

▶ **Lemma 3.** *If $1 < k < n \leq k^{1+\epsilon} \leq d$, $\epsilon > 0$, after running the* Methodical Counting *protocol for $p \geq \frac{(2+\epsilon)k^{1+\epsilon}}{1-1/k} \ln k$ phases, each of $r \geq (4+2\epsilon)dk^{2+2\epsilon} \ln k$ rounds, the potential ρ consumed by the leader is $\rho > k-1$.*

It remains to show that even if $n > k^{1+\epsilon}$ Methodical Counting still detects that the estimate is low. First, we prove the following two claims that establish properties of the potential during the execution of Methodical Counting. (Recall that we use round $r+1$ to refer to potentials at the end of the phase right before the leader consumes its potential in Line 1.23.) The proofs of both claims, based on observing the mass-distribution properties and the alarms in the algorithm, are left to the full version of this paper.

[5] Their analysis relies on Lemma 12, which bounds the eigenvalues of the transition matrix as long as it is stochastic, connected, symmetric, and non-zero entries lower bounded by $1/d$. Those conditions hold for all the transition matrices, even if the evolving graph is not regular.

▶ **Claim 1.** *Given an Anonymous Dynamic Network of n nodes running* Methodical Counting *with parameter d, for any round t of the first phase, such that $1 \leq t \leq r+1$, if d was larger than the number of neighbors of each node x for every round $t' < t$, then $||\mathbf{\Phi}_{1,t}||_1 = n-1$.*

▶ **Claim 2.** *Given an Anonymous Dynamic Network of n nodes running* Methodical Counting*, for any round t of any phase and any node x, it is $0 \leq \Phi_t[x] \leq 1$.*

It remains to show that even if $n > k^{1+\epsilon}$ Methodical Counting still detects that the estimate is low. We focus on the first phase. We define a threshold τ such that, after the phase is completed, all nodes that have potential above τ can send an alarm to the leader, as such potential indicates that the estimate is low. We show that the alarm must be received after $k^{1+\epsilon}$ further rounds of communication in the following lemma. The proof, based on bounding the "room" that nodes have up to the maximum potential, is left to the full version of this paper.

▶ **Lemma 4.** *For $\epsilon > 0$, after running the first phase of the* Methodical Counting *protocol, there are at most $k^{1+\epsilon}$ nodes that have potential at most $\tau = 1 - 1/k^{1+\epsilon}$.*

In our last lemma, we show that if $k^{1+\epsilon} < n$ the leader detects the error. The proof, based on bounding the number of nodes with low potential at the end of the first phase, is left to the full version of this paper.

▶ **Lemma 5.** *If $k^{1+\epsilon} < n$, $\epsilon > 0$, and $r \geq (4 + 2\epsilon - 2\ln(k^\epsilon - 1)/\ln k)dk^2 \ln k$, within the following $k^{1+\epsilon}$ rounds after the first phase of the* Methodical Counting *protocol, the leader has received an alarm message.*

Based on the above lemmata, we establish our main result in the following theorem.

▶ **Theorem 6.** *Given an Anonymous Dynamic Network with n nodes, after running* Methodical Counting *for each estimate $k = 2, 3, \ldots, n$ with parameters*

$$
\begin{aligned}
d &= k^{1+\epsilon}, \\
p &= \left\lceil \frac{(2+\epsilon)k^{1+\epsilon}}{1-1/k} \ln k \right\rceil, \\
r &= \left\lceil \left(4 + 2\epsilon + \max\left\{0, -\frac{2\ln(k^\epsilon - 1)}{\ln k}\right\}\right) dk^{2+2\epsilon} \ln k \right\rceil, \\
\tau &= 1 - 1/k^{1+\epsilon},
\end{aligned}
$$

where $\epsilon > 0$, all nodes stop after $\sum_{k=2}^{n}(pr + k)$ rounds of communication and output n.

Proof. Notice that the above parameters fulfill the conditions of the previous lemmas. First, we prove that Methodical Counting is correct. To do so, it is enough to show that for each estimate $k < n$ the algorithm detects the error and moves to the next estimate, and that if otherwise $k = n$ the algorithm stops and outputs k. We consider three cases: $k = n$, $k < n \leq k^{1+\epsilon}$, and $k^{1+\epsilon} < n$, for a chosen value of $\epsilon > 0$.

Assume first that $k < n \leq k^{1+\epsilon}$. Then, even if the leader does not receive an alarm during the execution, as shown in Lemma 3, at the end of the epoch in Line 1.21 the leader will detect that ρ is out of range and will not change its status to done. Therefore, no other node will receive a termination message (loop in Line 1.28), and all nodes will continue to the next epoch.

Assume now that $k^{1+\epsilon} < n$. Lemma 5 shows that within the following $k^{1+\epsilon}$ rounds after the first phase the leader has received an alarm message, even if no node has more than $d-1$ neighbors during the execution and alarms due to this are not triggered. For the given value of p and $k \geq 2$, the epoch has more than one phase. Therefore, within $k^{1+\epsilon}$ rounds into the second phase the leader will change to alarm status in Line 1.13, will not change its status to done later in this epoch, and no other node will receive a termination message. Hence, all nodes will continue to the next epoch.

Finally, if $k = n$, Lemma 2 shows that the accumulated potential ρ will be $k - 1 - 1/k \leq \rho \leq k - 1$. Thus, in Line 1.21 the leader will change its status to done, and in the loop of Line 1.28 will inform all other nodes that the current estimate is correct. The number of iterations of such loop are enough due to 1-interval connectivity.

The claimed running time can be obtained by inspection of the algorithm, either for the leader or non-leader since they are synchronized. Refer for instance to the leader algorithm in Algorithm 1. The outer loop in Line 1.6 corresponds to each epoch with estimates $k = 2, 3, \ldots, n$. For each epoch, Line 1.7 starts a loop of p phases followed by k rounds in Line 1.28. Each of the p phases has r rounds. Thus, the overal number of rounds is $\sum_{k=2}^{n}(pr + k)$. ◀

Choosing $\epsilon = \log_k 2$ and replacing in Theorem 6 yields the following corollary.

▶ **Corollary 7.** *The time complexity of* Methodical Counting *is* $O(n^5 \log^2 n)$.

7 Extensions

We argue that Methodical Counting can be extended to compute the sum of values stored in the nodes, and thus also the average (as it computes the number of nodes n), and other functions. Given that our Counting algorithm is based on mass-distribution, the standard approach could be to compute the average (by sharing the input values repeatedly until convergence to average) at the same time we compute the count. Then, the sum would be simply the average times the count. However, mass-distribution algorithms only converge to the result. That is, we may not get the exact sum with the procedure described. Then, a more careful method is needed.

Assume that each node of the Anonymous Dynamic Network initially stores a value, represented as a sequence of bits. Without loss of generality, we could assume that the value stored at the leader is zero; otherwise, the nodes could compute the sum of other initial values (with the leader value set up to 0), and later the leader could propagate its actual initial value appended to the message "done" at the end of the execution to be added to the computed sum of other nodes.

The modified Methodical Counting prepends the potential to the sequence. Instead of sending potential by the original Methodical Counting, each node transmits its current sequence (in which the potential stands in the first location). Changes at each position of the sequence are done independly by the same algorithm as used for the potential, cf. Algorithms 1 and 2. Re-setting the values, in the beginning of each epoch, means putting back the initial values of the sequence. It means that the modified algorithm maintains potential in exactly the same way as the original Methodical Counting, regardless of the initial values. At the end of some epoch, with number corresponding to the number of nodes n, all nodes terminate. When it happens, each node recalls the sequence stored in it at the end of the first phase of the epoch, multiplies the values stored at each position of the sequence by the epoch number n, and rounds each of the results to the closest integer;

then it sums up the subsequent values multiplied by corresponding (consecutive) powers of 2. Note that such "recalling" could be easily implemented by storing and maintaining the sequence after the first phase of each epoch.

We argue that the computed value is the sum of the initial values. It is enough to analyze how the modified algorithm processes values at one position of the sequence, as positions are treated independently; therefore, w.l.o.g. we assume that each node has value 0 or 1 in the beginning. Consider the last epoch before the leader sends the final sequence (in our case, representing one value). In the beginning of the epoch, the values are re-set to the original one, and manipulated independently according to the rules in Algorithms 1 and 2. Therefore, let us focus on the first phase of this epoch. Since we already proved that the estimate of the last epoch is equal to the number of nodes, the value of d in this epoch (and thus also in its first phase) is an upper bound on the node degree. Thus, the mass distribution scaled down by the sum of the initial values behaves exactly the same as the probabilities of being at nodes in the corresponding round of the lazy random walk, with parameter d and starting from initial distribution equal to the initial values divided by the sum. Since the length of the phase is set up to guarantee that the distribution is close to the stationary uniform within error $1/n$, and the sum of bits is not bigger than n, at the end of the phase the value stored by each node is close to the sum (i.e., scaling factor) divided by n by at most $1/n^4$ (cf. proof of Lemma 2). Therefore, after multiplying it by n, each node gets value of sum within error of at most $1/n^3$, which after rounding will give the integer equal to the value of the sum.

Once having the number n and the sum, each node can compute the average. As argued in [17], the capacity of computing the sum of the input values makes possible the computation of more complex functions. As opposed to [17] where the computation only converges, our approach outputs the exact sum. Therefore, the extension to database queries that can be approximated using *linear synopses*[6] is straightforward. Boolean functions $f : \{0,1\}^n \to \{0,1\}$, such as AND (sum $= n$), OR (sum > 0), and XOR (sum $= 1$), as well as their complementaries NAND (sum $\neq n$), NOR (sum $= 0$), and XNOR (sum $\neq 1$), can also be implemented having n and the sum. This applies also to other "symmetric" (i.e., do not depend on the order of variables) Boolean functions, as they could be computed based on computed sum of ones and n [18]. Maximum (L_∞ norm) and minimum can be computed subsequently by flooding. That is, each node broadcasts the maximum and minimum input values seen so far. Due to 1-interval connectivity within n rounds all nodes have the answers.

Note that all these computations, including the Methodical Counting, could be done using only polynomial estimates of values, that is, with messages of length $O(\log n)$, multiplied by the maximum number of coordinates of any of the initial values. This could be also traded for time: we could use only messages of length $O(\log n)$ with time increased by the maximum number of coordinates of any initial value (which is still polynomial in the size of the input,[7] which in this case is at least n plus the maximum number of coordinates).

8 Open Directions

Straightway questions emerging from our work include existence of polynomial (in n) lower bound and improvement of our upper bound. One of the potential ways could be through investigating bi-directional relationships between random processes and computing algebraic functions in Anonymous Dynamic Network. Extending the range of polynomially computable

[6] Additive functions on multisets, e.g. $f(A \cup B) = f(A) + f(B)$.
[7] The input in this case is distributed among the nodes, and each node possesses at least one bit

functions is another intriguing future direction. Finally, generalizing the model by not assuming connectivity in every round or dropping assumption on synchrony could introduce even more challenging aspects of communication and computation, including group communication and its impact on the common knowledge about the system parameters.

References

1. Chen Avin, Michal Kouckỳ, and Zvi Lotker. How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). In *Automata, languages and programming*, pages 121–132. Springer, 2008.
2. Roberto Baldoni and Giuseppe A Di Luna. Counting on anonymous dynamic networks: Bounds and algorithms. manuscript, 2016.
3. Siddhartha Banerjee, Aditya Gopalan, Abhik Kumar Das, and Sanjay Shakkottai. Epidemic spreading with external agents. *IEEE Transactions on Information Theory*, 60(7):4125–4138, 2014.
4. Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52(6):2508–2530, 2006.
5. Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.
6. Maitri Chakraborty, Alessia Milani, and Miguel A. Mosteiro. Counting in practical anonymous dynamic networks is polynomial. In *Proceedings of the 4th International Conference on Networked Systems*, volume 9944 of *Lecture Notes in Computer Science*, pages 131–136, 2016.
7. Yuxin Chen, Sanjay Shakkottai, and Jeffrey G Andrews. On the role of mobility for multimessage gossip. *IEEE Transactions on Information Theory*, 59(6):3953–3970, 2013.
8. Giuseppe Antonio Di Luna and Roberto Baldoni. Investigating the cost of anonymity on dynamic networks. *CoRR*, abs/1505.03509, 2015. URL: `http://arxiv.org/abs/1505.03509`, `arXiv:1505.03509`.
9. Giuseppe Antonio Di Luna and Roberto Baldoni. Non trivial computations in anonymous dynamic networks. In *Proceedings of the 19th International Conference on Principles of Distributed Systems*, Leibniz International Proceedings in Informatics, 2015. To appear.
10. Giuseppe Antonio Di Luna, Roberto Baldoni, Silvia Bonomi, and Ioannis Chatzigiannakis. Conscious and unconscious counting on anonymous dynamic networks. In *Proceedings of the 15th International Conference on Distributed Computing and Networking*, volume 8314 of *Lecture Notes in Computer Science*, pages 257–271. Springer Berlin Heidelberg, 2014.
11. Giuseppe Antonio Di Luna, Roberto Baldoni, Silvia Bonomi, and Ioannis Chatzigiannakis. Counting in anonymous dynamic networks under worst-case adversary. In *Proceedings of the 34th International Conference on Distributed Computing Systems*, pages 338–347. IEEE, 2014.
12. Giuseppe Antonio Di Luna, Silvia Bonomi, Ioannis Chatzigiannakis, and Roberto Baldoni. Counting in anonymous dynamic networks: An experimental perspective. In *Proceedings of the 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, volume 8243 of *Lecture Notes in Computer Science*, pages 139–154. Springer Berlin Heidelberg, 2014.
13. M. Farach-Colton, A. Fernández Anta, A. Milani, M. A. Mosteiro, and S. Zaks. Opportunistic information dissemination in mobile ad-hoc networks: adaptiveness vs. obliviousness and randomization vs. determinism. In *Proc. of the 10th Latin American Theoretical Informatics Symposium*, volume 7256 of *Lecture Notes in Computer Science*, pages 303–314. Springer-Verlag, Berlin, 2012.

14 Martin Farach-Colton and Miguel A. Mosteiro. Initializing sensor networks of non-uniform density in the weak sensor model. *Algorithmica*, 73(1):87–114, 2015. doi:10.1007/s00453-014-9905-5.

15 A. Fernández Anta and M. A. Mosteiro. Contention resolution in multiple-access channels: k-selection in radio networks. *Discrete Mathematics, Algorithms and Applications*, 02(04):445–456, 2010. doi:10.1142/S1793830910000796.

16 Antonio Fernández Anta, Alessia Milani, Miguel A. Mosteiro, and Shmuel Zaks. Opportunistic information dissemination in mobile ad-hoc networks: the profit of global synchrony. *Distributed Computing*, 25(4):279–296, 2012. doi:10.1007/s00446-012-0165-9.

17 D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proc. of the 44th IEEE Ann. Symp. on Foundations of Computer Science*, pages 482–491, 2003.

18 E. Kranakis, D. Krizanc, and J. Vandenberg. Computing boolean functions on anonymous networks. *Information and Computation*, 114(2):214–236, 1994. doi:10.1006/inco.1994.1086.

19 Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 513–522. ACM, 2010.

20 Othon Michail, Ioannis Chatzigiannakis, and Paul G Spirakis. Naming and counting in anonymous unknown dynamic networks. In *Stabilization, Safety, and Security of Distributed Systems*, pages 281–295. Springer, 2013.

21 Alessia Milani and Miguel A. Mosteiro. A faster counting protocol for anonymous dynamic networks. In *Proceedings of the 19th International Conference on Principles of Distributed Systems*, volume 46 of *Leibniz International Proceedings in Informatics*, pages 1–13, 2015.

22 Damon Mosk-Aoyama and Devavrat Shah. Fast distributed algorithms for computing separable functions. *IEEE Transactions on Information Theory*, 54(7):2997–3007, 2008.

23 Angelia Nedic, Alex Olshevsky, Asuman Ozdaglar, and John N Tsitsiklis. On distributed averaging algorithms and quantization effects. *IEEE Transactions on Automatic Control*, 54(11):2506–2517, 2009.

24 Sujay Sanghavi, Bruce Hajek, and Laurent Massoulié. Gossiping with multiple messages. *IEEE Transactions on Information Theory*, 53(12):4640–4654, 2007.

The Unfortunate-Flow Problem

Orna Kupferman
School of Computer Science and Engineering, The Hebrew University, Israel

Gal Vardi
School of Computer Science and Engineering, The Hebrew University, Israel

Abstract

In the traditional maximum-flow problem, the goal is to transfer maximum flow in a network by directing, in each vertex in the network, incoming flow into outgoing edges. The problem is one of the most fundamental problems in TCS, with application in numerous domains. The fact a maximal-flow algorithm directs the flow in all the vertices of the network corresponds to a setting in which the authority has control in all vertices. Many applications in which the maximal-flow problem is applied involve an adversarial setting, where the authority does not have such a control.

We introduce and study the *unfortunate flow problem*, which studies the flow that is guaranteed to reach the target when the edges that leave the source are saturated, yet the most unfortunate decisions are taken in the vertices. When the incoming flow to a vertex is greater than the outgoing capacity, flow is lost. The problem models evacuation scenarios where traffic is stuck due to jams in junctions and communication networks where packets are dropped in overloaded routers.

We study the theoretical properties of unfortunate flows, show that the unfortunate-flow problem is co-NP-complete and point to polynomial fragments. We introduce and study interesting variants of the problem: *integral unfortunate flow*, where the flow along edges must be integral, *controlled unfortunate flow*, where the edges from the source need not be saturated and may be controlled, and *no-loss controlled unfortunate flow*, where the controlled flow must not be lost.

2012 ACM Subject Classification Mathematics of computing → Network flows

Keywords and phrases Flow Network, Graph Algorithms, Games

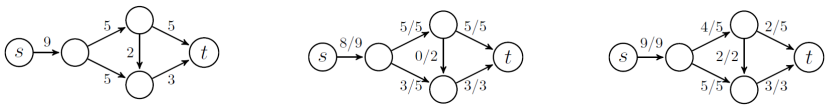
Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.157

Related Version A full version of the paper is available at `http://www.cs.huji.ac.il/~ornak/publications/icalp18.pdf`.

1 Introduction

A *flow network* is a directed graph in which each edge has a capacity, bounding the amount of flow that can travel through it. The amount of flow that enters a vertex equals the amount of flow that leaves it, unless the vertex is a *source*, which has only outgoing flow, or a *target*, which has only incoming flow. The fundamental *maximum-flow problem* gets as input a flow network and searches for a maximal flow from the source to the target [4, 10]. The problem was first formulated and solved in the 1950's [8, 9]. It has attracted much research on improved algorithms, variants, and applications [6, 5, 11, 15].

The maximum-flow problem can be applied in many settings in which something travels along a network. This covers numerous application domains, including traffic in road or rail systems, fluids in pipes, packets in a communication network, and many more [1]. Less obvious applications involve flow networks that are constructed in order to model settings with an abstract network, as in the case of scheduling with constraints [1]. In addition, several

(a) (b) (c)

Figure 1 A flow network $\mathcal{G}$, and preflows that attain its maximum-flow and unfortunate-flow values.

classical graph-theory problems can be reduced to the maximum-flow problem. This includes the problem of finding a maximum bipartite matching, minimum path cover, maximum edge-disjoint or vertex-disjoint path, and many more [4, 1]. Variants of the maximum-flow problem can accommodate further settings, like circulation problems [18], multiple source and target vertices, costs for unit flows, multiple commodities, and more [7].

Studies of flow networks so far assume that the vertices in the network are controlled by a central authority. Indeed, maximum-flow algorithms directs the flow in all vertices of the network. In many applications of flow networks, however, vertices of the network may not be controlled. Thus, every vertex may make autonomous and independent decisions regarding how to direct incoming flow to outgoing edges.

Consider, for example, a road network of a city, where the source s models the center of the city and the target t models the area outside the city. In order to *evacuate* the center of the city, drivers navigate from s to t. In each vertex, every incoming driver chooses an arbitrary outgoing edge with free capacity. If the outgoing capacity from a vertex is less than the incoming flow, then a traffic jam occurs, and flow is lost. As another example, consider a communication network in which packets are sent from a source router s and should reach a target router t. Whenever an internal router receives a packet it forwards it to an arbitrary neighbor router. If the outgoing capacity from a vertex is less than the incoming flow, then packets are dropped, and flow is lost.

In both examples, we want to find the flow that is guaranteed to reach the target in the worst scenario. We now formalize this intuition. Let $\mathcal{G} = \langle V, E, c, s, t\rangle$ be a flow network, where $\langle V, E\rangle$ is a directed graph, $c : E \to \mathbb{N}$ assigns a capacity for each edge, and s, t are the source and target vertices. A *preflow* is a function $f : E \to \mathbb{R}$ that assigns to each edge $e \in E$, a flow in $[0, c(e)]$ such that the incoming flow to each vertex is greater or equal to its outgoing flow. A *saturating preflow* is a preflow in which all outgoing edges from s are saturated, and for every vertex $v \in V \setminus \{s, t\}$, the outgoing flow from v is the minimum between the incoming flow to v and the outgoing capacity from v. That is, in a saturating preflow, flow loss occurs in a vertex v if and only if the incoming flow to v is greater than the capacity of the edges outgoing from v. The *unfortunate flow value* of $\mathcal{G}$ is the minimal flow that reaches t in a saturating preflow. Thus, it is the flow that is guaranteed to reach t when the edges that leave s are saturated, yet the most unfortunate routing decisions are taken in junctions. In the *unfortunate-flow problem*, we want to find the unfortunate flow value of $\mathcal{G}$.

▶ **Example 1.** Consider the flow network $\mathcal{G}$ appearing in Figure 1 (a). A maximum flow in $\mathcal{G}$ has value 8, attained, for example, with the preflow in (b). A saturating preflow in $\mathcal{G}$ appears in (c), and has value 5. While the edges leaving s are saturated, the routing of 7 flow units to the vertex at the bottom leads to a loss of 4 flow units in this vertex. ◀

We introduce the unfortunate-flow problem, study the theoretical properties of saturating preflows, and study the complexity of the problem. We also introduce and study interesting variants of the problem: *integral unfortunate flow*, where the flow along edges must be

integral, *controlled unfortunate flow*, where the edges from the source need not be saturated and may be controlled, and *no-loss controlled unfortunate flow*, where the controlled flow must avoid loss.

Before we describe our contribution, let us review *flow games* and their connection to our contribution here. In flow games [14], the vertices of a flow network are partitioned between two players. Each player controls how incoming flow is partitioned among edges outgoing from her vertices. Then, one player aims at maximizing the flow that reaches t and the other player aims at minimizing it. It is shown in [14] that when the players are restricted to *integral strategies*, thus when the flows along the edges are integers, then the problem of finding the maximal flow that the maximizer player can guarantee is Σ_2^P-complete. The restriction to integral strategies is crucial. Indeed, unlike the case of the traditional maximum-flow problem, non-integral strategies may be better than integral ones. In fact, the problem of finding a maximal flow for the maximizer in a setting with non-integral strategies was left open in [14]. The unfortunate-flow problem can be viewed as a special case of flow games, in which the maximizer player controls no vertex.

We start with the complexity of the unfortunate-flow problem. We consider the decision-problem variant, where we are given a threshold $\gamma > 0$ and decide whether the unfortunate flow value is at least γ. In the case of maximal flow, the problem can be solved in polynomial time [9], and so are many variants of it. We first show that, quite surprisingly, the unfortunate-flow problem is co-NP-hard and that it is NP-hard to approximate within any multiplicative factor. We then point to a polynomial fragment. Intuitively, the fragment restricts the number of vertices in which flow may be lost, which we pinpoint as the computational bottleneck. Formally, we say that a vertex is a *funnel* if its incoming capacity is greater than its outgoing capacity. We show that the unfortunate-flow problem can be solved in time $O(2^{|H|} \cdot (|E|^2 log|V| + |E||V|log^2|V|))$, where $H \subseteq V$ is the set of funnels in $\mathcal{G}$. In particular, the problem can be solved in strongly-polynomial time if the network has a logarithmic number of funnels. Our solution reduces the problem to a sequence of *min-cost max-flow* problems [1], implying the desirable *integral flow property*: the unfortunate-flow value can always be attained by an integral flow. The integral flow property implies a matching co-NP upper bound, thus the unfortunate-flow problem is co-NP-complete.

In some scenarios, we have some initial control on the flow. For example, in evacuation scenarios, as in the example of traffic leaving the city, police may direct cars at the center of the city, but has no control on them once they leave the center. Likewise, when entering or evacuating stadiums, police may direct the crowd to different gates, but has no control on how people proceed once they pass the gates [13]. We study the *controlled unfortunate-flow problem*, where the outgoing flow from s is bounded and controlled. Formally, there is an integer $\alpha \geq 0$ such that the total outgoing flow from s is bounded by α, and it is possible to control how this outgoing flow is partitioned among the edges that leave s. Our goal is to control this flow so that the flow that reaches t in the most unfortunate case is maximized. [1] We show that the integral-flow property no longer holds in this setting. Thus, there are networks in which an optimal strategy is to partition the α units of flow that leave s into non-integers. A troublesome implication of this is that an algorithm that guesses the strategy has to go over unboundedly many possibilities. This challenge is what has left flow games undecidable [14]. We show that we can still reduce the controlled unfortunate-flow problem into the second alternation level of the theory of real numbers under addition and order [17].

[1] We note that this is different from work done in *evacuation planning*, where the goal is to find routes and schedules of evacuees (for a survey, see [16]).

The reduction implies membership in Σ_2^P, and we show a matching lower bound. Thus, the controlled unfortunate-flow problem is Σ_2^P-complete. We also study a generalization of the problem, where control can be placed in a subset of the vertices.[2]

Finally, in some scenarios it is crucial for flow not to get lost. For example, in evacuation scenarios, we may prefer to give up an evacuation attempt under a loss risk, and in communication networks, we may tolerate low traffic and not tolerate dropping of packets. We say that a flow network $\mathcal{G}$ is *safe* if all saturating preflows have no loss. For example, networks with no funnels are clearly safe. It is easy to see that $\mathcal{G}$ is safe if its unfortunate flow value is equal to the maximal flow the source can generate, thus to the capacity of the edges outgoing from the source. This gives a co-NP algorithm for deciding the safety of a network. We show one can do better and reduce the safety problem to a maximum weighted flow problem, which can be solved in polynomial time. We then turn to study the *no-loss controlled unfortunate-flow problem*, where we control the flow in edges from s, and we want to maximize the flow to t but in a way that flow loss is impossible. We show that the problem is NP-complete.

Due to space limitations, some examples and proofs are omitted and can be found in the full version, in the authors' URLs.

2 Preliminaries

A *flow network* is $\mathcal{G} = \langle V, E, c, s, t\rangle$, where V is a set of vertices, $E \subseteq V \times V$ is a set of directed edges, $c : E \to \mathbb{N}$ is a capacity function, and $s, t \in V$ are source and target vertices. The capacity function assigns to each edge $e \in E$ a nonnegative capacity $c(e) \geq 0$. We define the *size* of $\mathcal{G}$, denoted $|\mathcal{G}|$ by $|V| + |E| + |c|$, where $|c|$ is the size required for encoding the capacity function c, thus assuming the capacities are given in binary. For a vertex $v \in V$, let $E^{\to v}$ and $E^{v\to}$ be the sets of incoming and outgoing edges to and from v, respectively. That is, $E^{\to v} = (V \times \{v\}) \cap E$ and $E^{v\to} = (\{v\} \times V) \cap E$. A *sink* is a vertex v with no outgoing edges, thus $E^{v\to} = \emptyset$. We assume that t is a sink, it is reachable from s, and $E^{\to s} = \emptyset$. We also assume that $\langle V, E\rangle$ does not contain parallel edges and self loops. For a vertex $v \in V$, let $c(\to v) = \sum_{e\in E^{\to v}} c(e)$ and $c(v\to) = \sum_{e\in E^{v\to}} c(e)$ be the sums of capacities of edges that enter and leave v, respectively. We say that a vertex $v \in V$ is a *funnel* if $c(v\to) < c(\to v)$. We use C_s to denote the total capacity of edges outgoing from the source, thus $C_s = c(s\to)$.

A *preflow* in $\mathcal{G}$ is a function $f : E \to \mathbb{R}$ that satisfies the following two properties:

- For every $e \in E$, we have that $0 \leq f(e) \leq c(e)$.
- For every vertex $v \in V \setminus \{s\}$, the incoming flow to v is greater or equal to its outgoing flow. Formally, $\sum_{e\in E^{\to v}} f(e) \geq \sum_{e\in E^{v\to}} f(e)$.

For a preflow f and an edge $e \in E$, we say that e is *saturated* if $f(e) = c(e)$. We extend f to vertices: for every vertex $v \in V$, let $f(\to v) = \sum_{e\in E^{\to v}} f(e)$ and $f(v\to) = \sum_{e\in E^{v\to}} f(e)$. For a vertex $v \in V \setminus \{s, t\}$, the *flow loss of f in v*, denoted $l_f(v)$, is the quantity that enters v and does not leave v. Formally, $l_f(v) = f(\to v) - f(v\to)$. Then, $L_f = \sum_{v\in V\setminus\{s,t\}} l_f(v)$ is the *flow loss of f*. The value of a preflow f, denoted $val(f)$, is $f(\to t)$; that is, the incoming flow to t. Note that $val(f) = f(s\to) - L_f$. A *flow* is a preflow f with $L_f = 0$. A *maximum flow* is a flow with a maximal value.

[2] Not to confuse with the problem of finding critical nodes for firefighters [2, 3]. While there the firefighters block the fire, in our setting they direct the evacuation. Thus, there, the goal is to block undesired vulnerabilities in the network, and here the goal is maximize desired traffic.

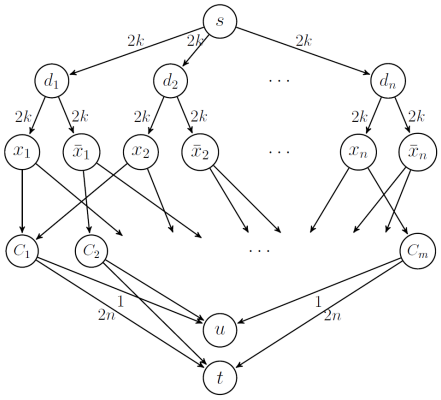


Figure 2 The flow network $\mathcal{G}$. The capacities of the edges entering $C_1, \ldots, C_m$ are 1.

A *saturating preflow* is a preflow in which all edge in $E^{s\rightarrow}$ are saturated, and for every $v \in V \setminus \{s, t\}$, we have $f(v\rightarrow) = \min\{f(\rightarrow v), c(v\rightarrow)\}$. That is, in a saturating preflow, flow loss may occur in a vertex v only if the incoming flow to v is bigger than the capacities of the edges outgoing from v.

The *unfortunate value* of a flow network $\mathcal{G}$, denoted $uval(\mathcal{G})$, is the minimal value of a saturating preflow in $\mathcal{G}$. That is, it is the value that is guaranteed to reach t when the edges that leave s are saturated, yet the most unfortunate routing decisions are taken in junctions. An *unfortunate saturating preflow* is a saturating preflow that attains the network's unfortunate value. The *unfortunate flow* problem (UF problem, in short) is to decide, given a flow network $\mathcal{G}$ and a threshold $\gamma > 0$, whether $uval(\mathcal{G}) \geq \gamma$.

3 The Complexity of the Unfortunate-Flow Problem

In this section we study the complexity of the unfortunate-flow problem. We start with bad news and show that the problem is co-NP-hard, and in fact is NP-hard to approximate within any multiplicative factor. A more precise analysis of the complexity then enables us to point to a polynomial fragment and to prove an integral-flow property, which implies a matching co-NP upper bound.

▶ **Theorem 2.** *The UF problem is co-NP-hard.*

Proof. We show a reduction from CNF-SAT to the complement problem, namely deciding whether $uval(\mathcal{G}) < \gamma$ for some $\gamma \in \mathbb{N}$. Let $\psi = C_1 \wedge \ldots \wedge C_m$ be a CNF formula over the variables $x_1 \ldots x_n$. We assume that every literal in $x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n$ appears in exactly k clauses in ψ. Indeed, every CNF formula can be converted to such a formula in polynomial time and with a polynomial blowup. We construct a flow network $\mathcal{G} = \langle V, E, c, s, t\rangle$ as demonstrated in Figure 2. Let $Z = \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$. For a literal $z \in Z$ and a clause C_i, the network $\mathcal{G}$ contains an edge $\langle z, C_i\rangle$ iff C_i contains z. Thus, each vertex in Z has exactly k outgoing edges. The capacities of these edges are 1. Each vertex C_i has two outgoing edges – to t and to the sink u. In the full version, we prove that ψ is satisfiable iff $uval(\mathcal{G}) < kn - m + 1$. ◀

By a simple manipulation of the network $\mathcal{G}$ constructed in the reduction in the proof of Theorem 2, we can obtain, given a CNF-SAT formula ψ, a network $\mathcal{G}'$ such that if ψ is

satisfiable, then $uval(\mathcal{G}') = 0$, and otherwise, $uval(\mathcal{G}') \geq 1$. Hence the following (a detailed proof can be found in the full version.

▶ **Theorem 3.** *It is NP-hard to approximate the UF problem within any multiplicative factor.*

Following the hardness of the problem, we turn to analyze its complexity in terms of the different parameters of the flow network. Our analysis points to a class of networks for which the UF problem can be solved in polynomial time.

Consider a flow network $\mathcal{G} = \langle V, E, c, s, t\rangle$. Let $H \subseteq V \setminus \{s, t\}$ be the set of funnels in $\mathcal{G}$. Thus, $H = \{v : c(\rightarrow v) > c(v\rightarrow)\}$. For $L \subseteq V$, let $\mathcal{F}_L$ be a set of saturating preflows in which edges outgoing from vertices in L are saturated, and flow loss may occur only in vertices in L. Thus, $f \in \mathcal{F}_L$ iff f is a saturating preflow in $\mathcal{G}$ such that for every $u \in L$, we have $f(u\rightarrow) = c(u\rightarrow)$, and for every $u \in V \setminus L$, we have $l_f(u) = 0$. By the definition of a saturating flow, flow loss in $\mathcal{G}$ may occur only in vertices in H. Accordingly, we have the following.

▶ **Lemma 4.** $\bigcup_{L\subseteq H} \mathcal{F}_L$ *contains all the saturating preflows in* $\mathcal{G}$.

By Lemma 4, a search for the unfortunate value of $\mathcal{G}$ can be restricted to preflows in $\mathcal{F}_L$, for $L \subseteq H$. Accordingly, the UF problem can be solved by solving $2^{|H|}$ optimization problems, solvable by either linear programming (Theorem 5) or a reduction to the min-cost max-flow problem (Theorem 6).

▶ **Theorem 5.** *Consider a flow network* $\mathcal{G}$ *and let* H *be the set of funnels in* $\mathcal{G}$. *The UF problem for* $\mathcal{G}$ *can be solved in time* $2^{|H|} \cdot poly(|\mathcal{G}|)$.

Proof. The algorithm goes over all the subsets of H and for each subset $L \subseteq H$, finds a minimum-value preflow in $\mathcal{F}_L$. The latter is done by linear programming. Given L, the linear program for $\mathcal{F}_L$ is described below. The variable x_e, for every $e \in E$, stands for $f(e)$. The program is of size linear in $|\mathcal{G}|$, thus the overall complexity is $2^{|H|} \cdot poly(|\mathcal{G}|)$.

minimize	$\sum_{e\in E^{\rightarrow t}} x_e$	
subject to	$0 \leq x_e \leq c(e)$	for each $e \in E$
	$x_e = c(e)$	for each $u \in L \cup \{s\}, e \in E^{u\rightarrow}$
	$\sum_{e\in E^{u\rightarrow}} x_e \leq \sum_{e\in E^{\rightarrow u}} x_e$	for each $u \in L$
	$\sum_{e\in E^{u\rightarrow}} x_e = \sum_{e\in E^{\rightarrow u}} x_e$	for each $u \notin L \cup \{s, t\}$

◀

The complexity of solving each linear program in the algorithm described in the proof of Theorem 5 is polynomial in $|\mathcal{G}|$, but not *strongly polynomial*. Thus its running time depends (polynomially) on the number of bits required for representing the capacities in $\mathcal{G}$. We now describe an alternative algorithm whose complexity depends only on the number of vertices and edges in the network.

Our algorithm reduces the problem of finding a minimal-value preflow in $\mathcal{F}_L$ to the *min-cost max-flow* problem in flow networks with costs [1]. A flow network with costs is $\mathcal{G} = \langle V, E, a, c, s, t\rangle$, where $\langle V, E, c, s, t\rangle$ is a flow network and $a : E \rightarrow \mathbb{R}$ is a cost function. The cost of a flow f in $\mathcal{G}$, denoted $cost(f)$, is $\sum_{e\in E} a(e) \cdot f(e)$. In the min-cost max-flow problem we are given a flow network with costs, and find a maximum flow with a minimum cost. By [1], this problem can be solved in time $O(|E|^2 log|V| + |E||V|log^2|V|)$.

▶ **Theorem 6.** *Consider a flow network* $\mathcal{G} = \langle V, E, c, s, t\rangle$ *and let* H *be the set of funnels in* $\mathcal{G}$. *The UF problem for* $\mathcal{G}$ *can be solved in time* $O(2^{|H|} \cdot (|E|^2 log|V| + |E||V|log^2|V|))$.

Proof. The algorithm finds, for each subset $L \subseteq H$, a minimum-value preflow in $\mathcal{F}_L$ by a reduction to the min-cost max-flow problem. By Lemma 4, the minimum value found for some $L \subseteq H$ is $uval(\mathcal{G})$.

Consider the flow network with costs $\mathcal{G}' = \langle V', E', a, c', s, t' \rangle$ that is obtained from $\mathcal{G}$ as follows. We add a new vertex t' and edges $\langle u, t' \rangle$ for every $u \in L \cup \{t\}$, thus $V' = V \cup \{t'\}$ and $E' = E \cup (L \cup \{t\}) \times \{t'\}$. The capacity $c'(e)$ for every new edge $e \in E' \setminus E$ is large (for example, it may be C_s), and for every $e \in E$, we have $c'(e) = c(e)$. Let $C = \max\{c(e) : e \in E\}$ denote the maximal capacity in $\mathcal{G}$. For edges $e \in L \times \{t'\}$, we define $a(e) = -1$; for edges $e \in L \times V$, we define $a(e) = -C \cdot |V|^2$; and for all the other edges, we define $a(e) = 0$. Intuitively, the costs of the edges in $L \times V$ are negative and small enough, so that a min-cost max-flow in $\mathcal{G}'$ would have to saturate them first, and only then try to direct flow to edges in $L \times \{t'\}$.

In the full version, we prove the correctness of the following algorithm: First, find a min-cost max-flow f' in $\mathcal{G}'$. If $val(f') < C_s$ or $cost(f') > -C \cdot |V|^2 \cdot \sum_{e \in L \times V} c(e)$, then $\mathcal{F}_L = \emptyset$. Otherwise, the minimal value of a preflow in $\mathcal{F}_L$ is $C_s + cost(f') + C \cdot |V|^2 \cdot \sum_{e \in L \times V} c(e)$. Since the min-cost max-flow problem can be solved in time $O(|E|^2 log|V| + |E||V|log^2|V|)$ [1] and there are $2^{|H|}$ subsets of funnels to check, the required complexity follows. ◀

▶ **Corollary 7.** *The UF problems for networks with a logarithmic number of funnels can be solved in strongly-polynomial time.*

We say that a preflow $f : E \to \mathbb{R}$ is *integral* if $f(e) \in \mathbb{N}$ for all $e \in E$. It is sometimes desirable to restrict the flow to an integral one, for example in settings in which the objects we transfer along the network cannot be partitioned into fractions. We now show that the UF problem always has an integral-flow solution, and that such a solution can be obtained by the algorithm shown in the proof of Theorem 6. Essentially (see proof in the full version), it follows from the fact that the min-cost max-flow problem has an integral solution. As we show in Section 4, this *integral flow property* is not maintained in variants of the UF problem.

▶ **Theorem 8.** *The UF problem has an integral-flow solution: for every flow network, there exists an integral unfortunate saturating preflow. Moreover, such integral preflow can be found by the algorithm described in the proof of Theorem 6.*

The integral-flow property suggests an optimal algorithm for solving the UF problem:

▶ **Theorem 9.** *The UF problem is co-NP-complete.*

Proof. Hardness in co-NP is proven in Theorem 2. We prove membership in NP for the complementary problem: given $\gamma > 0$ and a flow network $\mathcal{G}$, we need to decide whether $uval(\mathcal{G}) < \gamma$. According to Theorem 8, it is enough to decide whether there is a saturating preflow f in which for every $e \in E$, the value $f(e)$ is an integer, and $val(f) < \gamma$. Given a function $f : E \to \mathbb{N}$, checking whether f satisfies these requirements can be done in polynomial time, implying membership in NP. ◀

4 The Controlled Unfortunate-Flow Problem

In this section we study the controlled unfortunate-flow problem, where the outgoing flow from s is bounded and controlled. That is, there is $0 \leq \alpha \leq C_s$ such that the total outgoing flow from s is bounded by α, and it is possible to control how this outgoing flow is partitioned among the edges that leave s. Our goal is to control this flow so that the flow that reaches t in the worst case is maximized. As discussed in Section 1, this problem is motivated by scenarios where we have an initial control on the flow, say by positioning police at the entrance to a stadium or at the center of a city we need to evacuate, or by transmitting messages we want to send from a router we own.

For $\alpha \geq 0$, a *regulator with bound* α is a function $g : E^{s\rightarrow} \rightarrow \mathbb{R}$ that directs α flow units from s. Formally, for every $e \in E^{s\rightarrow}$, we have $0 \leq g(e) \leq c(e)$, and $\sum_{e \in E^{s\rightarrow}} g(e) \leq \alpha$. A *controlled saturating preflow that respects a regulator* g is a preflow $f : E \rightarrow \mathbb{R}$ such that for every $e \in E^{s\rightarrow}$, we have $f(e) = g(e)$, and for every $v \in V \setminus \{s, t\}$, we have $f(v\rightarrow) = \min\{f(\rightarrow v), c(v\rightarrow)\}$. Thus, unlike saturating preflow, here the edges in $E^{s\rightarrow}$ need not be saturated and the flow in them is induced by g. The *unfortunate g-controlled value* of $\mathcal{G}$, denoted $cuval(\mathcal{G}, g)$, is the minimal value of a controlled saturating preflow that respects g. Then, the *unfortunate α-controlled value* of $\mathcal{G}$, denoted $cuval(\mathcal{G}, \alpha)$ is the maximal unfortunate g-controlled value of $\mathcal{G}$ for some regulator g with bound α. In the *controlled unfortunate flow* problem (CUF problem, for short), we are given a flow network $\mathcal{G}$, a bound $\alpha \geq 0$, and a threshold $\gamma > 0$, and we need to decide whether $cuval(\mathcal{G}, \alpha) \geq \gamma$. Thus, in the CUF problem we need to decide whether there is a regulator g with bound α that ensures a value of at least γ.

For two regulators g and g', we denote $g \geq g'$ if for every $e \in E^{s\rightarrow}$, we have $g(e) \geq g'(e)$. In the following theorem we show that the g-controlled unfortunate value is monotonic with respect to g. Thus, increasing g can only increase the value. In particular, it follows that a maximal $cuval(\mathcal{G}, \alpha)$ is obtained with $\alpha = C_s$ and a regulator g in which $g(e) = c(e)$ for every $e \in E^{s\rightarrow}$. Thus, if the outgoing flow from s is not bounded, then the optimal behavior is to saturate the edges in $E^{s\rightarrow}$. Essentially (see full proof in the full version of the paper), it follows from the fact that given g and g' such that $g \geq g'$, and a minimum-value controlled saturating preflow f that respects g, we can construct a controlled saturating preflow f' that respects g' and such that $val(f) \geq val(f')$.

▶ **Theorem 10.** *Consider a flow network $\mathcal{G}$, and let g, g' be two regulators such that $g \geq g'$. Then, $cuval(\mathcal{G}, g) \geq cuval(\mathcal{G}, g')$.*

We now turn to study the complexity of the CUF problem. We first explain why the problem is challenging. One could expect an algorithm in which, given $\mathcal{G}$, α, and γ, we guess an *integral regulator* $g : E^{s\rightarrow} \rightarrow \mathbb{N}$ with bound α, and then use an NP oracle in order to check whether $cuval(\mathcal{G}, g) \geq \gamma$. The problem with the above idea is that it restricts the regulators to integral ones. In Theorem 11 below we show that in some cases, an optimal regulator must use non-integral values. Accordingly, an algorithm that guesses a regulator, as has been the case with the guessed flows in Theorem 9, has to go over unboundedly many possibilities. In fact, when an arbitrary set of vertices (rather than the source only) may be controlled, the problem is not known to be decidable [14].

▶ **Theorem 11.** *Integral regulators are not optimal: There is a flow network $\mathcal{G}$ such that for every integral regulator $g : E^{s\rightarrow} \rightarrow \mathbb{N}$ with bound 2 we have $cuval(\mathcal{G}, g) = 1$, but there is a regulator $g' : E^{s\rightarrow} \rightarrow \mathbb{R}$ with bound 2 such that $cuval(\mathcal{G}, g') = 2$.*

Proof. Consider the flow network $\mathcal{G}$ appearing in Figure 3. For every pair $\langle u_i, u_j \rangle$ for $1 \leq i < j \leq 4$, the network $\mathcal{G}$ contains a vertex v_{ij} with incoming edges from u_i and u_j. The capacities of the edges in $\mathcal{G}$ are all 1. It is not hard to see that for every integral regulator g with bound 2 we have $cuval(\mathcal{G}, g) = 1$. Indeed, for such g there is a controlled saturating preflow that respects g, which directs a flow of 2 to some vertex v_{ij}, causing a loss of 1 in v_{ij}. Consider now the regulator g' that assigns a flow of 0.5 to every edge in $E^{s\rightarrow}$. In this case, a flow of more than 1 cannot be directed to any vertex v_{ij} and therefore $cuval(\mathcal{G}, g') = 2$. ◀

We turn to solve the CUF problem. Theorem 11 forces us to consider non-integral regulators. We do this by a reduction to a problem with a similar challenge, namely *the second alternation level of the theory of real numbers under addition and order*. There, we are given a

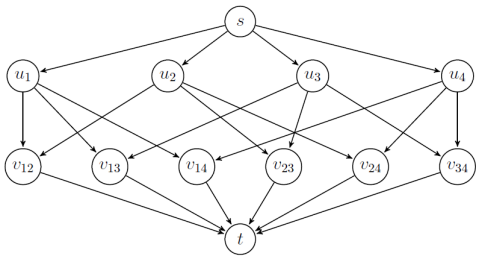


Figure 3 The flow network $\mathcal{G}$. All capacities are 1.

formula of the form $\exists x_1 \ldots x_n \forall y_1 \ldots y_m F(x_1, \ldots, x_n, y_1, \ldots, y_m)$, where F is a propositional combination of linear inequalities of the form $a_1 x_1 + \ldots a_n x_n + b_1 y_1 + \ldots + b_m y_m \leq d$, for constant integers $a_1, \ldots, a_n$, $b_1, \ldots, b_m$, and d, and we have to decide whether there is an assignment of $x_1, \ldots, x_n$ to real numbers so that F is satisfied for every assignment of $y_1, \ldots, y_m$ to real numbers. Even though the domain of possible solutions is infinite, It is shown in [17] that the problem can be solved in Σ_2^P, namely the class of problems that can be solved by a nondeterministic polynomial Turing machine that has an oracle to some NP-complete problem. In [14], a Σ_2^P lower bound is proven for the problem of finding the value of a flow game, where the outgoing flow of a subset of the vertices can be controlled. Recall that in the CUF problem, only the flow from the source vertex can be controlled. While this corresponds to the "exists-forall" nesting of quantifiers that characterizes reasoning in Σ_2^P, it not clear how to reduce Boolean formulas to unfortunate flows. Indeed, in the reduction in [14], control in intermediate vertices is used in order to model disjunctions in the formulas. In the CUF problem, such a control is impossible, as all vertices in the network except for the source are treated in a conjunctive manner.

▶ **Theorem 12.** *The CUF problem is Σ_2^P-complete.*

Proof. We first prove membership in Σ_2^P by a reduction to the second alternation level of the theory of real numbers under addition and order. Given G, α, and γ, we construct a propositional combination F of linear inequalities over the variables x_e for every $e \in E^{s\rightarrow}$, and variables y_e, for every $e \in E$. The formula F states that the values of the variables x_e corresponds to a regulator g with bound α, and that if the values of the variables y_e correspond to a controlled saturating preflow f that respects g then $val(f) \geq \gamma$. Then, our problem amounts to deciding whether there are real values x_e such that for every real values y_e the formula F holds.

For the lower bound, we describe a reduction from QBF_2, namely satisfiability for quantified Boolean formulas with one alternation of quantifiers, where the external quantifier is "exists". Let ψ be a propositional formula over the variables $x_1, \ldots, x_n, y_1, \ldots, y_m$, and let $\theta = \exists x_1 \ldots \exists x_n \forall y_1 \ldots \forall y_m \psi$. Also, let $X = \{x_1, \ldots, x_n\}$, $\bar{X} = \{\bar{x}_1, \ldots, \bar{x}_n\}$, $Y = \{y_1, \ldots, y_m\}$, $\bar{Y} = \{\bar{y}_1, \ldots, \bar{y}_m\}$, $Z = X \cup Y$, and $\bar{Z} = \bar{X} \cup \bar{Y}$. We construct a flow network $\mathcal{G}_\theta$ and define α and γ, such that θ holds iff there is a regulator g with bound α such that $cuval(\mathcal{G}_\theta, g) \geq \gamma$.

We assume that ψ is given in a positive normal form; that is, ψ is constructed from the literals in $Z \cup \bar{Z}$ using the Boolean operators $\vee$ and $\wedge$, and that there is $k \geq 1$ such that every literal in $Z \cup \bar{Z}$ appears in ψ exactly k times. Clearly, every Boolean propositional formula can be converted with only a quadratic blow-up to an equivalent one that satisfies these conditions.

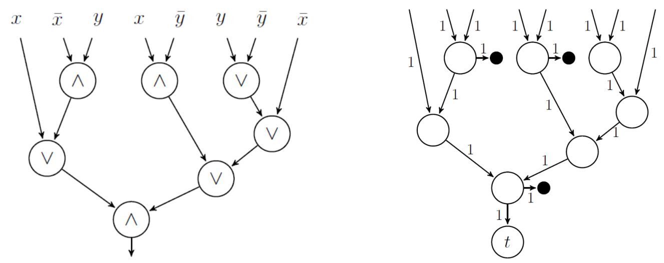


Figure 4 The Boolean circuit $\mathcal{C}_\psi$ and the external-source flow network G_ψ.

We first translate ψ into a Boolean circuit $\mathcal{C}_\psi$ with $k(2n+2m)$ inputs – one for each occurrence of a literal in ψ. For example, in Figure 4, on the left, we describe C_ψ for $\psi = x \vee (\bar{x} \wedge y) \wedge ((x \wedge \bar{y}) \vee (y \vee \bar{y} \vee \bar{x}))$. Each gate in $\mathcal{C}_\psi$ has fan-in 2 and fan-out 1. We say that an input assignment to $\mathcal{C}_\psi$ is consistent if it corresponds to an assignment to the variables in Z. That is, for each variable $z \in Z$, there is a value $b \in \{0,1\}$ such that all the k inputs that correspond to the literal z have value b and all the k inputs that correspond to the literal $\bar{z}$ have value $1-b$. If the input to $\mathcal{C}_\psi$ is consistent then $\mathcal{C}_\psi$ computes the value of ψ for the corresponding assignment.

Now, we translate $\mathcal{C}_\psi$ to an *external-source flow network* $\mathcal{G}_\psi = \langle V, E, c, t\rangle$: a flow network in which there is no source vertex, and an input flow is given externally. Formally, some of the edges in E have an unspecified source, to be later connected to edges with an unspecified target. The idea behind the translation is as follows: The capacities in $\mathcal{G}_\psi$ are all 1. Each OR gate in $\mathcal{C}_\psi$ induces a vertex v that has in-degree 2 and out-degree 1. Thus, if the incoming flow in each incoming edge to v is 0 or 1, then its outgoing flow is 1 iff at least one of its incoming edges has flow 1. Then, each AND gate in $\mathcal{C}_\psi$ induces a vertex v that has in-degree 2 and out-degree 2, yet, one of the two edges that leaves v leads to a sink. Accordingly, if the incoming flow in each incoming edge to v is 0 or 1, then the outgoing flow in the edge that does not lead to the sink must be 1 iff both incoming edges have flow 1. For example, the Boolean circuit $\mathcal{C}_\psi$ from Figure 4 is translated to the external-source flow network $\mathcal{G}_\psi$ to its right.

Given a flow from the external source, we define the unfortunate value of $\mathcal{G}_\psi$ as the minimal value of a controlled saturating preflow that respects the external flow. The following lemma can be easily proved by induction on the structure of ψ.

▶ **Lemma 13.** *Consider a Boolean formula ψ and its corresponding external-source flow network $\mathcal{G}_\psi$.*

1. *Given input flows to $\mathcal{G}_\psi$, if we increase some input flow, then the new unfortunate value of $\mathcal{G}_\psi$ is greater than or equal to the original unfortunate value.*
2. *Given input flows in $\{0,1\}$ to $\mathcal{G}_\psi$, the unfortunate value of $\mathcal{G}_\psi$ is equal to the output of $\mathcal{C}_\psi$ with the same input. Thus, if the input flow to $\mathcal{G}_\psi$ corresponds to a consistent input to $\mathcal{C}_\psi$, then the unfortunate value of $\mathcal{G}_\psi$ is the value of ψ for the corresponding assignment.*

We complete the reduction by constructing the flow network $\mathcal{G}_\theta$ that uses $\mathcal{G}_\psi$ as a subnetwork as shown in Figure 5 The vertices $d_{y_1}, \ldots, d_{y_m}$ are associated with the variables in Y. The vertices $x_i, \bar{x}_i, y_i, \bar{y}_i$ for every i are associated with the literals in $Z \cup \bar{Z}$. Each outgoing edge from a literal vertex that enters $\mathcal{G}_\psi$ is connected to an input of $\mathcal{G}_\psi$ that corresponds to this literal. The outgoing edge from the subnetwork $\mathcal{G}_\psi$ corresponds to an edge from the target vertex of $\mathcal{G}_\psi$. In the full version we describe the network $\mathcal{G}_\theta$ for the case $\psi = (x \vee y) \wedge (\bar{x} \vee \bar{y})$.

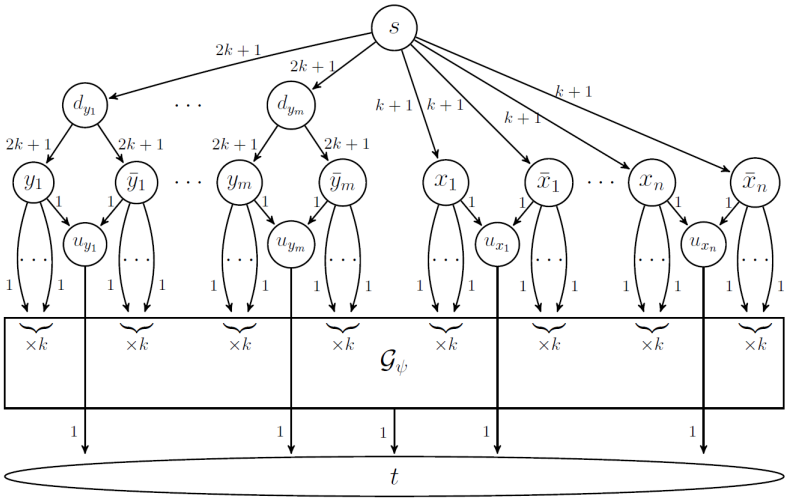


Figure 5 The flow network $\mathcal{G}_\theta$.

In the full version we prove that θ holds iff there is a regulator g with bound $(2k+1)m + (k+1)n$ such that for every controlled saturating preflow f that respects g we have $val(f) \geq m + n + 1$.

◀

In Theorem 11 we showed that in some cases an optimal regulator must use non-integral values. Sometimes, however, it is desirable to restrict attention to integral regulators. In the following theorem (see proof in the full version) we show that the Σ_2^P-completeness stays valid also for integral regulators.

▶ **Theorem 14.** *Let $\mathcal{G}$ be a flow network and let α, γ be integral constants. Deciding whether there exists an integral regulator $g : E^{s\rightarrow} \to \mathbb{N}$ with bound α such that $cuval(\mathcal{G}, g) \geq \gamma$, is Σ_2^P-complete.*

▶ Remark. **[Bounded Global Control]** In the CUF problem, it is possible to control the flow leaving the source. This could be generalized by letting an authority control also internal vertices in the network. In the *bounded global control* problem, we get as input a flow network $\mathcal{G}$, a number $k \geq 0$, and a threshold $\gamma > 0$, and we need to decide whether we can guarantee an unfortunate flow of at least γ by controlling the outgoing flow in at most k vertices. Note that while in the problem of finding critical nodes for firefighters [2, 3], a firefighter blocks the fire, in our setting the firefighters direct the evacuation. Thus, there, the goal is to block undesired vulnerabilities in the network, and here the goal is maximize desired traffic in the network. The formal definition of the bounded global control problem goes through the flow games of [14], which includes the notion of strategies for controlling flow. The Σ_2^P algorithm for solving flow games with integral flows can be extended to solve the bounded global control problem. By making the control on the source vertex essential (say, by adding a transition with a large capacity to a sink), the CUF problem can be reduced to the global control problem with $k = 1$, implying Σ_2^P completeness.

5 Safe Networks and No-Loss Unfortunate Flow

In this section we consider settings in which loss must be avoided. We say that a flow network $\mathcal{G}$ is *safe* if $L_f = 0$ for every saturating preflow f. For example, networks with no funnels are clearly safe. It is easy to see that $\mathcal{G}$ is safe iff $uval(\mathcal{G}) = C_s$. Together with Theorem 9, this gives a co-NP algorithm for deciding the safety of a network. We first show that by reducing the safety problem to the maximum weighted flow problem, we can decide safety in polynomial time. Essentially, the reduction checks, for every vertex $v \in V$, whether it is possible to direct to v flow that is greater than its outgoing capacity, and the weights are used in order to filter flow incoming to v. For details, see the full version.

▶ **Theorem 15.** *Deciding whether a flow network is safe can be done in polynomial time.*

We now consider the case where the total outgoing flow from s is controlled, and we need to find an optimal regulator that guarantees no flow loss. Formally, in the *no-loss controlled unfortunate-flow problem* (NLCUF problem, for short), we are given a flow network $\mathcal{G}$ and an integer $\gamma > 0$, and we need to decide whether there exists a regulator g such that $\sum_{e \in E^{s\rightarrow}} g(e) \geq \gamma$, and for every controlled saturating preflow that respects g the flow loss is 0 (equivalently, $cuval(\mathcal{G}, g) = \gamma$). That is, decide whether there is a regulator that ensures no loss and a value of at least γ. We show that the NLCUF problem is NP-complete. For the upper bound one could expect an algorithm in which we guess an integral regulator $g : E^{s\rightarrow} \rightarrow \mathbb{N}$ in which the total flow is at least γ, and then use Theorem 15 in order to check in polynomial time whether flow loss is possible. However, Theorem 11 shows that in some cases a regulator must use non-integral values in order to ensure that flow loss is impossible. Consequently, our algorithm is more complicated and uses a result from the theory of real numbers with addition.

▶ **Theorem 16.** *The NLCUF problem is NP-complete.*

Proof. We start with the upper bound. For a rational number q we denote by $\#(q)$ the *length* of q, namely, if $q = a/b$ with a, b relatively prime, then $\#(q)$ is the sum of the number of bits in the binary representations of a and b. Consider a formula $\varphi = \exists x_1, ..., x_n \forall y_1, ..., y_m F(x_1, ..., x_n, y_1, ..., y_m)$, where F is a propositional combination of linear inequalities of the form $a_1x_1 + ... + a_nx_n + b_1y_1 + ... + b_my_m \leq d$ for integral constants $a_1, ..., a_n$, $b_1, ..., b_m$, and d. The variables $x_1, ..., x_n, y_1, ..., y_m$ are real. In [17] (in the proof of Theorem 3.1 there) it is shown that φ holds iff there exists rational values $x_1, ..., x_n$ such that for every i the length $\#(x_i)$ is polynomial in the size of φ and for every real values $y_1, ..., y_m$ the formula F holds.

We construct a propositional combination F of linear inequalities over the variables x_e, for every $e \in E^{s\rightarrow}$, and y_e, for every $e \in E$. The formula F states that the values of the variables x_e correspond to a regulator g with bound γ, and that if the values of the variables y_e correspond to a controlled saturating preflow f that respects g, then $L_f = 0$. Then, our problem amounts to deciding whether there are real values x_e such that for every real values y_e, the formula F holds. By [17], it is enough to check whether there are rational values x_e for $e \in E^{s\rightarrow}$ with polynomial lengths such that for every real values y_e for $e \in E$, the formula F holds. Given values for the variables x_e, checking whether for every real values y_e the formula F holds can be done in polynomial time with the algorithm shown in the proof of Theorem 15. Hence the membership in NP.

We proceed to the lower bound. We show a reduction from CNF-SAT. Let $\psi = C_1 \wedge ... \wedge C_m$ be a CNF formula over the variables $x_1 \dots x_n$. We denote $Z = \{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$. We assume that for every literal $z \in Z$ there is at least one clause in ψ that does not contain

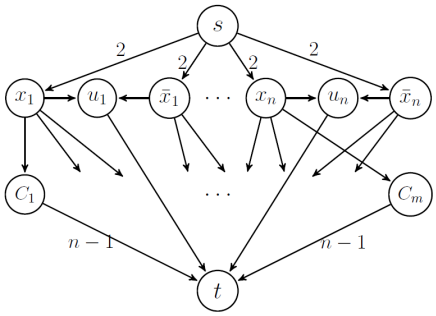


Figure 6 The flow network $\mathcal{G}$. Unless stated otherwise, the capacities are 1.

z. We construct a flow network $\mathcal{G} = \langle V, E, c, s, t\rangle$ as demonstrated in Figure 6. For a literal $z \in Z$ and a clause C_i, the network $\mathcal{G}$ contains an edge $\langle z, C_i\rangle$ iff the clause C_i does not contain the literal z. Let $\gamma = 2n$. In the full version, we show that ψ is satisfiable iff there is a regulator that ensures no loss and a value of at least γ. ◀

Sometimes it is desirable to restrict attention to integral regulators. As we show below, NP-completeness applies for them too (see the full version for proof).

▶ **Theorem 17.** *Let $\mathcal{G}$ be a flow network and let $\gamma > 0$ be an integer. Deciding whether there exists an integral regulator $g : E^{s\to} \to \mathbb{N}$ in $\mathcal{G}$ such that $\sum_{e\in E^{s\to}} g(e) \geq \gamma$, and for every controlled saturating preflow that respects g the flow loss is 0, is NP-complete.*

6 Discussion

The unfortunate-flow problem captures settings in which the authority has no control on how flow is directed in the vertices of a flow network. For many problems, a transition from a cooperative setting to an adversarial one dualizes the complexity class to which the problem belongs, as in NP for satisfiability vs. co-NP for validity. In the case of flow, the polynomial complexity of the maximum-flow problem is not preserved when we move to the dual unfortunate-flow problem, and we prove that the problem is co-NP-complete.

On the positive side, the integral-flow property of maximal flow is preserved in unfortunate flows. This property, however, is lost once we move to controlled unfortunate flows, where non-integral regulators may be more optimal than integral ones. The need to consider real-valued flows questions the decidability of the controlled unfortunate-flow problem. As we show, the problem is decidable, by a reduction to the second alternation level of the theory of real numbers under addition and order [17]. There, the infinite domain of the real numbers is reduced to a finite one, namely rational numbers of length polynomial in the input. A direct algorithm for the controlled unfortunate-flow problem, thus one that does not rely on [17], is still open. Such a direct algorithm would reduce the real-number domain to a finite one in a tighter manner – one that depends on the network. We see several interesting problems in this direction, in particular finding a *sufficient granularity* that a regulator may need, and *bounding the non-optimality* caused by integral regulators. Similar problems are open in the settings of flow games with two or more players [14, 12].

Finally, the unfortunate-flow problem sets the stage to problems around network design, where the goal is to design networks with maximal unfortunate flows. In particular, in *network repair*, we are given a network and we are asked to modify it in order to increase its unfortunate flow value. Different algorithms correspond to different types of allowed

modifications. For example, we may be allowed to change the capacity of a fixed number of edges. Note that unlike the case of maximal flow, here a repair may reduce the capacity of edges. Also, unlike the case of maximal flow, there is no clear theory of minimal cuts that may assist us in such a repair.

References

1 R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: Theory, algorithms, and applications*. Prentice Hall Englewood Cliffs, 1993.

2 C. Bazgan, M. Chopin, M. Cygan, M.R. Fellows, F. V. Fomin, and E. Leeuwen. Parameterized complexity of firefighting. *Journal of Computer and Systems Science*, 80(7):1285–1297, 2014.

3 J. Choudhari, A. Dasgupta, N. Misra, and M.S. Ramanujan. Saving critical nodes with firefighters is FPT. In *Proc. 44th Int. Colloq. on Automata, Languages, and Programming*, volume 80 of *LIPIcs*, pages 135:1–135:13, 2017.

4 T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990.

5 E.A. Dinic. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Math. Doll*, 11(5):1277–1280, 1970. English translation by RF. Rinehart.

6 J. Edmonds and R.M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

7 S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

8 L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.

9 L.R. Ford and D.R. Fulkerson. *Flows in networks*. Princeton Univ. Press, Princeton, 1962.

10 A.V. Goldberg, É. Tardos, and R.E. Tarjan. Network flow algorithms. Technical report, DTIC Document, 1989.

11 A.V. Goldberg and R.E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988.

12 S. Guha, O. Kupferman, and G. Vardi. Multi-player flow games. In *Proc. 17th International Conference on Autonomous Agents and Multiagent Systems*, 2018.

13 S. Keren, A. Gal, and E. Karpas. Goal recognition design for non optimal agents. In *Proc. 29th AAAI conference*, pages 3298–3304, 2015.

14 O. Kupferman, G. Vardi, and M.Y. Vardi. Flow games. In *Proc. 37th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 93 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:38–38:16, 2017.

15 A. Madry. Computing maximum flow with augmenting electrical flows. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 593–602. IEEE, 2016.

16 L. Qingsong, G. Betsy, and S. Shashi. Capacity constrained routing algorithms for evacuation planning: A summary of results. In *International Symposium on Spatial and Temporal Databases*, pages 291–307. Springer, 2005.

17 E.D. Sontag. Real addition and the polynomial hierarchy. *Information Processing Letters*, 20(3):115–120, 1985.

18 É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

Spanning Trees With Edge Conflicts and Wireless Connectivity

Magnús M. Halldórsson
School of Computer Science, Reykjavik University, Iceland
mmh@ru.is

Guy Kortsarz
Rutgers University, Camden, NJ, USA
guyk@camden.rutgers.edu

Pradipta Mitra
Google Research, New York, USA
ppmitra@gmail.com

Tigran Tonoyan
School of Computer Science, Reykjavik University, Iceland
ttonoyan@gmail.com

Abstract

We introduce the problem of finding a spanning tree along with a partition of the tree edges into fewest number of feasible sets, where constraints on the edges define feasibility. The motivation comes from wireless networking, where we seek to model the irregularities seen in actual wireless environments. Not all node pairs may be able to communicate, even if geographically close – thus, the available pairs are specified with a link graph $\mathcal{L} = (V, E)$. Also, signal attenuation need not follow a nice geometric formula – hence, interference is modeled by a conflict (hyper)graph $\mathcal{C} = (E, F)$ on the links. The objective is to maximize the efficiency of the communication, or equivalently, to minimize the length of a schedule of the tree edges in the form of a coloring.

We find that in spite of all this generality, the problem can be approximated linearly in terms of a versatile parameter, the inductive independence of the interference graph. Specifically, we give a simple algorithm that attains a $O(\rho \log n)$-approximation, where n is the number of nodes and ρ is the inductive independence, and show that near-linear dependence on ρ is also necessary. We also treat an extension to Steiner trees, modeling multicasting, and obtain a comparable result.

Our results suggest that several canonical assumptions of geometry, regularity and "niceness" in wireless settings can sometimes be relaxed without a significant hit in algorithm performance.

2012 ACM Subject Classification Theory of computation → Routing and network design problems, Networks → Network design and planning algorithms

Keywords and phrases spanning trees, wireless capacity, aggregation, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.158

Related Version A full version of the paper is available at `https://arxiv.org/abs/1803.04578`.

Funding The first and last authors are supported by grants nos. 152679-05 and 174484-05 from the Icelandic Research Fund. The second author is supported by NSF grant 1540547.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 158; pp. 158:1–158:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

We introduce the problem of finding a spanning tree along with a partition of the tree edges into fewest number of feasible sets, which are independent sets in a given conflict (hyper)graph. The motivation comes from wireless networking, where we seek a basic communication structure while capturing the irregularities seen in actual wireless environments.

A spanning tree is the minimal structure for connecting the given set of nodes into a mutually communicable network. The *cost* of a communication spanning tree is the *time* required to *schedule* all the tree edges – the transmission *links* – while obeying the *interference* caused by simultaneous transmissions. The *scheduling complexity* of the tree represents its throughput capacity: how much communication can be sustained in the long run. The task might be to *aggregate* the data measured at the sensor nodes, or to *broadcast* using one-to-one communication to all nodes of the network.

Algorithmic studies of wireless connectivity to date have generally involved strong "niceness" assumptions. One core assumption is that points are located in the Euclidean plane and *all (close enough) pairs of nodes are available as links* for use in the spanning tree. Interference modeling has become progressively more realistic, starting with range-based graph models to the fractional *SINR* model of interference, but the common thread is that *interference is a direct function of the geometry.* While natural, these assumptions depend on a simplified view of the nature of wireless communication.

Wireless networking in the real world behaves quite different from these theoretical models [10, 32, 38] and typically displays a high degree of irregularity. This manifests in how the strength of signals (and the corresponding interference) often varies greatly within the same region, and is often poorly correlated with distance [2]. This behavior holds even in simple outdoor environments, but is magnified inside buildings. It is also evidenced by fluctuations, sensitivity to environmental changes (even levels of humidity), and hard-to-explain unreliability.

There has been increased emphasis for greater robustness in the design and analysis of wireless algorithms to address the observed irregularities. In the world of communications engineering, the default is to introduce stochastic distributions, e.g., on signal strengths. The algorithms world prefers more adversarial effects, but that can easily lead to intractability.

The objective of this work is to embrace this irregularity in connectivity problems. We replace the previous assumptions by the opposite premises:

A link may not be usable even if it should be.

and

Interference need not follow (or even relate to) the underlying geometry.

Technically, the former premise means that the set of usable or *available links* is now given as a link graph $\mathcal{L} = (V, L)$. We place no restrictions on the structure of this graph. The second premise implies another graph, this time on top of the *links*. Namely, the conflict (hyper)graph $\mathcal{C} = (L, F)$ specifies whether a given pair of links in L can coexist in the same color (of a spanning tree). In the Connectivity Scheduling problem, we seek a spanning tree T of $\mathcal{L}$ and a coloring of the links of T minimizing the number of colors used.

These formulations naturally raise a number of questions: Can arbitrary sets of available/usable links actually be handled effectively? Can we disconnect the conflicts/interference from the geometry? Since the ugly specter of intractability is bound to raise its head somewhere, what are minimal restrictions that keep these problems well-approximable?

Our Results. Given the generality of the ConnectivityScheduling problem, it is unsurprising that it is very hard even to approximate. We show that strong $n^{1-\epsilon}$-approximation hardness holds, even for the natural special case of 2-hop interference. Instead, we aim to obtain approximations in terms of natural instance parameters.

We show that the problem is approximable within $O(\rho \log n)$-factor, where ρ is the *inductive independence* of the (fractional) conflict graph. This is particularly relevant since ρ is known to be constant in both of the predominant interference models: the physical (or SINR) model, and the protocol model. This is attained by a simple greedy algorithm that can be viewed as a combination of Kruskal's MST algorithm and a link scheduling algorithm for the physical model. In contrast, we find that the (perhaps more natural) approach of selecting and coloring an MST fails badly.

We also generalize the problem to Steiner trees and obtain a similar logarithmic approximation. The results carry over to the SINR model, where we obtain the first results that hold for general metric spaces. We also give a supplementary result in the full version (for space reasons), involving natural geometric interference assumptions, namely that all links of length smaller than a threshold are available.

Definitions. In line with a modern view of wireless interference, we represent the interference conflicts by a fractional conflict graph $\mathcal{C} = (L, W)$. Here L is the set of communication links and $W : L \times L \to \mathbb{R}^+$ is a function on ordered pairs of links, where $W(e, f)$ represents (or approximates) the degree to which a transmission on link e interferes with a transmission on link f. Of particular interest are functions W in terms of geometric relationships involving link lengths and distances between links. Note that W may be asymmetric. For convenience, let $W(e, e) = 0$. We shall write $W(S, e) = \sum_{f \in S} W(f, e)$ and $W(f, S) = \sum_{e \in S} W(f, e)$. Let $\mathcal{C}[Y] = (Y, W\restriction_Y)$ denotes the subgraph induced by a given subset $Y \subseteq L$.

A set S of links is an independent or a feasible set if $W(S, e) \leq 1$, for all $e \in S$. A coloring of $\mathcal{C} = (L, W)$ is a partition of L into independent sets. Observe that when W is a 0-1 function, we have the usual independent sets and colorings of graphs. Also, the formulation with fractional conflicts corresponds to hypergraphs that contain a hyperedge for each minimal set S' where $W(S', e) \geq 1$ holds for some $e \in S'$.

We can now state our Connectivity Scheduling problem formally:

> Given a link graph $\mathcal{L} = (V, L)$ and a fractional conflict graph $\mathcal{C} = (L, W)$, we seek a spanning tree T of $\mathcal{L}$ and a coloring of $\mathcal{C}[T]$, using the fewest number of colors.

A fractional conflict graph $\mathcal{C} = (L, W)$ is said to be ρ-inductive independent, w.r.t. an ordering $\prec$ of the links, if for every link e and every feasible set I with $e \prec I$, $W(I, e) + W(e, I) \leq \rho$, where $e \prec I$ means that e precedes each link in I. Here, "inductive" refers to how the interference is measured only towards later links, and "independence" that it is towards independent sets. In geometric settings (including range-based and SINR models), $\prec$ corresponds to a non-decreasing ordering by link length.

For a fractional conflict graph $\mathcal{C} = (L, W)$, let $\chi(\mathcal{C})$ denote the smallest number of independent sets into which L can be partitioned; when $\mathcal{C}$ is an ordinary graph, $\chi(\mathcal{C})$ is the chromatic number of $\mathcal{C}$. We view a coloring of $\mathcal{C}$ also as a schedule and refer to the colors also as slots (which could be time slots or frequency bands).

Notable Instantiations. Connectivity Scheduling has a number of special cases of independent interest, both graph-based and geometric:

- A well-studied setting is where two links conflict if they are incident on a common link, i.e., when $\mathcal{C}$ is the square of the line graph of the link graph $\mathcal{L}$. This case corresponds to bidirectional version of the classic *radio network* model. The directed version of Connectivity Scheduling was treated in [9] as the *radio aggregation scheduling problem*.
- In *range-based or disk models*, nodes are embedded in the plane and two links are adjacent if the distance between (the closest points on) them is less than K times the length of the longer link, where K is some fixed constant. In a variant, the condition is on distances between particular nodes on the links. Also, in the the related *protocol* model, adjacency occurs if the distance is less than K_1 times the length of the longer link plus K_2 times the length of the shorter link, for some constants K_1, K_2.
- The original driving motivation is when nodes and links are embedded in a metric space and the fractional conflicts follow the *geometric SINR model* of interference in terms of the lengths and distances between links. Before this work, only the case when $\mathcal{L}$ is the complete graph over a set of points in a Euclidean metric was considered.
- A different geometric version is when we view that no signal gets transmitted between nodes on unavailable links, perhaps due to an obstacle. The links are then unavailable, but the nodes also don't interfere with each other. We refer to this as the Missing Links version.
- A natural special case occurs when link unreliability is restricted by link length, so that only reasonably long links are unavailable or attenuated, but short links follow the normal SINR laws (short links are reliable). This is treated in the full version of this paper.
- Finally, when the conflict graph $\mathcal{C}$ is the *line graph* of the link graph $\mathcal{L}$, i.e., $\mathcal{C} = L(\mathcal{L})$, we obtain the well-known *minimum degree spanning tree* (MDST) problem, where given a graph $\mathcal{L}$, the goal is to find a spanning tree of smallest maximum degree. By König's theorem, the chromatic number of the line graph of a tree (in fact, of any bipartite graph) is equal to the maximum degree of the tree. This problem has more structure that allows for better solution: while it is NP-hard, it can be approximated within an additive one [8]. In particular, $L(\mathcal{L})$ is claw-free (does not contain an induced star graph $K_{1,3}$), which is stronger than being 2-inductive independent), and is intimately related to $\mathcal{L}$.

Related Work. The connectivity problem in the geometric SINR model was first considered by Moscibroda and Wattenhofer [35]. It was, in fact, the first work on worst-case analysis in the SINR model. They show that unlike in random networks, the worst-case connectivity depends crucially on the use of power control, and with optimal power control, $O(\log^4 n)$ slots suffice to connect the nodes. They soon improved this to $O(\log^2 n)$ [36, 34]. Currently, the best upper bounds known are $O(\log n)$ [17] and $O(\log^* \Lambda)$ [23], where Λ is the ratio between the longest to the shortest length of a link in a minimum spanning tree (MST), a structural parameter that is independent of n. Both of these results hold for the MST of the pointset; there are pointsets where $\Omega(\log^* \Lambda)$ slots are necessary for scheduling an MST [21].

The scheduling complexity of connectivity relates closely to the efficiency of *aggregation*, a key primitive for wireless sensor networks. We refer the reader to [26] for bibliography on aggregation/collection problems.

There are many approaches that have been proposed to model irregularity in wireless networks. We first examine static cases, or the modeling of non-geometric behavior. The basic SINR model allows the pathloss constant α to be adjusted [13], giving a first-order approximation of the signal gain. In the engineering community, it is most common to assume that the deviations are drawn from a particular stochastic distribution, typically

assuming independence of events. In the TCS camp, the prevailing approach is to view the variations as conforming the plane into a non-Euclidean metric space [7, 16], while retaining some tractable characteristics. This can also entail identifying appropriate parameters [4].

For frequent temporal changes, the standard engineering assumption is Rayleigh fading. Dams et al. [6] (see also [22]) showed that link scheduling algorithms are not significantly affected by such variation, assuming independence across time.

For unpredictably changing behavior, there is much research on adapting to new conditions, particularly with exponential backoff. A theoretic model proposed to specifically capture unreliability is the *dual graph model* [33], which extends the radio network model to a pair of graphs, the reliable and the unreliable links, where the latter are under adversarial control. The focus there is on distributed algorithms for one-shot problems, like global and local broadcast problems, where the nodes do not know which links are reliable. As far as we know, it has not been considered in settings involving a long-term communication structure.

Inductive independence was first defined by [1] and studied by [37] in the graph setting, while the weighted version was introduced by Hoefer and Kesselheim [25]. It has been used as a performance measure for various problems related to wireless networks, including admission control [11], dynamic packet scheduling [31, 15], and spectrum auctions [25, 24, 15].

Outline of the paper. We first examine, in Sec. 2, how the standard approach – finding a minimum spanning tree – fares for our problem, and show that it can give poor solutions in every known interference model when there are missing or unreliable links. We then give in Sec. 3 a greedy algorithm for Connectivity Scheduling achieving $O(\rho \log n)$-approximation, where ρ is the inductive independence number of the conflict graph. This dependence on ρ is shown to be essentially tight in Sec. 5. We also obtain a similar approximation of a *Steiner or multicast* version of the problem in Sec. 4. Implications of our results to the SINR (or physical) model are given in Sec. 6. The rest of the paper can safely be read without any background in that model. We then close with open problems. Missing proofs, as well as a brief primer on SINR concepts, are given in the full version of the paper.

2 MST Fails

In a basic setting, the nodes are located on the plane, and the interference between two links is a function of the lengths of links (distance between the two end-nodes), and the distance between the (endpoints of) links. For instance, in the SINR model, the interference between two links is a decreasing function of their distance, and an increasing function of the length of the interfered link. In this setting, the Euclidean minimum spanning tree (MST) over the set of nodes is a natural candidate for connectivity, since it favors short links and has low degree (or, more generally, contains few links in the vicinity of any node). Indeed, the MST of n nodes can be scheduled in $O(\log n)$ slots in the Euclidean SINR model [17].

Somewhat surprisingly, we find that when the set of possible links is restricted, the MST can actually fail quite badly. This holds in every reasonable model of interference.

▶ **Interference Assumption 1.** *We say that an interference model is reasonable if: a) incident links cannot be scheduled together, while b) sparse instances of equal length links can be scheduled in $O(1)$ slots, where a set of length ℓ links is sparse if any ball of radius ℓ contains $O(1)$ endpoints of those links.*

Every geometrically-defined wireless interference model known satisfies this reasonableness property. In particular, this holds in the protocol and Euclidean SINR models.

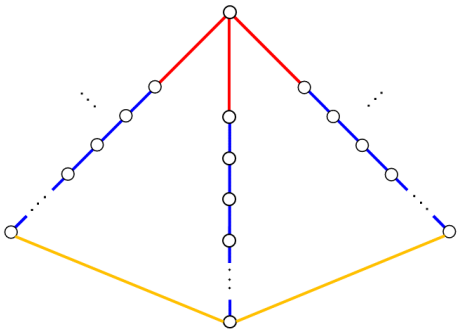


Figure 1 The construction from Thm. 1

▶ **Theorem 1.** *For any n, there is an instance of n nodes embedded on the plane together with a spanning tree that is schedulable in $O(1)$ slots while scheduling the MST requires $\Omega(n^{1/3})$ slots, in every reasonable interference model.*

Proof. Let $k \geq 1$ be a number and $K = 2k^2$. Let $V = \{o\} \cup \{v_{i,j} : i = 0, 1, \ldots, k-1, j = 0, 1, \ldots, K-1\}$ denote the set of $n = kK + 1 = 2k^3 + 1$ nodes. We position the nodes in the plane using polar coordinates, with the node o as the origin. For node $v_{i,j}$, angular coordinate $r_{i,j}$ is $2\pi \cdot i/k$, while its radial coordinate is $k + j$.

The links are given by $L = O \cup T \cup Y$, where $O = \{(o, v_{i,1}) : i = 0, \ldots, k-1\}$, $T = \{(v_{i,j}, v_{i,j+1}) : i = 0, \ldots, k-1, j = 0, \ldots, K-2\}$, $Y = \{(v_{i,K-1}, v_{i+1 \bmod k,K-1}) : i = 0, \ldots, k-1\}$, or the *ordinary*, the *tiny* and the *yuge* links. That is, the link graph is in the form of a wheel, centered at origin, with k spokes, and K nodes on each spoke (see Fig. 1). Ordinary links are incident with the origin, while the yuge links form the tire of the wheel.

We observe that $d(v_{i,K-1}, v_{i+1 \bmod k,K-1}) > k = d(o, v_{i',1})$, for any i, i'. Thus, the MST consists of the ordinary and tiny links, $S \cup L$. Since all the ordinary links have an endpoint in the origin, they must all be scheduled in different slots, implying that the MST requires $k = \Theta(n^{1/3})$ slots. On the other hand, a more efficient solution is to use the set Q, consisting of T, Y and one (arbitrary) link from O. This set Q is a union of three sparse subsets, and therefore can be scheduled in $O(1)$ slots. ◀

This same example shows why the known results for Euclidean SINR do not carry over to general metric spaces (even without missing links). Namely, one could simply form a metric space on the n nodes by shortest-path distances in the link graph.

3 Greedy Algorithm

A natural greedy approach is to find a large feasible subset of edges, assign it a fresh color, contract it, and iterate on the contracted graph. The key step is obtaining a constant-approximation for a maximum feasible subset. A logarithmic approximation then follows from a set cover argument.

We assume in this section that $\mathcal{L}$ can have parallel edges but no loops. We assume that the conflict graph $\mathcal{C}$ is ρ-inductive independent for a number $\rho > 0$, and the corresponding conflict function W and ordering of edges $\prec$ are given. In the maximum feasible forest problem, the goal is to find a maximum cardinality subset of edges of $\mathcal{L}$, which is both independent in $\mathcal{C}$ and acyclic in $\mathcal{L}$.

The algorithm, given as Alg. 1, is a greedy Kruskal-like algorithm that mixes the edge selection criteria of wireless capacity algorithms [16, 29] with the classic MST algorithm of

Kruskal, thus the name CapKruskal. It processes the edges in order of precedence $\prec$ and adds an edge to the forest if: a) the interference on that edge from previously selected edges is small, and b) the edge does not induce a cycle (as per Kruskal). We state it in terms of the classic union-find operations of MakeSet, Connected, and Union.

Algorithm 1 CapKruskal$(\mathcal{L}, \mathcal{C})$

MakeSet(v), for each $v \in V(\mathcal{L})$
$S \leftarrow \emptyset$
for $e = (u, v)$ in L in $\prec$ order **do**
  if $W(S, e) + W(e, S) \leq 1/2$ and **not** Connected(u, v) **then**
    $S \leftarrow S \cup \{e\}$
    Union(u, v)
  end if
end for
return $S' = \{e \in S : W(S, e) \leq 1\}$

Algorithm 2 Conn$(\mathcal{L}, \mathcal{C})$

$i \leftarrow 0$
$\mathcal{L}_0 \leftarrow \mathcal{L}$
while $\mathcal{L}_i$ has an edge **do**
  $S_i \leftarrow$ CapKruskal$(\mathcal{L}_i, \mathcal{C}[\mathcal{L}_i])$
  $\mathcal{L}_{i+1} \leftarrow Contract(\mathcal{L}_i, S_i)$
  $i \leftarrow i + 1$
end while
return $S_0, S_1, \ldots, S_{i-1}$

Recall that a subset S of edges in $\mathcal{L}$ is feasible if $W(S, e) = \sum_{f \in S} W(f, e) \leq 1$, for all $e \in S$. Define the ordered weight function W^+ as $W^+(e, f) = W(e, f)$ if $e \prec f$, and $W^+(e, f) = 0$, otherwise. Similarly, define W^- as $W^-(e, f) = W(e, f)$ if $f \prec e$, and $W^-(e, f) = 0$, otherwise. Also define the cumulative versions $W^+(S, e)$, $W^+(e, S)$ as before.

We say that a set S is semi-feasible if for each $e \in S$, $W^+(S, e) + W^-(e, S) \leq 1/2$. Namely, the weighted indegree from shorter nodes and to longer nodes is bounded, but the total indegree of e may not be. By an averaging argument, a semi-feasible set I contains a feasible subset of at least half its size. Indeed, using semi-feasibility and sum rearrangements, we have,

$$\sum_{e \in S} W(S, e) = \sum_{e \in S} \left(W^+(S, e) + W^-(e, S) \right) \leq \frac{|S|}{2} \tag{1}$$

so for at least half of the links $e \in S$ it holds that $W(S, e) \leq 1$.

▶ **Theorem 2.** *Let F be a maximum feasible forest of $\mathcal{L}$. Then* CapKruskal$(\mathcal{L}, \mathcal{C})$ *outputs a feasible forest of size $\Omega(|F|/\rho)$.*

Proof. Let S and S' be the sets computed in CapKruskal$(\mathcal{L}, \mathcal{C})$. By definition, S' is feasible. To argue that S' is large, we examine an arbitrary feasible forest, break it into three parts, and show that none of the parts can be too large compared to S'. This will hold, in particular, for the optimal feasible forest. By (1), we can focus on bounding $|S|$, as $|S'| \geq |S|/2$.

Let I be a feasible forest. Observe that the selection condition of the algorithm is equivalent to $W^+(S, e) + W^-(e, S) \leq 1/2$, since the edges are considered in the order of $\prec$. Let I_R be those edges e in I that failed the degree condition ($W^+(S, e) + W^-(e, S) > 1/2$), and I_T those edges $e = (u, v)$ in I that failed the connectivity condition (Connected(u, v)). The rest, $I_S = I \setminus (I_R \cup I_T)$ are contained in S. We bound these sets in terms of S.

Since I_T contains only edges inside components that S also connects (recalling that I induces a forest), $|I_T| \leq |S|$. Also, clearly $I_S \subseteq I \cap S \subseteq S$, so $|I_S| \leq |S|$. To bound the size of I_R, observe first that by the definition of ρ-inductive independence, $W^-(I_R, f) + W^+(f, I_R) \leq$

ρ, for every edge $f \in S$. This implies that

$$W^-(I_R, S) + W^+(S, I_R) = \sum_{f \in S} \left[W^-(I_R, f) + W^+(f, I_R)\right] \leq \rho \cdot |S|.$$

On the other hand, by the selection criteria,

$$W^+(S, I_R) + W^-(I_R, S) = \sum_{e \in I_R} \left[W^+(S, e) + W^-(e, S)\right] > \sum_{e \in I_R} \frac{1}{2} = \frac{|I_R|}{2}.$$

Thus, $|I_R| \leq 2\rho \cdot |S|$ and $|I| \leq (2\rho + 2)|S| \leq 4(\rho + 1)|S'|$. ◀

Connectivity Scheduling Algorithm. The algorithm Conn repeatedly calls CapKruskal to obtain a large independent set of links and assigns it to a new color class. These links are then contracted and the process repeated until we have obtained a spanning tree.

The contraction of an edge is defined in the standard way, except we discard loops. Note that *contraction leaves the conflict graph $\mathcal{C}$ intact.* The operation $Contract(\mathcal{L}, S)$ contracts all edges in S of a link graph $\mathcal{L}$ and outputs the resulting graph.

The pseudocode of the algorithm is given in Alg. 2. The proof of the following theorem, which is relegated to the full version, follows the classic set cover argument [27].

▶ **Theorem 3.** Conn *terminates in $O(\rho \log n) \cdot \chi$ rounds, where χ is the number of colors needed for coloring an optimum spanning tree.*

4 Multicast Tree Schedules

A natural generalization of Connectivity Scheduling is to allow for a set of optional nodes that can be used in the tree construction but need not. Formally, the node set V contains a subset X of terminals and we seek a Steiner tree that spans all the terminals. As before, we ask also for the shortest schedule of the tree links. We refer to this as the Steiner Connectivity Scheduling.

It is not hard to construct examples for which optimal multicast trees are arbitrarily better than trees that use only the terminals, even in a geometric setting. One instance can be obtained from the example of Sec. 2 by restricting the terminals to only the origin and the nodes incident on yuge links.

We give an algorithm for Steiner Connectivity Scheduling with *unweighted* conflict graph $\mathcal{C}$, and analyse it in terms of a parameter similar to ρ but involving clique covers rather than independence (this setting is also applicable to the SINR model, see Sec. 6). An unweighted graph $\mathcal{C} = (L, E)$ is η-simplicial if there is an ordering $\prec$ of L such that for each link $v \in L$, the subgraph induced by v's neighbors that are later in the ordering can be covered with η cliques. We refer to neighbors later in the ordering as post-neighbors. As before, in the geometric setting, the ordering is given by link length. Observe that $\rho \leq \eta$, while the best bound in the other direction is $\eta \leq \rho \log n$.

Our algorithm is a reduction to a multi-dimensional version of the Steiner tree (MMST) problem, recently treated by Bilò et al. [3]. In MMST, each edge of the input graph has an associated d-dimensional weight vector, where the weight of edge e along dimension i indicates how much of the i-th *resource* is required by e. The objective is to find a tree that minimizes the ℓ_p-norm of its load vector, where the load vector of a Steiner tree is the sum of the weight vectors of its edges. We use the ℓ_∞-norm, as we want to minimize the maximum use of a resource. They give a greedy $O(\log d)$-approximation algorithm for that case.

Given an instance of Steiner Connectivity Scheduling with link graph $\mathcal{L}$ and conflict graph $\mathcal{C}$, our reduction is as follows. Each link e in $\mathcal{L}$ is itself (or corresponds to) a resource, so there are n (=number of edges) resources. The weight of link f along dimension e is 1 if f is a post-neighbor of e in the conflict graph $\mathcal{C}$, and 0 otherwise.

Suppose now that the MMST algorithm of [3] returns a tree T with ℓ_∞-norm Z. Then, the sum of the tree edges along each dimension is at most Z, namely, each link (whether in T or not) has at most Z post-neighbors in T. In particular, $\mathcal{C}[T]$ is Z-inductive, and can then be colored greedily using $Z+1$ colors.

On the other hand, consider an optimal tree T^* and let Z^* denote the infinity norm of its load vector. From [3], we know that $Z = O(\log n) \cdot Z^*$. Let f be a link with Z^* post-neighbors in T^*, and let N_f be its set of post-neighbors in T^*. By assumption, $\mathcal{C}[N_f]$ can be covered with η cliques, and thus N_f contains a clique of size at least $|N_f|/\eta = Z^*/\eta$. It follows that the length of the schedule of the optimal tree is at least the chromatic number of $\mathcal{C}[N_f]$, which is at least Z^*/η. Thus, our solution yields a $O(\eta \log n)$-approximation.

▶ **Theorem 4.** *There is a $O(\eta \log n)$-approximation algorithm for* Steiner Connectivity Scheduling*, where conflicts are given by a η-simplicial graph.*

It is a folklore that $\eta \leq 6$ in disk graphs.

▶ **Corollary 5.** *There is a $O(\log n)$-approximation algorithm for* Steiner Connectivity Scheduling*, where conflicts are given by a disk graph.*

5 Hardness of Approximation

It is easy to see that with an arbitrary conflict graph $\mathcal{C}$, the problem is hard to approximate. For instance, if the link graph $\mathcal{L}$ is already a spanning tree, Connectivity Scheduling becomes simply the classical graph coloring problem (of $\mathcal{C}$). We show below that the hardness extends to other more restricted settings. These results also show that near-linear dependence on ρ, the inductive independence, is unavoidable.

We first show that hardness holds when $\mathcal{C}$ is the square of the line graph of $\mathcal{L}$ (for general $\mathcal{L}$), $\mathcal{C} = L^2(\mathcal{L})$, then extend the construction to the case when $\mathcal{L}$ is a complete graph and $\mathcal{C}$ is general (Thm. 7), and to signal strength models (Sec. 6). This corresponds to (bidirectional) 2-hop interferences: two transmission links conflict if they are incident on a common edge. The reduction is from the Distance-2 Edge Coloring problem in general graphs, also known as Strong Edge Coloring: Given a graph $\mathcal{L}$, find a partition of the edge set into induced matchings, i.e., induced subgraphs where every vertex is of degree 1.

▶ **Theorem 6.** *The* Connectivity Scheduling *problem is hard to approximate within a $n^{1-\epsilon}$-factor, for any $\epsilon > 0$, even when $\mathcal{C} = L^2(\mathcal{L})$.*

Proof. Given an instance of Strong Edge Coloring with graph $G' = (V', E')$, we construct an instance of Connectivity Scheduling problem with the graph $\mathcal{L}$ constructed as follows. Consider a bipartite graph $G'' = (V_1, V_2, E)$, as follows. For each vertex v in V', there are two vertices v_1, v_2 in $V = V_1 \cup V_2$, where $v_i \in V_i$, $i = 1, 2$. If $uv \in E'$ then v_1u_2 and v_2u_1 are in E. Link graph $\mathcal{L}$ is obtained from G'' by taking a complete binary tree with $|V_2|$ leaves and identifying each leaf with a vertex of V_2. The conflict graph is given by a simple graph $\mathcal{C}$ with vertex set E, where $e_1, e_2 \in E$ are adjacent in $\mathcal{C}$ if and only if they form an induced matching in G', i.e., there is no edge in G' connecting an endpoint of e_1 to an endpoint of e_2. This completes the construction.

First, let us show that a strong edge coloring of G' can be used to construct a spanning tree in $\mathcal{L}$ with a similar coloring number. Consider a strong coloring that partitions the edges of G' into c color classes $E_1, E_2, \ldots, E_c$. Each class E_i induces a pair of feasible slots S_i, S'_i in $\mathcal{L}$, where $S_i = \{v_1u_2 : uv \in E_i\}$ and $S'_i = \{u_1v_2 : uv \in E_i\}$. Indeed, since E_i is an induced matching in G', each of the slots S_i, S'_i is also an induced matching in $\mathcal{L}$ (and hence independent in $\mathcal{C}$). Note that the edges in these slots cover all vertices of $\mathcal{L}$, except for the binary tree. We also add $O(\log n)$ slots to the schedule, two for each layer in the binary tree. The number of slots used then is $O(c + \log n)$. This gives us a connected subgraph of $\mathcal{L}$ that can be scheduled in $O(c + \log n)$ slots.

Next, consider a spanning tree of $\mathcal{L}$ with a corresponding schedule of the edges in slots $S_1, S_2, \ldots, S_t$. Ignoring all edges within the binary tree, we obtain a partition of the edges of the bipartite graph G'' between V_1 and V_2. We claim that each class corresponds to an induced matching in G', leading to a strong edge coloring of G' with t colors.

Consider a pair of edges v_1u_2 and w_1x_2 in the same feasible slot. Since they are feasible, there are no edges v_1x_2 nor w_1u_2 in $\mathcal{L}$, and thus no edges vx nor wu in E. Then vu and wx form an induced matching in G'.

Hence, the optimum number of colors in strong edge coloring of G' is within a constant factor plus a logarithmic term of the optimal number of slots needed for scheduling a spanning tree in $\mathcal{L}$. Since the former is hard to approximate within $n^{1-\epsilon}$-factor [5], so is the latter. ◀

▶ **Theorem 7.** *For general graphs $\mathcal{C}$, the* Connectivity Scheduling *problem is hard to approximate within a $n^{1-\epsilon}$-factor, for any $\epsilon > 0$, even if the link graph $\mathcal{L}$ is complete.*

Proof. We modify the instance of Thm. 6, by adding to $\mathcal{L}$ all edges that were not there and make them adjacent (in $\mathcal{C}$) to *all* other edges in the graph. If these new edges are used in a spanning tree, they have to be scheduled separately in individual time slots. Thus, using them can only increase the length of any schedule. ◀

6 Implications to Signal Strength Models

We consider in this section the implementation and implication of our results to signal strength models, most importantly metric SINR model.

SINR-feasibility, besides the underlying metric, also depends on the *transmission power control* regime. Different power control methods give different notions of feasibility. Nevertheless, it is known that for most interesting cases, SINR-feasibility has constant-inductive independence property. In particular, power control is usually split into two modes: fixed monotone power schemes, where links use only local information, such as the link length, to define the power level, and global power control, where all power levels are controlled simultaneously to give larger independent sets. The former includes the uniform power mode, where all links use equal power. Another technical issue is *directionality* of links, which is not explicitly addressed by our general results, but will be addressed below.

Let us start the discussion from Euclidean metrics (or more generally doubling metrics). For the global power control mode, [29] introduced a weight function W and proved that with this function, the conflict graph of any set of links is constant-inductive independent (see [29, Thm. 1]), so our results apply here directly (except for directionality issues, addressed below). Similarly, for fixed monotone power schemes (excluding uniform power), [15] showed that in order to get constant-inductive independence, one may take the natural weight function, affectance (also called *relative* or normalized interference) [15, Thm.3.3]. In all cases, the ordering $\prec$ corresponds to a non-decreasing order of links by length.

For general metric spaces, a slightly more technical definition of inductive independence is used, where a fractional conflict graph $\mathcal{C} = (L, W)$ is (ρ, γ)-inductive independent, w.r.t. an ordering $\prec$ of the links, if for every link e and every feasible set $I \in \mathcal{F}$ with $e \prec I$, there is a subset $I' \subseteq I$ of size $|I'| \geq |I|/\gamma$, such that $W(I, e) + W(e, I) \leq \rho$. The old definition corresponds to the setting $\gamma = 1$. It is easily verified that Thms. 2 and 3 extend to cover this new definition, with approximation ratios multiplied by a factor of γ. Now, the counterparts of the results from the previous paragraph in general metrics can be found in [18, Lemmas 2,4] and [30, Thm. 1, Lemma 3], where it is shown that with appropriate weight functions, feasibility for any fixed monotone power scheme (including uniform power), as well as feasibility with global power control, can be expressed by a fractional conflict graph, which is $(O(1), O(1))$-inductive independent.

The claims above concern settings where the links have fixed directions. In particular, if we apply Thm. 3 to the weighted functions from the previous paragraph, then we should add "there exists a direction of links, such that..." to the claim. This issue is easily resolved for the global power control mode, where the weight function of [29] does not depend on directions. Namely, it gives a schedule, such that whatever direction is assigned to the links, one can find a power assignment that makes it work (the power assignment could be different for different orientations of links).

For oblivious powers, the following trick applies. It is known that for a set of links with some direction and an oblivious power assignment, and with the weight function W defined in terms of the affectances, if $W(e, S) \leq 1/2$ for all $e \in S$ (call this dual-feasibility), then there is another oblivious power assignment (called the *dual* of the original one) that makes S feasible with the reversed directions of links [28]. Thus, we would like to have schedules with slots S being also dual-feasible. To this end, it is enough to modify CapKruskal, so that the threshold $1/2$ in the acceptance condition is replaced with $1/4$, and the output set S' is given by $S' = \{e \in S : (W(S, e) \leq 1) \wedge (W(e, S) \leq 1/2)\}$. Very similar methods then show that this again gives an $O(\rho)$-approximation to the maximum feasible forest problem. The rest of the analysis is left intact, so we obtain an $O(\log n)$-approximation as before, but with schedule slots that are both feasible and dual-feasible. Then we can replace each slot with its two copies and revert the directions of links in one of the copies. Every link thus gets scheduled in both directions, while the schedule length increases by a factor of two.

Summarizing the observations above, we state the following theorem.

▶ **Theorem 8.** *There is an $O(\log n)$-approximation to* Connectivity Scheduling *problem in the SINR model in arbitrary metric spaces. This holds both in the case of fixed monotone power assignments, and for arbitrary power control. It holds even when only a subset of the node-pairs are available as links (but interferences follow the metric SINR definitions).*

These are the first results that hold in general metrics. They are necessarily relative approximations, since in general metric spaces, there is no good upper bound on the connectivity number, even for complete graphs. Two simple examples are the metric induced by the star $K_{1,t}$ with unit-length edges, and the unit metric formed by distances on the unit-length clique metric.

For the case of points in the plane (i.e., a complete link graph with conflicts induced by distances), connectivity can be achieved in $O(\log n)$ slots [17]. Since it is not known if $O(1)$ slots always suffice, this result is not directly implied by Thm. 3. However, it was also shown in [17] that the MST contains a feasible forest of $\Omega(n)$ edges. The rest of our analysis (using constant-inductive independence) then implies a result matching [17].

▶ **Corollary 9.** *Let P be a set of points in the plane. Then,* Conn *finds and schedules a spanning tree of P in $O(\log n)$ slots.*

Steiner trees. In the geometric SINR model with a fixed monotone power scheme (with not all links available), we reduce the problem to a graph question as follows. It was observed in [14] that links of the same length class behave approximately like unit-disk graphs, where a length class refers to links whose lengths differ by at most a factor of 2. Namely, there are constants c_1 and c_2 such that for a set S of links of length approximately ℓ, if all links are of mutual distance greater than $c_2\ell$, then they form a feasible set, whereas any pair of links in S of distance at most $c_1\ell$ must be scheduled separately.

We modify the reduction to MMST to that of the graph construction so that weight of link f along dimension e is 1 only if f is a post-neighbor of e in $\mathcal{C}$ *and* f and e are of the same length class. We then take the resulting tree and schedule the length classes separately, at an extra cost of $O(\log \Lambda)$ (the number of length classes).

▶ **Corollary 10.** *There is a $O(\log \Lambda \log n)$-approximation algorithm for* Steiner Connectivity Scheduling *in the geometric SINR model, under any fixed monotone power scheme.*

Using power control, we can do considerably better. The main result of [19] shows that for any set L of links, there is an unweighted conflict graph $\mathcal{C}(L)$, such that every independent set in $\mathcal{C}$ is feasible, and the chromatic number of $\mathcal{C}$ is at most $O(\log^* \Delta)$ factor away from the optimum schedule length of L (using global power control). Moreover, $\mathcal{C}$ is constant-simplicial [19, Prop. 1].

▶ **Corollary 11.** *There is a $O(\log n \log^* \Lambda)$-approximation algorithm for* Steiner Connectivity Scheduling *in the geometric SINR model with global power control.*

A similar result with $O(\log \log \Lambda)$-factor holds also for certain monotone power schemes (but not, for instance, uniform power) [20].

Hardness. A special Missing Links variant of the geometric case is where the nodes/links are embedded in the plane and all interferences are either zero or follow the SINR model (with either fixed power or global power control).

▶ **Theorem 12.** *The geometric Missing Links variant is $n^{1-\epsilon}$-hard to approximate, for any $\epsilon > 0$. It is also $\Lambda^{2-\epsilon}$-hard, where Λ is the ratio between the longest to the shortest node distance. This holds even if all unavailable links are missing links.*

Proof. We embed the instance of Thm. 6 in the plane. The nodes of V_1 are located in a unit square in a mesh pattern, $1/\sqrt{n}$ apart in $\sqrt{n}$ columns $\sqrt{n}$ abreast. At a unit distance, a similar unit square holds the nodes of V_2. The length of an edge in $\mathcal{L}$ (in distance in the plane) is then between 1 and 4.

An induced matching in $\mathcal{L}$ corresponds to a set of links with no mutual interference. On the other hand, a pair of links that are incident on a common edge or share a vertex, will receive interference from each other according to the SINR formula (using the shared edge or each other). Given that distances along available edges vary only by a constant factor, the interference between the links is a constant (specifically, at least $1/4^\alpha$, where α is the "pathloss" constant of the SINR model). Thus, in the setting where the SINR threshold is at least the reciprocal of that constant (i.e., $\beta \geq 4^\alpha$), feasible sets are necessarily induced matchings in $\mathcal{L}$. We can then conclude by recalling a "signal-strengthening" result [12] that shows that varying the threshold by a constant factor only affects the schedule length by a constant factor.

The longest node distance is at most $\log n$, which is from the root of the binary tree to its leaves, while the shortest distance is $1/\sqrt{n}$. Thus, $\Lambda \leq 4\sqrt{n}\log n$, and $n^{1-\epsilon} \geq \Lambda^{2-\epsilon'}$, for some $\epsilon' \geq \epsilon/3$.

We can restrict the available edges incident to (non-leaf) nodes on the binary tree to the tree edges alone. Thus, non-leaf nodes in the tree must be connected via the tree edges. Then, all unavailable edges are missing edges. ◀

7 Open Issues

Many related problems are left addressing; we list the most prominent ones.

- Latency minimization: Bounding the time it takes for a packet to filter through the tree from a leaf to a root (and back). This requires optimizing both the height of the tree as well as the ordering of the links in the schedule.
- Directed case: Finding an arborescence. This requires new techniques, as our argument crucially depends on the graph being undirected.
- Distributed algorithms: This relates also to the issue of detecting or learning whether a link is usable/reliable or not.

References

1. Karhan Akcoglu, James Aspnes, Bhaskar DasGupta, and Ming-Yang Kao. Opportunity cost algorithms for combinatorial auctions. In *Computational Methods in Decision-Making, Economics and Finance*, pages 455–479. Springer, 2002.
2. Nouha Baccour, Anis Koubaa, Luca Mottola, Marco Antonio Zuniga, Habib Youssef, Carlo Alberto Boano, and Mario Alves. Radio link quality estimation in wireless sensor networks: a survey. *ACM Trans. Sensor Netw.*, 8(4):34, 2012.
3. Vittorio Bilò, Ioannis Caragiannis, Angelo Fanelli, Michele Flammini, and Gianpiero Monaco. Simple greedy algorithms for fundamental multidimensional graph problems. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 125:1–125:13, 2017.
4. Marijke H. L. Bodlaender and Magnús M. Halldórsson. Beyond geometry: towards fully realistic wireless models. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 347–356, 2014.
5. Parinya Chalermsook, Bundit Laekhanukit, and Danupon Nanongkai. Graph products revisited: Tight approximation hardness of induced matching, poset dimension and more. In *SODA*, pages 1557–1576. SIAM, 2013.
6. Johannes Dams, Martin Hoefer, and Thomas Kesselheim. Scheduling in wireless networks with Rayleigh-fading interference. *IEEE Transactions on Mobile Computing*, 14(7):1503–1514, 2015.
7. A. Fanghänel, T. Kesselheim, H. Räcke, and B. Vöcking. Oblivious interference scheduling. In *PODC*, pages 220–229, 2009.
8. Martin Fürer and Balaji Raghavachari. Approximating the minimum-degree steiner tree to within one of optimal. *Journal of Algorithms*, 17(3):409–423, 1994.
9. Rajiv Gandhi, Magnús M. Halldórsson, Christian Konrad, Guy Kortsarz, and Hoon Oh. Radio aggregation scheduling. In *Algosensors*, pages 169–182, 2015.
10. Deepak Ganesan, Bhaskar Krishnamachari, Alec Woo, David Culler, Deborah Estrin, and Stephen Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. Technical report, UCLA/CSD-TR 02, 2002.
11. Oliver Göbel, Martin Hoefer, Thomas Kesselheim, Thomas Schleiden, and Berthold Vöcking. Online independent set beyond the worst-case: Secretaries, prophets, and periods. In *ICALP*, pages 508–519, 2014.
12. Olga Goussevskaia, Magnús M Halldórsson, and Roger Wattenhofer. Algorithms for wireless capacity. *IEEE/ACM Transactions on Networking (TON)*, 22(3):745–755, 2014.

13 P. Gupta and P. R. Kumar. The Capacity of Wireless Networks. *IEEE Trans. Information Theory*, 46(2):388–404, 2000.

14 M. M. Halldórsson. Wireless scheduling with power control. *ACM Transactions on Algorithms*, 9(1):7, December 2012.

15 Magnús M. Halldórsson, Stephan Holzer, Pradipta Mitra, and Roger Wattenhofer. The power of oblivious wireless power. *SIAM J. Comput.*, 46(3):1062–1086, 2017.

16 Magnús M. Halldórsson and Pradipta Mitra. Wireless capacity with oblivious power in general metrics. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1538–1548, 2011.

17 Magnús M. Halldórsson and Pradipta Mitra. Wireless connectivity and capacity. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 516–526, 2012.

18 Magnús M. Halldórsson and Pradipta Mitra. Nearly optimal bounds for distributed wireless scheduling in the SINR model. *Distributed Computing*, 29(2):77–88, 2016. doi:10.1007/s00446-014-0222-7.

19 Magnús M. Halldórsson and Tigran Tonoyan. How well can graphs represent wireless interference? In *STOC*, pages 635–644, 2015.

20 Magnús M. Halldórsson and Tigran Tonoyan. The price of local power control in wireless scheduling. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, pages 529–542, 2015.

21 Magnús M. Halldórsson and Tigran Tonoyan. Optimal aggregation throughput is nearly constant. In *Mobihoc*, 2017. Poster abstract.

22 Magnús M. Halldórsson and Tigran Tonoyan. Wireless link capacity under shadowing and fading. In *MobiHoc*, pages 27:1–27:10, 2017.

23 Magnús M. Halldórsson and Tigran Tonoyan. Optimal aggregation throughput is nearly constant. In *38th IEEE International Conference on Distributed Computing Systems, July 2–5, 2018, Vienna, Austria*, 2018.

24 Martin Hoefer and Thomas Kesselheim. Secondary spectrum auctions for symmetric and submodular bidders. *ACM Transactions on Economics and Computation*, 3(2):9, 2015.

25 Martin Hoefer, Thomas Kesselheim, and Berthold Vöcking. Approximation algorithms for secondary spectrum auctions. *ACM Transactions on Internet Technology (TOIT)*, 14(2-3):16, 2014.

26 Özlem Durmaz Incel, Amitabha Ghosh, and Bhaskar Krishnamachari. Scheduling algorithms for tree-based data collection in wireless sensor networks. In *DCOSS*, pages 407–445. Springer, 2011.

27 David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9:256–278, 1974. doi:10.1016/S0022-0000(74)80044-9.

28 T. Kesselheim and B. Vöcking. Distributed contention resolution in wireless networks. In *DISC*, pages 163–178, August 2010.

29 Thomas Kesselheim. A constant-factor approximation for wireless capacity maximization with power control in the SINR model. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1549–1559, 2011.

30 Thomas Kesselheim. Approximation algorithms for wireless link scheduling with flexible data rates. In *ESA*, pages 659–670, 2012.

31 Thomas Kesselheim. Dynamic packet scheduling in wireless networks. In *PODC*, pages 281–290, 2012.

32 David Kotz, Calvin Newport, Robert S Gray, Jason Liu, Yougu Yuan, and Chip Elliott. Experimental evaluation of wireless simulation assumptions. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 78–82. ACM, 2004.

33 Fabian Kuhn, Nancy Lynch, Calvin Newport, Rotem Oshman, and Andrea Richa. Broadcasting in unreliable radio networks. In *PODC*, pages 336–345. ACM, 2010.

34 Thomas Moscibroda. The worst-case capacity of wireless sensor networks. In *IPSN*, pages 1–10, 2007.

35 Thomas Moscibroda and Roger Wattenhofer. The complexity of connectivity in wireless networks. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 23-29 April 2006, Barcelona, Catalunya, Spain*, 2006.

36 Thomas Moscibroda, Roger Wattenhofer, and Aaron Zollinger. Topology control meets SINR: The scheduling complexity of arbitrary topologies. In *MOBIHOC*, pages 310–321, 2006.

37 Yuli Ye and Allan Borodin. Elimination graphs. *ACM Trans. Algorithms*, 8(2):14:1–14:23, 2012.

38 Marco Zúñiga Zamalloa and Bhaskar Krishnamachari. An analysis of unreliability and asymmetry in low-power wireless links. *ACM Transactions on Sensor Networks (TOSN)*, 3(2):7, 2007.

Eigenvector Computation and Community Detection in Asynchronous Gossip Models

Frederik Mallmann-Trenn
CSAIL, MIT, US
mallmann@mit.edu

Cameron Musco
CSAIL, MIT, US
cnmusco@mit.edu

Christopher Musco
CSAIL, MIT, US
cpmusco@mit.edu

Abstract

We give a simple distributed algorithm for computing adjacency matrix eigenvectors for the communication graph in an asynchronous gossip model. We show how to use this algorithm to give state-of-the-art asynchronous community detection algorithms when the communication graph is drawn from the well-studied *stochastic block model*. Our methods also apply to a natural alternative model of randomized communication, where nodes within a community communicate more frequently than nodes in different communities.

Our analysis simplifies and generalizes prior work by forging a connection between asynchronous eigenvector computation and Oja's algorithm for streaming principal component analysis. We hope that our work serves as a starting point for building further connections between the analysis of stochastic iterative methods, like Oja's algorithm, and work on asynchronous and gossip-type algorithms for distributed computation.

2012 ACM Subject Classification Computing methodologies → Distributed algorithms

Keywords and phrases block model, community detection, distributed clustering, eigenvector computation, gossip algorithms, population protocols

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.159

Related Version A longer version of this paper with full proofs is available at [20], `https://arxiv.org/abs/1804.08548`.

Funding This work was supported in part by NSF Award Numbers BIO-1455983, CCF-1461559, CCF-0939370, and CCF-1565235. Cameron Musco was partially supported by an NSF graduate student fellowship.

1 Introduction

Motivated by the desire to process and analyze increasingly large networks – in particular social networks – considerable research has focused on finding efficient distributed protocols for problems like triangle counting, community detection, PageRank computation, and node centrality estimation. Many of the most popular systems for massive-scale graph processing, including Google's Pregel [19] and Apache Giraph [29] (used by Facebook), employ programming models based on the simulation of distributed message passing algorithms, in which each node is viewed as a processor that can send messages to its neighbors.

Apart from computational benefits, distributed graph processing can also be required when privacy constraints apply: for example, EU regulations restrict the personal data sent to countries outside of the EU [8]. Distributed algorithms avoid possibly problematic aggregation of network information, allowing each node to compute a local output based on their own neighborhood and messages received from their neighbors.

One of the main problems of interest in network analysis is the computation of the eigenvectors of a networks' adjacency matrix (or related incidence matrices, such as the graph Laplacian). The extremal eigenvectors have many important applications – from graph partitioning and community detection [13, 23], to embedding in graph-based machine learning [5, 26], to measuring node centrality and computing importance scores like PageRank [6].

Due to their importance, there has been significant work on distributed eigenvector approximation. In *synchronous* message passing systems, it is possible to simulate the well-known power method for iterative eigenvector approximation [17]. However, this algorithm requires that each node communicates synchronously with all of its neighbors in each round.

In an attempt to relax this requirement, models in which a subset of neighbors are sampled in each communication round [18] have been studied. However, the computation of graph eigenvectors in fully asynchronous and gossip-based message passing systems, in which nodes communicate with a single neighbor at a time in an asynchronous fashion, is not well-understood. While a number of algorithms have been proposed, which give convergence to the true eigenvectors as the number of iterations goes to infinity, strong finite iteration approximation bounds are not known [14, 24].

Our contributions

In this work, we give state-of-the-art algorithms for graph eigenvector computation in asynchronous systems with randomized schedulers, including the classic gossip model [7, 12] and population protocol model [2]. We show that in these models, communication graph eigenvectors can be computed via a very simple adaption of Oja's classic iterative algorithm for principal components analysis [27]. Our analysis leverages recent work studing Oja's algorithm for streaming covariance matrix eigenvector estimation [1, 16].

By making an explicit connection between work on streaming eigenvector estimation and asynchronous computation, we hope to generally expand the toolkit of techniques that can be applied to analyzing graph algorithms in asynchronous systems.

As a motivating application, we use our results to give state-of-the-art distributed community detection protocols, significantly improving upon prior work for the well-studied stochastic-block model and related models where nodes communicate more frequently within their community than outside of it. We summarize our results below.

Asynchronous eigenvector computation. First, we provide an algorithm (Algorithm 2) that approximates the k largest eigenvectors $\mathbf{v}_1, ..., \mathbf{v}_k$ for an arbitrary communication matrix (essentially a normalized adjacency matrix, defined formally in Theorem 1).

For an n-node network, the algorithm ensures, with good probability, that each node $u \in [n]$ computes the u^{th} entries of vectors $\tilde{\mathbf{v}}_1, ..., \tilde{\mathbf{v}}_k$ such that for all $i \in [k]$, $\|\tilde{\mathbf{v}}_i - \mathbf{v}_i\|_2^2 \leq \epsilon$. Each message sent by the algorithm requires communicating just $O(k)$ numbers, and the global time complexity is $\tilde{O}(\frac{\Lambda k^3}{\text{gap} \cdot \min(\text{gap}, \gamma_{mix})\epsilon^3})$ local rounds, where gap is the minimal gap between the k largest eigenvalues, γ_{mix} is roughly speaking the spectral gap, i.e., the difference between the largest and second-largest eigenvalue, and Λ is the sum of the k largest eigenvalues. We note that we use $\tilde{O}(\cdot)$ to suppress logarithmic terms, and in particular, factors of $\text{poly}\log n$. See Theorem 6 for a more precise statement.

For illustration, consider a communication graph generated via the stochastic block model – $G(n, p, q)$, which has n nodes, partitioned into two equal-sized clusters. Each intracluster edge added independently with probability p and each intercluster edge is added with probability $q < p$. If, for example, $p = \Omega\left(\frac{\log n}{n}\right)$ and $q = p/2$, and $k = 2$, we can bound with high probability $\Lambda = \Theta(1/n)$, gap $= \Theta(1/n)$, and $\gamma_{mix} = \Theta(1/n)$, which yields an eigenvector approximation algorithm running in $\tilde{O}(\frac{n}{\epsilon^3})$ global rounds, or $\tilde{O}(\frac{1}{\epsilon^3})$ local rounds.

Approximate community detection. Second, we harness our eigenvector approximation routine for community detection in the stochastic block model with connection probabilities p, q (we give two natural definitions of this model in an asynchronous distributed system with a random scheduler; see Theorem 2 and Theorem 3). After executing our protocol (Algorithm 5), with good probability, all but an ϵ fraction of the nodes output a correct community label in $\tilde{O}\left(1/\epsilon^3\rho^2\right)$ local rounds, where $\rho = \min\left(\frac{q}{p+q}, \frac{p-q}{p+q}\right)$. For example, when $q = p/2$, this complexity is $\tilde{O}\left(1/\epsilon^3\right)$. See Theorem 8 and Theorem 9 for precise bounds.

Exact community detection. Finally, we show how to produce an exact community labeling, via a simple gossip-based error correction scheme. For ease of presentation, here we just state our results in the case when $q = p/2$ and we refer to section 5 (Theorem 10 and Theorem 11) for general results. Starting from an approximate labeling in which only a small constant fraction of the nodes are incorrectly labeled, we show that, with high probability, after $O(\log n)$ local rounds, all nodes are labeled correctly.

Related work

Community detection via graph eigenvector computation and other spectral methods has received ample attention in centralized setting [22, 9, 32]. Such methods are known to recover communities in the stochastic block model close to the information theoretic limit. Interestingly, many state-of-the-art community detection algorithms in this model, which improve upon spectral techniques, are based on message passing (belief propagation) algorithms [11, 25]. However, these algorithms are not known to work in asynchronous contexts.

Community detection in asynchronous distributed systems has received less attention. It has recently been tackled in a beautiful paper by Becchetti et al. [3]. The algorithm studied in this paper is a very simple averaging protocol, originally considered by the authors in a synchronous setting [4]. Each node starts with a random value chosen uniformly in $\{-1, 1\}$. Each time two nodes communicate, they update their values to the average of their previous values. After each round of communication, a node's estimated community is given by the *sign* of the change of its value due to the averaging update in that round.

Beccheti et al. analyze their algorithm for *regular* clustered graphs, including regular stochastic block model graphs, where all nodes have exactly a edges to (randomly selected) nodes in their cluster and exactly $b < a$ edges to nodes outside their cluster. As discussed in [3], for regular graphs their protocol can be viewed as estimating the sign of entries in the second largest adjacency matrix eigenvector. Thus, it has close connections with our protocols, which explicitly estimate this eigenvector and label communitues using the signs of its entries.

The results of Becchetti et al. apply with $O(\text{polylog}\, n)$ local rounds of communication when either $\frac{a}{b} = \Omega(\log^2 n)$, or when $a - b = \Omega(\sqrt{a+b})$. In contrast, our results for the (non-regular) stochastic block model give $O(\text{polylog}\, n)$ local runtime when $\frac{p}{q} = \Omega(1)$ or $n(p-q) = \Omega(\sqrt{n(p+q)\log n})$. Here we assume that q is not too small – see Theorem 9 for

details. Note that $n \cdot p$ and $n \cdot q$ can be compared to a and b, since they are the expected number of intra- and inter-cluster edges respectively. Thus, our results give comparable bounds, tightening those of Becchetti et al. in some regimes and holding in the most commonly studied family of stochastic block model graphs, without any assumption of regularity[1].

Outside of community detection, our approach to asynchronous eigenvector approximation is related to work on asynchronous distributed stochastic optimization [31, 10, 28]. Often, it is assumed that many processors update some decision variable in parallel. If these updates are sufficiently sparse, overwrites are rare and the algorithm converges as if it were run in a synchronous manner. Our implementation of Oja's algorithm falls under this paradigm. Each update to our eigenvector estimates is sparse – requiring a modification just by the two nodes that communicate at a given time. In this way, we can fully parallelize the algorithm, even in an asynchronous system.

2 Preliminaries

2.1 Notation

For integer $n > 0$, let $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. Let $\mathbf{1}_{n,m}$ be an $n \times m$ all-ones matrix and $\mathbf{I}_{n \times n}$ be an $n \times n$ identity. Let $\mathbf{e}_i$ be the i^{th} standard basis vector, with length apparent from context. Let V denote a set of nodes with cardinality $|V| = n$. Let $\mathcal{P}$ be the set of all unordered node pairs (u, v) with $u \neq v$. $|\mathcal{P}| = \binom{n}{2}$.

For vector $\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\|_2$ is the Euclidean norm. For matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$, $\|\mathbf{M}\|_2 = \max_{\mathbf{x}} \frac{\|\mathbf{M}\mathbf{x}\|_2}{\|\mathbf{x}\|_2}$ is the spectral norm. $\|\mathbf{M}\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} \mathbf{M}_{i,j}^2}$ denotes the Frobenius norm. $\mathbf{M}^T$ is the matrix transpose of $\mathbf{M}$. When $\mathbf{M} \in \mathbb{R}^{n \times n}$ is symmetric we let $\lambda_1(\mathbf{M}) \geq \lambda_2(\mathbf{M}) \geq \ldots \geq \lambda_n(\mathbf{M})$ denote its eigenvalues. $\mathbf{M}$ is positive semidefinite (PSD) if $\lambda_i(\mathbf{M}) \geq 0$ for all i. For symmetric $\mathbf{M}, \mathbf{N} \in \mathbb{R}^{n \times n}$ we use $\mathbf{M} \preceq \mathbf{N}$ to indicate that $\mathbf{N} - \mathbf{M}$ is PSD.

2.2 Computational model

We define an asynchronous distributed computation model that encompasses both the well-studied population protocol [2] and asynchronous gossip models [7]. Computation proceeds in rounds and a random scheduler chooses a single pair of nodes to communicate in each round. The choice is independent across rounds, but may be nonuniform across node pairs.

▶ **Definition 1** (Asynchronous communication model)**.** Let V be a set of nodes with $|V| = n$. Computation proceeds in rounds, with every node $v \in V$ having some state $s(v, t)$ in round t.

Recall that $\mathcal{P}$ denotes all unordered pairs of nodes in V. Let $w : \mathcal{P} \to \mathbb{R}^+$ be a nonnegative weight function. In each round, a random scheduler chooses exactly one $(u, v) \in \mathcal{P}$ with probability $w(u, v) / \left[\sum_{(i,j) \in \mathcal{P}} w(i, j)\right]$ and u, v both update their states according to some common (possibly randomized) transition function σ. Specifically, they set $s(v, t+1) = \sigma(s(v, t), s(u, t))$ and $s(u, t+1) = \sigma(s(u, t), s(v, t))$.

Note that in our analysis we often identify the weight function w with a symmetric weight matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ where $\mathbf{W}_{u,u} = 0$ and $\mathbf{W}_{u,v} = \mathbf{W}_{v,u} = w(u, v) / \left[\sum_{(i,j) \in \mathcal{P}} w(i, j)\right]$. Let $\mathbf{D}$ be a diagonal matrix with $\mathbf{D}_{u,u} = \sum_{v \in V} \mathbf{W}_{u,v}$. $\mathbf{D}_{u,u}$ is the probability that node u

[1] We note that the analysis of Bechitti et al. seems likely to extend to our alternative communication model (Theorem 2), where the communication graph is weighted and regular

communicates in any given round. Since two nodes are chosen in each round, $\sum_u \mathbf{D}_{u,u} = 2$. We will refer to $\mathbf{D} + \mathbf{W}$ as the *communication matrix* of the communication model.

▶ Remark (Asynchronous algorithms). Since the transition function σ in Theorem 1 is universal, nodes can be seen as identical processes, with no knowledge of w or unique ids. We do assume that nodes can initiate and terminate a protocol synchronously. That is, nodes interact from round 0 up to some round T, after which they cease to interact, or begin a new protocol. This assumption is satisfied if each node has knowledge of the global round number but, in general, is much weaker. For example, in the asynchronous gossip model discussed below, it is sufficient for nodes to have access to a synchronized clock.

We use *algorithm* to refer to a sequence of transition functions, each corresponding to a subroutine run for specified number of rounds. Subroutines are run sequentially. The first has input nodes with identical starting states (as prescribed by Theorem 1) but later subroutines start once nodes have updated their states and thus have distinguished inputs.

▶ Remark (Simulation of existing models). The standard *population protocol model* [2] is recovered from Theorem 1 by setting $w(u, v) = 1$ for all (u, v) – i.e., pairs of nodes communicate uniformly at random. A similar model over a fixed communication graph $G = (E, V)$ is recovered by setting $w(u, v) = 1$ for all $(u, v) \in E$ and $w(u, v) = 0$ for $(u, v) \notin E$.

Theorem 1 also encompasses the *asynchronous gossip model* [7, 12], where each node holds an independent Poisson clock and contacts a random neighbor when the clock ticks. If we identify rounds with clock ticks, let λ_u be the rate of node u's clock, and let $p(u, v)$ be the probability that u contacts v when its clock ticks. Then the probability that nodes u and v interact in a given round is $\frac{1}{2}\left[\frac{\lambda_u}{\sum_{z\in V}\lambda_z} \cdot p(u, v) + \frac{\lambda_v}{\sum_{z\in V}\lambda_z} \cdot p(v, u)\right]$. With $w(u, v)$ set to this value, Theorem 1 corresponds exactly to the asynchronous gossip model.

2.3 Distributed community detection problem

This paper studies the very general problem of computing communication matrix eigenvectors with asynchronous protocols run by the nodes in V. One primary application of computing eigenvectors is to detect community structure in G. Below we formalize this application as the *distributed community detection problem* and introduce two specific cases of interest.

In the distributed community detection problem, the weight function w and corresponding weight matrix $\mathbf{W}$ of Theorem 1 are clustered: nodes in the same cluster are more likely to communicate than nodes in different clusters. The goal is for each node to independently identify what cluster it belongs to (up to a permutation of the cluster labels).

We consider two models of clustering. In the first (n, p, q)-*weighted communication model*, the weight function directly reflects the increased likelihood of intracluster communication. In the second, $G(n, p, q)$-*communication model*, weights are uniform on a graph sampled from the well-studied planted-partition or *stochastic block model* [15]. For simplicity, we focus on the setting in which there are two equal sized clusters, but believe that our techniques can be extended to handle a larger number of clusters, potentially with unbalanced sizes.

▶ **Definition 2** ((n, p, q)-weighted communication model)**.** An asynchronous model (Theorem 1), where node set V is partitioned into disjoint sets V_1, V_2 with $|V_1| = |V_2| = n/2$. For values $q < p$, $w(u, v) = p$ if $u, v \in V_i$ for some i and $w(u, v) = q$ if $u \in V_i$ and $v \in V_j$ for $i \neq j$.

▶ **Definition 3** ($G(n, p, q)$-communication model)**.** An asynchronous model (Theorem 1), where node set V is partitioned into disjoint sets V_1, V_2 with $|V_1| = |V_2| = n/2$. The weight matrix $\mathbf{W}$ is a normalized adjacency matrix of a random graph $G(V, E)$ generated as follows:

for each pair of nodes $u, v \in V$, add edge (u, v) to edge set E with probability p if u and v are in the same partition V_i and probability $q < p$ if u and v are in different partitions.

Analysis of community detection in the (n, p, q)-weighted communication model is more elegant, and will form the basis of our analysis for the $G(n, p, q)$-communication model, which more closely matches models considered in prior work on in both distributed and centralized settings. Formally, we define the distributed community detection problem as follows:

▶ **Definition 4** (Distributed community detection problem)**.** An algorithm executing in the communication models of Theorem 2 and Theorem 3 solves community detection in T rounds if for every $t \geq T$, all nodes in V_1 hold some integer state $s_1 \in \{-1, 1\}$, while all nodes in V_2 hold state $s_2 = -s_1$. An algorithm solves the community detection problem in L *local rounds* if every node's state remains fixed after L local interactions with other nodes.

3 Asynchronous Oja's algorithm

Our main contribution is a distributed algorithm for computing eigenvectors of the communication matrix $\mathbf{D} + \mathbf{W}$. These eigenvectors can be used to solve the distributed community detection problem or in other applications. Our main algorithm is a distributed, asynchronous adaptation of Oja's classic iterative eigenvector algorithm [27], described below:

Algorithm 1 Oja's method (centralized)

Input: $\mathbf{x}_0, ..., \mathbf{x}_{T-1} \in \mathbb{R}^n$ drawn i.i.d. from some distribution $\mathcal{D}$ such that for some constant C, $\mathbb{P}_{\mathbf{x}\sim\mathcal{D}}[\|\mathbf{x}\|_2^2 \leq C] = 1$ and $\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[\mathbf{x}\mathbf{x}^T] = \mathbf{M}$. Rank parameter k and step size η.
Output: Orthonormal $\tilde{\mathbf{V}} \in \mathbb{R}^{n\times k}$ whose columns approximate $\mathbf{M}$'s k top eigenvectors.

1: Choose $\mathbf{Q}_0$ with entries drawn i.i.d. from the standard normal distribution $\mathcal{N}(0, 1)$.
2: **for** $t = 0, ..., T-1$ **do**
3: $\quad \mathbf{Q}_{t+1} := (\mathbf{I} + \eta \mathbf{x}_t \mathbf{x}_t^T)\mathbf{Q}_t$.
4: **end for**
5: **return** $\tilde{\mathbf{V}}_T := \text{orth}(\mathbf{Q}_T)$. ▷ Orthonormalizes the columns of $\mathbf{Q}_T$.

3.1 Approximation bounds for Oja's method

A number of recent papers have provided strong convergence bounds for the *centralized* version of Oja's method [1, 16]. We will rely on the following theorem, which we prove in full verison using a straightforward application of the arguments in [1].

▶ **Theorem 5.** *Let* $\mathbf{M} \in \mathbb{R}^{n\times n}$ *be a PSD matrix with* $\frac{\sum_{i=1}^{k}\lambda_i(\mathbf{M})}{C} \leq \Lambda$ *and* $\frac{\lambda_k(\mathbf{M})-\lambda_{k+1}(\mathbf{M})}{C} \geq \text{gap}$ *for some values* Λ, gap*. For any* $\epsilon, \delta \in (0, 1)$*, let* $\xi = \frac{n}{\delta\epsilon\cdot\text{gap}}$*,* $\eta = \frac{c_1\epsilon^2\cdot\text{gap}\cdot\delta^2}{C\Lambda k\log^3\xi}$ *for some sufficiently small constant* c_1*, and* $T = \frac{c_2\cdot(\log\xi+1/\epsilon)}{C\cdot\text{gap}\cdot\eta}$ *for sufficiently large* c_2*. Then with probability* $\geq 1-\delta$*, Algorithm 1 run with step size* η *returns* $\tilde{\mathbf{V}}_T$ *satisfying,*

$$\|\mathbf{Z}^T\tilde{\mathbf{V}}_T\|_F^2 \leq \epsilon.$$

where $\mathbf{Z}$ *is an orthonormal basis for the bottom* $n-k$ *eigenvectors of* $\mathbf{M}$*.*

If $\tilde{\mathbf{V}}_T$ exactly spanned $\mathbf{M}$'s top k eigenvectors, $\|\mathbf{Z}^T\tilde{\mathbf{V}}_T\|_F^2$ would equal 0. To obtain an approximation of ϵ, the number of iterations required by Oja's method naturally depends inversely on ϵ, the failure probability δ, and the gap between eigenvalues $\lambda_k(\mathbf{M})$ and $\lambda_{k+1}(\mathbf{M})$.

3.2 Distributed Oja's method via random edge sampling

Oja's method can be implemented in the asynchronous communication model (Theorem 1) to compute top eigenvectors of the communication matrix $\mathbf{D}+\mathbf{W}$, defined in subsection 2.2.

For any pair of nodes (u,v), let $\mathbf{e}_{u,v}=\mathbf{e}_u+\mathbf{e}_v$ be the vector with all zero entries except 1's in its u^{th} and v^{th} positions. Given weight function w and associated matrix $\mathbf{W}$, let $\mathcal{D}_{\mathbf{W}}$ be the distribution in which each $\mathbf{e}_{u,v}$ is selected with probability $\mathbf{W}_{u,v}$. That is, the same distribution by which edges are selected to be active by the scheduler in Theorem 1. Noting that $\mathbf{e}_{u,v}\mathbf{e}_{u,v}^T$ is all zero except at its (u,u), (v,v), (u,v), and (v,u) entries, we can see that

$$\mathop{\mathbb{E}}_{\mathbf{e}_{u,v}\sim\mathcal{D}_{\mathbf{W}}}\left[\mathbf{e}_{u,v}\mathbf{e}_{u,v}^T\right]=\sum_{(u,v)\in\mathcal{P}}\mathbf{W}_{u,v}\cdot\mathbf{e}_{u,v}\mathbf{e}_{u,v}^T=\mathbf{D}+\mathbf{W}, \tag{1}$$

where $\mathcal{P}$ denotes the set of unordered node pairs (u,v) with $u\neq v$. So if we run Oja's algorithm with $\mathbf{e}_{u,v}$ sampled according to $\mathcal{D}_{\mathbf{W}}$, we will obtain an approximation to the top eigenvectors of $\mathbf{D}+\mathbf{W}$. Note that this matrix is PSD, by the fact that each $\mathbf{e}_{u,v}\mathbf{e}_{u,v}^T$ is PSD.

Furthermore, the algorithm can be implemented in our communication model as an *extremely simple averaging protocol.* Each iteration of Algorithm 1 requires computing $\mathbf{Q}_{t+1}=(\mathbf{I}+\eta\mathbf{x}_t\mathbf{x}_t^T)\mathbf{Q}_t$. If $\mathbf{x}_t=\mathbf{e}_{u,v}$ for $\mathbf{e}_{u,v}\sim\mathcal{D}_{\mathbf{W}}$, we can see that computing $\mathbf{Q}_{t+1}$ just requires updating the u^{th} and v^{th} rows of $\mathbf{Q}_t$. Thus, if the n rows of $\mathbf{Q}_t$ are distributed across n nodes, this update can be done locally by nodes u and v when they are chosen to interact by the randomized scheduler. Specifically, letting $[q_u^{(1)},...,q_u^{(k)}]$ be the u^{th} row of $\mathbf{Q}_t$, stored as the state at node u, applying $(\mathbf{I}+\eta\mathbf{e}_{u,v}\mathbf{e}_{u,v}^T)$ just requires setting for all $i\in[k]$:

$$q_u^{(i)}:=(1+\eta)q_u^{(i)}+\eta q_v^{(i)}. \tag{2}$$

Node v makes a symmetric update, and all other entries of $\mathbf{Q}_t$ remain fixed.

We give the pseudocode for this protocol in Algorithm 2. Along with the main iteration based on the simple update in (2), the nodes need to implement Step 5 of Algorithm 1, where $\mathbf{Q}_T$ is orthogonalized. This can be done with a gossip-based protocol, which we abstract as the routine `AsynchOrth`. We give an implementation of `AsynchOrth` in subsection 3.3.

▶ Remark (Choice of communication matrix). While, as we will show, the eigenvectors of $\mathbf{D}+\mathbf{W}$ are naturally useful in our applications to community detection, the above techniques easily extend to computing eigenvectors of other matrices. For example, if we set $\mathbf{e}_{u,v}=\mathbf{e}_u-\mathbf{e}_v$, $\mathbb{E}_{\mathbf{e}_{u,v}\sim\mathcal{D}_{\mathbf{W}}}[\mathbf{e}_{u,v}\mathbf{e}_{u,v}^T]=\mathbf{D}-\mathbf{W}=\mathbf{L}$, a scaled Laplacian of the communication graph.

Algorithm 2 Asynchronous Oja's (`AsynchOja`(T,T',η))

Input: Time bounds T,T', step size η.
Initialization: $\forall u$, chose $[q_u^{(1)},...,q_u^{(k)}]$ independently from standard Gaussian $\mathcal{N}(0,1)$.

1: **if** $t<T$ **then**
2: (u,v) is chosen by the randomized scheduler.
3: For all $i\in[k]$, $q_u^{(i)}:=(1+\eta)q_u^{(i)}+\eta q_v^{(i)}$. ▷ Computes of $(\mathbf{I}+\eta\mathbf{e}_{u,v}\mathbf{e}_{u,v}^T)\mathbf{Q}_t$.
4: **else**
5: $[\hat{v}_u^{(1)},...,\hat{v}_u^{(k)}]=$ `AsynchOrth`$([q_u^{(1)},...,q_u^{(k)}],T')$. ▷ Implements of $\tilde{\mathbf{V}}_T=\text{orth}(\mathbf{Q}_T)$.
6: **end if**

Note that in the pseudocode above, when nodes u,v interact in the asynchronous model, they only need to share their respective values of $q_u^{(i)}$ and $q_v^{(i)}$ for $i\in[k]$.

Up to the orthogonalization step, we see that Algorithm 2 *exactly simulates* Algorithm 1 on input $\mathbf{M}=\mathbf{D}+\mathbf{W}$. Thus, assuming that `AsynchOrth`$([q_u^{(1)},...,q_u^{(k)}])$ exactly computes

$\tilde{\mathbf{V}}_T = \text{orth}(\mathbf{Q}_T)$ as in Step 5 of Algorithm 1, the error bound of Theorem 5 applies directly. Specifically, if we let the local states, $[q_1^{(1)}, \ldots, q_n^{(1)}], \ldots, [q_1^{(k)}, \ldots, q_n^{(k)}]$ correspond to the k length-n vectors in $\tilde{\mathbf{V}}_T$, Theorem 5 shows that $\|\mathbf{Z}^T\tilde{\mathbf{V}}_T\|_F^2 \leq \epsilon$. In subsection 3.3 we show that this bound still holds when `AsynchOrth` computes an approximate orthogonalization.

3.3 Distributed orthogonalization and eigenvector guarantees

In fact, a specific orthogonalization strategy yields a stronger bound, which is desirable in many applications, including community detection: Algorithm 2 can actually well approximate *each* of $\mathbf{D} + \mathbf{W}$'s top k eigenvectors, instead of just the subspace they span.

Specifically, let $\tilde{\mathbf{v}}_i$ denote the i^{th} column of $\tilde{\mathbf{V}}_T$ and $\mathbf{v}_i$ denote the i^{th} eigenvector of $\mathbf{D} + \mathbf{W}$. We want $(\tilde{\mathbf{v}}_i^T\mathbf{v}_i)^2 \geq 1 - \epsilon$ for all i. Such a guarantee requires sufficiently large gaps between the top k eigenvalues, so that their corresponding eigenvectors are identifiable. If these gaps exist, the guarantee can by using the following orthogonalization procedure:

Algorithm 3 Orthogonalization via Cholesky Factorization (centralized)

Input: $\mathbf{Q} \in \mathbb{R}^{n\times k}$ with full column rank. **Output**: Orthonormal span for $\mathbf{Q}$, $\tilde{\mathbf{V}} \in \mathbb{R}^{n\times k}$.

1: $\mathbf{L} := \text{chol}(\mathbf{Q}^T\mathbf{Q})$ ▷ Cholesky decomp. returns lower triangular $\mathbf{L}$ with $\mathbf{L}\mathbf{L}^T = \mathbf{Q}^T\mathbf{Q}$.
2: **return** $\tilde{\mathbf{V}} := \mathbf{Q}(\mathbf{L}^T)^{-1}$ ▷ Orthonormalize $\mathbf{Q}_T$'s columns using the Cholesky factor.

▶ Remark. Algorithm 3 requires an input that is *full-rank*, which always includes $\mathbf{Q}_T$ in Algorithms 1 and 2: $\mathbf{Q}_0$'s entries are random Gaussians so it is full-rank with probability 1 and each $(\mathbf{I} + \eta\mathbf{x}_t^T\mathbf{x}_t)$ is full-rank since $\eta < \|\mathbf{x}_t\|$. Thus, $\mathbf{Q}_T = \prod_{t=0}^{T-1}(\mathbf{I} + \eta\mathbf{x}_t^T\mathbf{x}_t)\mathbf{Q}_0$ is too.

Ultimately, our `AsynchOrth` is an asynchronous distributed implementation of Algorithm 3. We first prove an eigenvector approximation bound under the assumption that this implementation is exact and then adapt that result to account for the fact that `AsynchOrth` only outputs an approximate solution.

Pseudocode for `AsynchOrth` is included below. Each node first computes a (scaled) approximation to every entry of $\mathbf{Q}^T\mathbf{Q}$ using a simple averaging technique. Nodes then locally compute $\mathbf{L} = \text{chol}\left(\mathbf{Q}^T\mathbf{Q}\right)$ and the u^{th} row of $\tilde{\mathbf{V}}_T = \mathbf{Q}(\mathbf{L}^T)^{-1}$. In the full version of this paper [20] we argue that, due to numerical stability of Cholesky decomposition, each node's output is close to the u^{th} row of an exactly computed $\tilde{\mathbf{V}}_T$, despite the error in constructing $\mathbf{Q}^T\mathbf{Q}$.

Algorithm 4 Asynchronous Cholesky Orthogonalization (`AsynchOrth`(T))

Input: Time bound T.
Initialization: Each node holds $[q_u^{(1)}, ..., q_u^{(k)}]$. For all $i, j \in [k]$, let $r_u^{(i,j)} := q_u^{(i)} \cdot q_u^{(j)}$.

1: **if** $t < T$ **then**
2: (u, v) is chosen by the randomized scheduler.
3: for all $i, j \in [k]$, $r_u^{(i,j)} := \frac{r_u^{(i,j)} + r_v^{(i,j)}}{2}$. ▷ Estimation of $\frac{1}{n}\mathbf{q}_i^T\mathbf{q}_j$ via averaging.
4: **else**
5: Form $\mathbf{R}_u \in \mathbb{R}^{k\times k}$ with $(\mathbf{R}_u)_{i,j} = (\mathbf{R}_u)_{j,i} := n \cdot r_u^{(i,j)}$. ▷ Approximation of $\mathbf{Q}^T\mathbf{Q}$.
6: $\mathbf{L}_u := \text{chol}(\mathbf{R}_u)$.
7: $[\hat{v}_u^{(1)}, ..., \hat{v}_u^{(k)}] := [q_u^{(1)}, ..., q_u^{(k)}] \cdot (\mathbf{L}_u^T)^{-1}$. ▷ Approximation of u^{th} row of $\mathbf{Q}(\mathbf{L}_u^T)^{-1}$.
8: **end if**

Ultimately in [20] we prove the following result when Algorithm 4 is used to implement `AsynchOrth` as a subroutine for Algorithm 2, `AsynchOja`(T, T', η):

▶ **Theorem 6** (Asynchronous eigenvector approximation). *Let $\mathbf{v}_1, ..., \mathbf{v}_k$ be the top k eigenvectors of the communication matrix $\mathbf{D}+\mathbf{W}$ in an asynchronous communication model, and let $\Lambda, \overline{\text{gap}}, \gamma_{mix}$ be bounds satisfying: $\Lambda \geq \sum_{j=1}^{k} \lambda_j(\mathbf{D}+\mathbf{W})$, $\overline{\text{gap}} \leq \min_{j\in[k]}[\lambda_j(\mathbf{D}+\mathbf{W}) - \lambda_{j+1}(\mathbf{D}+\mathbf{W})]$, and $\gamma_{mix} \leq \min\left[\frac{1}{n}, \log\left(\lambda_2^{-1}(\mathbf{I} - \frac{1}{2}\mathbf{D} + \frac{1}{2}\mathbf{W})\right)\right]$.*

For any $\epsilon, \delta \in (0,1)$, let $\xi = \frac{n}{\delta\epsilon\cdot\overline{\text{gap}}}$. Let $\eta = \frac{c_1\epsilon^2\cdot\overline{\text{gap}}\cdot\delta^2}{\Lambda k^3 \log^3 \xi}$ for sufficiently small c_1, and $T = \frac{c_2\cdot(\log\xi + 1/\epsilon)}{\overline{\text{gap}}\cdot\eta}$, $T' = \frac{c_3(\log\xi+1/\epsilon)\cdot\lambda_1(\mathbf{D}+\mathbf{W})}{\overline{\text{gap}}\cdot\gamma_{mix}}$ for sufficiently large c_2, c_3. For all $u \in [n], i \in [k]$, let $\hat{v}_u^{(j)}$ be the local state computed by Algorithm 2. If $\hat{\mathbf{V}} \in \mathbb{R}^{n\times k}$ is given by $(\hat{\mathbf{V}})_{u,j} = \hat{v}_u^{(j)}$ and $\hat{\mathbf{v}}_i$ is the i^{th} column of $\hat{\mathbf{V}}$, then with probability $\geq 1 - \delta - e^{-\Theta(n)}$, for all $i \in [k]$:

$$\left|\hat{\mathbf{v}}_i^T \mathbf{v}_i\right| \geq 1 - \epsilon \quad \text{and} \quad \|\hat{\mathbf{v}}_i\|_2 \leq 1 + \epsilon.$$

4 Distributed community detection

From the results of section 3, we obtain a simple population protocol for distributed community detection that works for many clustered communication models, including the (n,p,q)-weighted communication and $G(n,p,q)$-communication models of Definitions 2 and 3.

In particular, we show that if each node $u \in V$ can locally compute the u^{th} entry of an approximation $\hat{\mathbf{v}}_2$ to the second eigenvector of the communication matrix $\mathbf{D}+\mathbf{W}$, then it can solve the community detection problem locally: u just sets its state to the sign of this entry.

Algorithm 5 Asynchronous Community Detection (`AsynchCD`(T, T', η))

Input: Time bounds T, T', step size η.

1: Run `AsynchOja`(T, T', η) (Algorithm 2) with $k = 2$.
2: Set $\hat{\chi}_u := \text{sign}(\hat{v}_u^{(2)})$.

Here $\hat{\chi}_u \in \{-1, 1\}$ is the final state of node u. We will claim that this state solves the community detection problem of Theorem 4. We use the notation $\hat{\chi}_u$ because we will use $\boldsymbol{\chi}$ to denote the true *cluster indicator vector* for communities V_1 and V_2 in a given communication model: $\boldsymbol{\chi}_u = 1$ for $u \in V_1$ and $\boldsymbol{\chi}_u = -1$ for $u \in V_2$.

In particular, we will show that if η is set so that `AsynchOja` outputs eigenvectors with accuracy ϵ, then a $1 - O(\epsilon)$ fraction of nodes will correctly identify their clusters. In section 5 we show how to implement a 'cleanup phase' where, starting with ϵ set to a small constant (e.g. $\epsilon = .1$), the nodes can converge to a state with *all* cluster labels correct with high probability.

4.1 Community detection in the (n,p,q)-weighted communication model

We start with an analysis for the (n,p,q)-weighted communication model. Recall that in this model the nodes are partitioned into two sets, V_1 and V_2, each with $n/2$ elements. Without loss of generality we can identify the nodes with integer labels such that $1, \ldots, n/2 \in V_1$ and $n/2+1, \ldots, n \in V_2$. We define the weighted cluster indicator matrix, $\mathbf{C}^{(p,q)} \in \mathbb{R}^{n\times n}$:

$$\mathbf{C}^{(p,q)} \stackrel{\text{def}}{=} \begin{bmatrix} p\cdot\mathbf{1}_{\frac{n}{2}\times\frac{n}{2}} & q\cdot\mathbf{1}_{\frac{n}{2}\times\frac{n}{2}} \\ q\cdot\mathbf{1}_{\frac{n}{2}\times\frac{n}{2}} & p\cdot\mathbf{1}_{\frac{n}{2}\times\frac{n}{2}} \end{bmatrix}. \tag{3}$$

p and q can be arbitrary, but we will always take $p > q > 0$. It is easy to check that $\mathbf{C}^{(p,q)}$ is a rank two matrix with eigendecomposition:

$$\mathbf{C}^{(p,q)} = \frac{n}{2}\begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2\end{bmatrix}\begin{bmatrix}p+q & 0\\ 0 & p-q\end{bmatrix}\begin{bmatrix}\mathbf{v}_1^T\\ \mathbf{v}_2^T\end{bmatrix} \quad \text{where} \quad \mathbf{v}_1 = \frac{\mathbf{1}_{n\times 1}}{\sqrt{n}},\ \mathbf{v}_2 = \frac{\boldsymbol{\chi}}{\sqrt{n}}. \tag{4}$$

So, if all nodes could compute their corresponding entry in the *second eigenvector* of $\mathbf{C}^{(p,q)}$, then by simply returning the sign of this entry, they would solve the distributed community detection problem (Theorem 4). If they compute this eigenvector approximately, then we can still show that a large fraction of them correctly solve community detection. Specifically:

▶ **Lemma 7.** *Let $\mathbf{v}_2$ be the second eigenvector of $\mathbf{C}^{(p,q)}$ for any $p > q > 0$. If $\tilde{\mathbf{v}}_2$ satisfies:*

$$\left|\tilde{\mathbf{v}}_2^T\mathbf{v}_2\right| \geq 1-\epsilon \quad \textit{and} \quad \|\tilde{\mathbf{v}}_2\|_2 \leq 1+\epsilon. \tag{5}$$

for $\epsilon \leq 1$, then $\operatorname{sign}(\tilde{\mathbf{v}}_2)$ gives a labeling such that, after ignoring at most $5\epsilon n$ nodes, all remaining nodes in V_1 have the same labeling, and all in V_2 have the opposite.

A proof can be found in [20]. With Theorem 7 in place, we can then apply Theorem 6 to prove the correctness of `AsynchCD` (Algorithm 5) for the (n,p,q)-weighted communication model

▶ **Theorem 8** (ϵ-approximate community detection: (n,p,q)-weighted communication model)**.** *Consider Algorithm 5 in the (n,p,q)-weighted communication model. Let $\rho = \min\left(\frac{q}{p+q}, \frac{p-q}{p+q}\right)$. For sufficiently small constant c_1 and sufficiently large c_2 and c_3, let*

$$\eta = \frac{c_1\epsilon^2\delta^2\rho}{\log^3\left(\frac{n}{\epsilon\delta\rho}\right)},\ T = \frac{c_2 n\left(\log^3\left(\frac{n}{\epsilon\delta\rho}\right) + \frac{\log\left(\frac{n}{\epsilon\delta\rho}\right)}{\epsilon}\right)}{\epsilon^2\delta^2\rho^2},\ T' = \frac{c_3 n\left(\log\left(\frac{n}{\epsilon\delta\rho}\right) + \frac{1}{\epsilon}\right)}{\rho^2}.$$

With probability $1-\delta$, after ignoring ϵn nodes, all remaining nodes in V_1 terminate in some state $s_1 \in \{-1,1\}$, and all nodes in V_2 terminate in state $s_2 = -s_1$. Suppressing polylogarithmic factors in the parameters, the total number of global rounds and local rounds required are: $T + T' = \tilde{O}\left(\frac{n}{\epsilon^3\delta^2\rho^2}\right)$ and $L = \tilde{O}\left(\frac{1}{\epsilon^3\delta^3\rho^2}\right)$.

Proof. In the (n,p,q)-weighted communication model the weight and degree matrices are:

$$\mathbf{W} = \frac{4}{n^2(p+q)-2np}\cdot(\mathbf{C}^{(p,q)} - p\cdot\mathbf{I}_{n\times n}) \quad \text{and} \quad \mathbf{D} = \frac{2}{n}\cdot\mathbf{I}_{n\times n}.$$

Thus, referring to the eigendecomposition of $\mathbf{C}^{(p,q)}$ shown in (4), the top eigenvector of $\mathbf{D}+\mathbf{W}$ is $\mathbf{v}_1 = \mathbf{1}_{n\times 1}/\sqrt{n}$ with corresponding eigenvalue: $\lambda_1 = \frac{4}{n^2(p+q)-2np}\cdot\left(\frac{n(p+q)}{2} - p\right) + \frac{2}{n} = \frac{4}{n}$. The second eigenvector is the scaled cluster indicator vector $\mathbf{v}_2 = \boldsymbol{\chi}/\sqrt{n}$ with eigenvalue

$$\lambda_2 = \frac{4}{n^2(p+q)-2np}\cdot\left(\frac{n(p-q)}{2} - p\right) + \frac{2}{n} = \frac{4}{n}\cdot\frac{p}{p + \frac{n}{n-2}\cdot q}.$$

Finally, for all remaining eigenvalues of $\mathbf{D}+\mathbf{W}$, $\{\lambda_3, ..., \lambda_n\}$, $\lambda_i = \frac{2}{n} - \frac{4p}{n^2(p+q)-2np}$. We can bound the eigenvalue gaps:

$$\lambda_1 - \lambda_2 \geq \frac{4}{n} - \frac{4}{n}\cdot\frac{p}{p+q} = \frac{4q}{n(p+q)} \qquad \lambda_2 - \lambda_3 = \frac{2(p-q)}{n(p+q)-2p} \geq \frac{2(p-q)}{n(p+q)}$$

Let $\rho = \min\left(\frac{q}{p+q}, \frac{p-q}{(p+q)}\right)$. We bound the mixing time of $\mathbf{W}+\mathbf{D}$ by noting that $\lambda_2(\mathbf{I} - 1/2\mathbf{D} + 1/2\mathbf{W}) \leq 1 - \frac{2q}{n(p+q)}$. Then using that $\log(1/x) \geq 1-x$ for all $x \in (0,1]$, $\log(\lambda_2^{-1}(\mathbf{I} - 1/2\mathbf{D} + 1/2\mathbf{W}) \geq \frac{2q}{n(p+q)} \geq \frac{2\rho}{n}$. We then apply Theorem 6 with $k=2$, $\Lambda = \frac{4}{n} + \frac{4}{n}\frac{p}{p+\frac{n}{n-2}q} \leq \frac{8}{n}$, $\overline{\text{gap}} = \frac{4}{n} \cdot \min\left(\frac{q}{p+q}, \frac{p-q}{2(p+q)}\right) \geq \frac{2\rho}{n}$, and $\gamma_{mix} = \frac{2\rho}{n}$. With these parameters we set, for sufficiently small c_1 and large c_2, c_3,

$$\eta = \frac{c_1\epsilon^2\delta^2 \cdot \rho}{\log^3\left(\frac{n}{\epsilon\delta\rho}\right)}, \quad T = \frac{c_2 \cdot n \cdot \left(\log^3\left(\frac{n}{\epsilon\delta\rho}\right) + \frac{\log\left(\frac{n}{\epsilon\delta\rho}\right)}{\epsilon}\right)}{\epsilon^2\delta^2\rho^2}, \quad T' = \frac{c_3 \cdot n \cdot \left(\log\left(\frac{n}{\epsilon\delta\rho}\right) + \frac{1}{\epsilon}\right)}{\rho^2}$$

where to bound T' we use that $\frac{\lambda_1(\mathbf{D}+\mathbf{W})}{\overline{\text{gap}}} \leq \frac{2}{\rho}$. Let $\hat{\mathbf{V}} \in \mathbb{R}^{n\times k}$ be given by $(\hat{\mathbf{V}})_{u,j} = \hat{v}_u^{(j)}$ where $\hat{v}_u^{(j)}$ are the states of $\texttt{AsynchOja}(T, T', \eta)$ and let $\hat{\mathbf{v}}_2$ be the second column of $\hat{\mathbf{V}}$. With these parameters, Theorem 6 gives with probability $\geq 1-\delta$ that $\left|\hat{\mathbf{v}}_2^T\mathbf{v}_2\right| \geq 1-\epsilon$ and $\|\hat{\mathbf{v}}_2\|_2 \leq 1+\epsilon$.

Applying Theorem 7 then gives the theorem if we adjust ϵ by a factor of $1/5$. Recall that the second eigenvector of $\mathbf{D}+\mathbf{W}$ is identical to that of $\mathbf{C}^{(p,q)}$. Additionally, in expectation, each node is involved in $L = \frac{2(T+T')}{n}$ interactions. This bound holds for all nodes within a factor 2 with probability $1-\delta$ by a Chernoff bound, since $L = \Omega(\log(n/\delta))$. We can union bound over our two failure probabilities and adjust δ by $1/2$ to obtain overall failure probability $\leq \delta$. ◀

4.2 Community Detection in the $G(n,p,q)$-communication model

In the $G(n,p,q)$-communication model, nodes communicate using a random graph which is equal to the communication graph in the (n,p,q)-weighted communication model *in expectation*. Using an approach similar to [30], which is a simplifies the perturbation method used in [21], we can prove that in the $G(n,p,q)$-communication model $\mathbf{W}$ is a small perturbation of $\mathbf{C}^{(p,q)}$ and so the second eigenvector of $\mathbf{D}+\mathbf{W}$ approximates that of $\mathbf{C}^{(p,q)}$ – i.e., the cluster indicator vector $\boldsymbol{\chi}$. We defer this analysis to the full version [20], stating the main result here:

▶ **Theorem 9** (ϵ-approximate community detection: $G(n,p,q)$-communication model)**.** *Consider Algorithm 5 in the $G(n,p,q)$-communication model. Let $\rho = \min\left(\frac{q}{p+q}, \frac{p-q}{p+q}\right)$. For sufficiently small constant c_1 and sufficiently large c_2 and c_3 let*

$$\eta = \frac{c_1\epsilon^2\delta^2\rho}{\log^3\left(\frac{n}{\epsilon\delta\rho}\right)}, \quad T = \frac{c_2 n \left(\log^3\left(\frac{n}{\epsilon\delta\rho}\right) + \frac{\log\left(\frac{n}{\epsilon\delta\rho}\right)}{\epsilon}\right)}{\epsilon^2\delta^2\rho^2}, \quad T' = \frac{c_3 n \left(\log\left(\frac{n}{\epsilon\delta\rho}\right) + \frac{1}{\epsilon}\right)}{\rho^2}.$$

If $\frac{\min[q,p-q]}{\sqrt{p+q}} \geq \frac{c_4\sqrt{\log(n/\delta)}}{\epsilon\sqrt{n}}$ for large enough constant c_4, then, with probability $1-\delta$, after ignoring ϵn nodes, all remaining nodes in V_1 terminate in some state $s_1 \in \{-1,1\}$, and all nodes in V_2 terminate in state $s_2 = -s_1$. Supressing polylogarithmic factors, the total number of global rounds and local rounds required are: $T+T' = \tilde{O}\left(\frac{n}{\epsilon^3\delta^2\rho^2}\right)$ and $L = \tilde{O}\left(\frac{1}{\epsilon^3\delta^3\rho^2}\right)$.

If for example, $p, q = \Theta(1)$ and thus the $G(n,p,q)$ graph is dense, we can recover the communities with probability $1-\delta$ up to $O(1)$ error as long as $q \leq p - c\sqrt{\log(n/\delta)/n}$ for sufficiently large constant c. Alternatively, if $p, q = \Theta\left(\log(n/\delta)/n\right)$, so the $G(n,p,q)$ graph is sparse, we require $q \leq cp$ for sufficiently small c.

5 Cleanup Phase

After we apply Theorem 9 (respectively, Theorem 8) an ϵ-fraction of nodes are incorrectly clustered. The goal of this section is to provide a simple algorithm that improves this clustering so that *all nodes* are labeled correctly after a small number of rounds.

For the (n, p, q)-weighted communication model, doing so is straightforward. After running Algorithm 2 and selecting a label, each time a node communicates in the future it records the chosen label of the node it communicates with. Ultimately, it changes its label to the majority of labels encountered. If ϵ is small enough so $p(1-\epsilon) > q + \epsilon p$, this majority tends towards the node's correct label. The number of required rounds for the majority to be correct, with good probability for all nodes, is a simple a function of p, q, and ϵ.

The $G(n, p, q)$-communication model is more difficult. Theorem 9 does not guarantee how incorrectly labeled nodes are distributed: it is possible that a majority of a node's neighbors fall into the set of ϵn "bad nodes". In that case, even after infinitely many rounds of communication, the majority label encountered will not tend towards the node's correct identity.

As a remedy, we introduce a phased algorithm (Algorithm 6) where each node updates its label to the majority of labels seen during a phase. We show that in each phase the fraction of incorrectly labeled nodes decreases by a constant factor. Our analysis establishes a graph theoretic bound on the external edge density of most subsets of nodes. Specifically, for all subsets S below a certain size, we show that, with high probability, there are at most $|S|/3$ nodes which have enough connections to S so that if an adversary gave all nodes in S incorrect labels, it could cause these nodes to have an incorrect majority label. This bound guarantees that at most $|S|/3$ bad labels 'propagate' to the next phase of the algorithm.

Algorithm 6 Cleanup phase (pseudocode for node u)
Input: Number of phases k and number of rounds per phase r.
Output: Label $\hat{\chi}_u \in \{-1, 1\}$

```
for Phase 1 to k do
    for Round i = 1 to r do
        S_i := χ̂_v, where χ̂_v denotes the i^th sample of node u.
    end for
    χ̂_u := 1 if Σ_i^r S_i ≥ 0, χ̂_u := −1 otherwise.
end for
```

▶ **Theorem 10.** *Consider the (n, p, q)-weighted communication model. Assume that a fraction of at most $\epsilon \leq 1/64$ of the nodes are incorrectly clustered after Algorithm 2. As long as $p' = (1-\epsilon)p$ and $q' = q + \epsilon p$ satisfy $p' > q'$, Algorithm 6 ensures that all nodes are correctly labeled with high probability after $O(\frac{p \ln n}{(\sqrt{p'} - \sqrt{q'})^2})$ local rounds. In particular, for $q \leq p/2$ and $\epsilon < 1/8$, the number of local rounds required is $O(\log n)$.*

▶ **Theorem 11.** *Consider the $G(n, p, q)$-communication model. Let $\Delta = \frac{p}{2} - \frac{q}{2} - \sqrt{12p \ln n/n} - \sqrt{12q \ln n/n}$. Assume that $\Delta = \Omega(\ln n/n)$ and at most $\epsilon \leq \Delta/24p$ nodes are incorrectly clustered after Algorithm 2. As long as $p'' = \frac{p}{2} - \sqrt{\frac{6p \ln n}{n}} - \frac{\Delta}{12}$ and $q'' = \frac{q}{2} + \sqrt{\frac{6q \ln n}{n}} + \frac{\Delta}{12}$ satisfy $p'' > q''$, Algorithm 6 ensures that all nodes are correctly labeled with high probability after $O(\frac{p \ln^2 n}{(\sqrt{p''} - \sqrt{q''})^2})$ local rounds. In particular, for $q \leq p/2$ the number of local rounds required is $O(\log^2 n)$.*

Note that if $p - q = \Omega(\sqrt{\log n/n})$, then Δ simplifies to $\Delta = \Theta(p - q)$. Incidentally, $p - q = \Omega(\sqrt{\log n/n})$ is sometimes tight because, in this regime, clustering correctly can be infeasible: some nodes will simply have more neighbors in the opposite cluster. Consider for example when $p = 1/2 + \sqrt{\ln n/(10n)}$ and $q = 1/2$.

References

1. Zeyuan Allen-Zhu and Yuanzhi Li. First efficient convergence for streaming k-pca: a global, gap-free, and near-optimal rate. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 487–492, 2017.
2. James Aspnes and Eric Ruppert. An introduction to population protocols. In *Middleware for Network Eccentric and Mobile Applications*, pages 97–120. Springer, 2009.
3. Luca Becchetti, Andrea Clementi, , Pasin Manurangsi, Emanuele Natale, Francesco Pasquale, Prasad Raghavendra, and Luca Trevisan. Average whenever you meet: Opportunistic protocols for community detection. *arXiv:1703.05045*, 2017.
4. Luca Becchetti, Andrea Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Find your place: Simple distributed algorithms for community detection. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 940–959, 2017.
5. Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002.
6. Phillip Bonacich. Some unique properties of eigenvector centrality. *Social networks*, 29(4):555–564, 2007.
7. Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE/ACM Transactions on Networking (TON)*, 14(SI):2508–2530, 2006.
8. Shakila Bu-Pasha. Cross-border issues under eu data protection law with regards to personal data protection. *Information & Communications Technology Law*, 26(3):1–16, 05 2017. doi:10.1080/13600834.2017.1330740.
9. Sam Cole, Shmuel Friedland, and Lev Reyzin. A simple spectral algorithm for recovering planted partitions. *Special Matrices*, 5:139–157, 2017.
10. Christopher M De Sa, Ce Zhang, Kunle Olukotun, and Christopher Ré. Taming the wild: A unified analysis of hogwild-style algorithms. In *Advances in neural information processing systems*, pages 2674–2682, 2015.
11. Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Physical Review E*, 84(6):066106, 2011.
12. Alexandros G Dimakis, Soummya Kar, José MF Moura, Michael G Rabbat, and Anna Scaglione. Gossip algorithms for distributed signal processing. *Proceedings of the IEEE*, 98(11):1847–1864, 2010.
13. Chris HQ Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst D Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 107–114. IEEE, 2001.
14. Nisrine Ghadban, Paul Honeine, Farah Mourad-Chehade, Joumana Farah, and Clovis Francis. Gossip algorithms for principal component analysis in networks. In *Signal Processing Conference (EUSIPCO), 2015 23rd European*, pages 2366–2370. IEEE, 2015.
15. Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.
16. Prateek Jain, Chi Jin, Sham M. Kakade, Praneeth Netrapalli, and Aaron Sidford. Streaming pca: Matching matrix bernstein and near-optimal finite sample guarantees for oja's

algorithm. In *Proceedings of the 29th Annual Conference on Computational Learning Theory (COLT)*, 2016.

17 David Kempe and Frank McSherry. A decentralized algorithm for spectral analysis. *Journal of Computer and System Sciences*, 74(1):70–83, 2008.

18 Satish Babu Korada, Andrea Montanari, and Sewoong Oh. Gossip pca. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 209–220. ACM, 2011.

19 Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

20 Frederik Mallmann-Trenn, Cameron Musco, and Christopher Musco. Eigenvector computation and community detection in asynchronous gossip models. *arXiv:1804.08548*, 2018.

21 F. McSherry. Spectral partitioning of random graphs. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 529–537, 2001.

22 Frank McSherry. Spectral partitioning of random graphs. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 529–537, 2001.

23 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

24 Gemma Morral, Pascal Bianchi, and Jérémie Jakubowicz. Asynchronous distributed principal component analysis using stochastic approximation. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 1398–1403. IEEE, 2012.

25 Elchanan Mossel, Joe Neeman, and Allan Sly. Belief propagation, robust reconstruction and optimal recovery of block models. In *Conference on Learning Theory*, pages 356–370, 2014.

26 Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

27 Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273, 1982.

28 Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

29 Semih Salihoglu, Jaeho Shin, Vikesh Khanna, Ba Quan Truong, and Jennifer Widom. Graft: A debugging tool for apache giraph. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1403–1408. ACM, 2015.

30 Daniel Spielman. Spectral graph theory: Spectral partitiong in a stochastic block model. University Lecture, 2015.

31 John Tsitsiklis, Dimitri Bertsekas, and Michael Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986.

32 Van Vu. A simple SVD algorithm for finding hidden partitions. *Combinatorics, Probability and Computing*, 27(1):124–140, 2018.

$(\Delta + 1)$ Coloring in the Congested Clique Model

Merav Parter
Weizmann IS, Rehovot, Israel
merav.parter@weizmann.ac.il

Abstract

In this paper, we present improved algorithms for the $(\Delta + 1)$ (vertex) coloring problem in the *Congested Clique* model of distributed computing. In this model, the input is a graph on n nodes, initially each node knows only its incident edges, and per round each two nodes can exchange $O(\log n)$ bits of information.

Our key result is a randomized $(\Delta + 1)$ vertex coloring algorithm that works in $O(\log\log\Delta \cdot \log^*\Delta)$-rounds. This is achieved by combining the recent breakthrough result of [Chang-Li-Pettie, STOC'18] in the LOCAL model and a degree reduction technique. We also get the following results with high probability: (1) $(\Delta + 1)$-coloring for $\Delta = O((n/\log n)^{1-\epsilon})$ for any $\epsilon \in (0, 1)$, within $O(\log(1/\epsilon)\log^*\Delta)$ rounds, and (2) $(\Delta + \Delta^{1/2+o(1)})$-coloring within $O(\log^*\Delta)$ rounds.
Turning to *deterministic* algorithms, we show a $(\Delta+1)$-coloring algorithm that works in $O(\log\Delta)$ rounds. Our new bounds provide *exponential* improvements over the state of the art.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed Graph Algorithms, Coloring, Congested Clique

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.160

Related Version A full version of the paper is available at [22], https://arxiv.org/abs/1805.02457.

Acknowledgements I am very grateful to Hsin-Hao Su, Eylon Yogev, Seth Pettie, Yi-Jun Chang and the anonymous reviewers for helpful comments.

1 Introduction & Related Work

Graph coloring is one of the most central symmetry breaking problems, with a wide range of applications to distributed systems and wireless networks. The most studied coloring problem is the $(\Delta + 1)$ *vertex* coloring in which all nodes are given the *same* palette of $\Delta + 1$ colors, where Δ is the maximum degree in the graph. Vertex coloring among other LCL problems[1] (e.g., MIS, matching) are traditionally studied in the LOCAL model in which any two neighboring vertices in the input graph can exchange arbitrarily long messages.

In recent years there has been a tremendous progress in the understanding of the randomized and the deterministic complexities of many LCL problems in the LOCAL model [6, 20, 8, 9, 1]. Putting our focus on the $(\Delta + 1)$ coloring problem, in a seminal work, Schneider and Wattenhofer [23] showed that increasing the number of colors from $\Delta + 1$ to $(1 + \epsilon)\Delta$ has a dramatic effect on the round complexity and coloring can be computed in just $O(\log^* n)$ rounds when $\epsilon = \Omega(1)$ and $\Delta > \mathsf{poly}\log n$. This has led to two recent breakthroughs. Harris, Schneider and Su [13] showed an $O(\sqrt{\log\Delta})$-round algorithm for $(\Delta + 1)$ coloring, providing a separation for the first time between MIS and coloring (due to the MIS lower bound of [17]). In a recent follow-up breakthrough, Chang, Li and Pettie [7] extended the

[1] LCL stands for Locally Checkable Labelling problems, see [21].

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 160; pp. 160:1–160:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

technique of [13] to obtain the remarkable and quite extraordinary round complexity of $O(\log^* n + \mathsf{Det}_{\deg}(\mathsf{polylog} n))$ for the $(\Delta+1)$-list coloring problem where $\mathsf{Det}_{\deg}(n')$ is the deterministic round complexity of $(\deg+1)$ list coloring algorithm[2] in n'-vertex graph. Both of these recent breakthroughs use messages of large size, potentially of $\Omega(n)$ bits.

In view of these recent advances, the understanding of LCL problems in bandwidth-restricted models is much more lacking. Among these models, the congested clique model [19], which allows all-to-all communication has attracted a lot of attention in the last decade and more recently, in the context of LCL problems [4, 15, 14, 5, 12, 24]. In the congested clique model, each node can send $O(\log n)$ bits of information to any node in the network (i.e., even if they are not connected in the input graph). The ubiquitous of overlay networks and large scale distributed networks make the congested clique model far more relevant (compared to the LOCAL and the CONGEST models) in certain settings.

Randomized LCL in the Congested Clique Model. Starting with Barenboim et al. [2], currently, all efficient randomized algorithms for classical LCL problems have the following structure: an initial randomized phase and a *post-shattering* deterministic phase. The shattering effect of the randomized phase which dates back to Beck [3], breaks the graph into subproblems of $\mathsf{poly}\log n$ size to be solved deterministically. In the congested-clique model, the shattering effect has an even more dramatic effect. Usually, a node survives (i.e., remained undecided) the randomized phase with probability of $1/\mathsf{poly}(\Delta)$. Hence, in expectation the size of the remaining unsolved graph is[3] $O(n)$. At that point, the entire unsolved subgraph can be solved in $O(1)$ rounds, using standard congested clique tools (e.g., the routing algorithm by Lenzen [18]). Thus, as long as the main randomized part uses short messages, the congested clique model "immediately" enjoys an improved round complexity compared to that of the LOCAL model.

In a recent work [12], Ghaffari took it few steps farther and showed an $\widetilde{O}(\sqrt{\log\Delta})$-round randomized algorithm for MIS in the congested clique model, improving upon the state-of-the-art complexity of $O(\log\Delta + 2^{O(\sqrt{\log\log n})})$ rounds in the LOCAL model, also by Ghaffari [11]. When considering the $(\Delta+1)$ coloring problem, the picture is somewhat puzzling. On the one hand, in the LOCAL model, $(\Delta+1)$ coloring is provably simpler then MIS. However, since all existing $o(\log\Delta)$-round algorithms for $(\Delta+1)$ coloring in the LOCAL model, use large messages, it is not even clear if the power of all-to-all communication in the congested clique model can compensate for its bandwidth limitation and outperform the LOCAL round complexity, not to say, even just match it. We note that on hind-sight, the situation for MIS in the congested clique was somewhat more hopeful (compared to coloring), for the following reason. The randomized phase of Ghaffari's MIS algorithm although being in the LOCAL model [11], used *small* messages and hence could be implemented in the CONGEST model with the same round complexity. To sum up, currently, there is no $o(\log\Delta)$-round algorithm for $(\Delta+1)$ coloring in any bandwidth restricted model, not even in the congested-clique.

Derandomization of LCL in the Congested-Clique Model. There exists a curious gap between the known complexities of randomized and deterministic solutions for local problems in the LOCAL model ([6, 20]). Censor et al. [5] initiated the study of *deterministic* LCL algorithms in the congested clique model by means of derandomization. The main take home message of [5] is as follows: for most of the classical LCL problems there are $\mathsf{poly}\log n$ round

[2] In the $(\deg+1)$ list coloring problem, each vertex v is given a palette with $\deg(v,G)+1$ colors.
[3] Using the bounded dependencies between decisions, this holds also with high probability.

randomized algorithms (even in the CONGEST model). For these algorithms, it is usually sufficient that the random choices made by vertices are *almost* independent. This implies that each round of the randomized algorithm can be simulated by giving all nodes a shared random seed of $\mathsf{poly}\log n$ bits. To dernadomize a single round of the randomized algorithm, nodes should compute (deterministically) a seed which is at least as "good"[4] as a random seed would be. To compute this seed, they need to estimate their "local progress" when simulating the random choices using that seed. Combining the techniques of conditional expectation, pessimistic estimators and bounded independence leads to a simple "voting"-like algorithm in which the bits of the seed are computed *bit-by-bit*. Once all bits of the seed are computed, it is used to simulate the random choices of that round. For a recent work on other complexity aspects in the congested clique, see [16].

1.1 Main Results and Our Approach

In this paper, we show that the power of all-to-all communication compensates for the bandwidth restriction of the model:

▶ **Theorem 1.** *There is a randomized algorithm that computes a* $(\Delta+1)$ *coloring in* $O(\log\log\Delta\cdot\log^* n)$ *rounds of the congested clique model, with high probability*[5]*.*

This significantly improves over the state-of-the-art of $O(\log\Delta)$-round algorithm for $(\Delta+1)$ in the congested clique model. It should also be compared with the round complexity of $(2^{O(\sqrt{\log\log n})})$ in the LOCAL model, due to [7]. As noted by the authors, reducing the LOCAL complexity to below $O((\log\log n)^2)$ requires a radically new approach.

Our $O(\log\log\Delta\cdot\log^* n)$ round algorithm is based on a recursive degree reduction technique which can be used to color any almost-clique graph with $\Delta=\widetilde{O}(n^{1-o(1)})$ in essentially $O(\log^* n)$ rounds.

▶ **Theorem 2.** *(i) For every* $\epsilon\in(0,1)$*, there is a randomized algorithm that computes a* $(\Delta+1)$ *coloring in* $O(\log(1/\epsilon)\cdot\log^* n)$ *rounds for graphs with* $\Delta=O((n/\log n)^{1-\epsilon})$*, (ii) This also yields a* $(\Delta+\Delta^{1/2+o(1)})$ *coloring in* $O(\log^* n)$ *rounds, with high probability.*

Claim (ii) improves over the $O(\Delta)$-coloring algorithm of [14] that takes $O(\log\log\log n)$ rounds in *expectation*. We also provide fast *deterministic* algorithms for $(\Delta+1)$ list coloring. The stat-of-the-art in the LOCAL model is $\widetilde{O}(\sqrt{\Delta})+\log^* n$ rounds due to Fraigniaud, Heinrich, Marc and Kosowski [10].

▶ **Theorem 3.** *There is a deterministic algorithm that computes a* $(\Delta+1)$ *coloring in* $O(\log\Delta)$ *rounds of the congested clique model and an* $O(\Delta^2)$ *coloring in* $O(1)$ *rounds.*

In [5], a deterministic algorithm for $(\Delta+1)$ coloring in $O(\log\Delta)$ round was shown only for the case where $\Delta=O(n^{1/3})$. Here it is extended for $\Delta=\Omega(n^{1/3})$. This is done by derandomizing an $(\Delta+1)$-list coloring algorithm which runs in $O(\log n)$ rounds. Similarly to [5], we first show that this algorithm can be simulated when the random choices made by the nodes are pairwise independent. Then, we enjoy the small search space and employ the method of conditional expectations. Instead of computing the seed bit by bit, we compute it in chunks of $\lfloor\log n\rfloor$ bits at a time, by fully exploiting the all-to-all power of the model.

[4] The random seed is usually shown provide a large progress in expectation. The deterministically computed seed should provide a progress at least as large as the expected progress of a random seed.

[5] As usual, by high probability we mean $1-1/n^c$ for some constant $c\geq 1$.

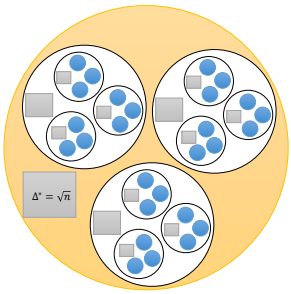


Figure 1 Illustration of the recursive sparsification. Gray boxes correspond to subgraphs with maximum degree $O(\sqrt{\Delta})$.

The Challenges and the Degree Reduction Technique. Our starting observation is that the CLP algorithm [7] can be implemented in $O(\log^* n)$ rounds in congested clique model for $\Delta = O(\sqrt{n})$. When $\Delta = O(\sqrt{n})$, using Lenzen's routing algorithm [18], each node can learn in $O(1)$ rounds, the palettes of all its neighbors along with the neighbors of its neighbors. Such knowledge is mostly sufficient for the CLP algorithm to go through.

To handle large degree graphs, we design a graph sparsification technique that essentially reduces the problem of $(\Delta+1)$ coloring for an arbitrarily large $\Delta = \widetilde{O}(n^{1-\epsilon})$ into $\ell = O(\log(1/\epsilon))$ (non-independent) subproblems. In each subproblem, one has to compute a $(\Delta'+1)$ coloring for a subgraph with $\Delta' = O(\sqrt{n})$, which can be done in $O(\log^* n)$ rounds, using a modification of the CLP algorithm, that we describe later on. Since there many dependencies between these ℓ sub-problems, it is required by our algorithm to solve them one-by-one, leading to a round complexity of $O(\log(1/\epsilon)\log^* n)$. See Figure 1 for an illustration of the recursion levels. To get an intuition into our approach and the challenges involved, consider an input graph G with maximum degree $\Delta = (n/\log n)^{1-\epsilon}$ and a palette $\mathsf{Pal}(G) = \{1,\ldots,\Delta+1\}$ given to each node in G. A natural approach (also taken in [14]) for handling a large degree graph is to decompose it (say, randomly) into k vertex disjoint graphs $G_1, G_2, \ldots, G_k$, allocate a *distinct* set of colors for each of the subgraphs taken from $\mathsf{Pal}(G)$ and solve the problem recursively on each of them, enjoying (hopefully) smaller degrees in each G_i. Intuitively, assigning a *disjoint* set of colors to each G_i has the effect of *removing* all edges connecting nodes in different subgraphs. Thus, the input graph G is sparsified into a graph $G' = \bigcup G_i$ such that a legal coloring of G' (with the corresponding palettes given to the nodes) is a legal coloring for G. The main obstacle in implementing this approach is that assigning a distinct set of $\Delta(G_i)+1$ colors to each of the G_i subgraph might be beyond the budget of $\Delta+1$ colors. Indeed in [14] this approach led to $O(\Delta)$ coloring rather than $(\Delta+1)$. To reduce the number of colors allocated to each subgraph G_i, it is desirable that the maximum degree $\Delta(G_i)$ would be as small as possible, for each G_i. This is exactly the problem of (k,p) *defective coloring* where one needs to color the graph with k colors such that the number of neighbors with the same color is at most p. To this point, the best defective coloring algorithm for large degrees is the randomized one: let each node pick a subgraph G_i (i.e., a color in the defective coloring language) uniformly at random. By a simple application of Chernoff bound, it is easy to see that the partitioning is "almost" perfect: w.h.p., for every i, $\Delta(G_i) \leq \Delta/k + \sqrt{\log n \cdot \Delta/k}$. Hence, allocating $\Delta(G_i)+1$ colors to each subgraphs consumes $\Delta + \widetilde{O}(\sqrt{\Delta k})$ colors. To add insult to injury, this additive penalty of $\widetilde{O}(\sqrt{\Delta k})$ is only for one recursion call!

It is interesting to note that the parameter k – number of subgraphs (colors) – plays a key role here. Having a large k has the benefit of sharply decreasing the degree (i.e., from Δ

to Δ/k). However, it has the drawback of increasing the standard deviation and hence the total number of colors used. Despite these opposing effects, it seems that for whatever value of k chosen, increasing the number of colors to $\Delta + \Delta^{\epsilon}$ is unavoidable.

Our approach bypasses this obstacle by partitioning only a large fraction of the vertices into small-degree subgraphs *but not all of them*. Keeping in mind that we can handle efficiently graphs with maximum degree $\sqrt{n}$, in every level of the recursion, roughly $1-1/\sqrt{\Delta}$ of the vertices are partitioned into subgraphs $G_1, \ldots, G_k$. Let $\Delta(G_i)$ be the maximum degree of G_i. The remaining vertices join a left-over subgraph G^*. The number of subgraphs, k, is chosen carefully so that allocating $\Delta(G_i)+1$ colors to each of the k subgraphs, consumes at most Δ colors, on the one hand; and that the degree reduction in each recursion level is large enough on the other hand. These subgraphs are then colored recursively, until all remaining subgraphs have degree of $O(\sqrt{n})$. Once all vertices in these subgraphs are colored, the algorithm turns to color the left-over subgraph G^*. Since the maximum degree in G^* is $O(\sqrt{n})$, it is tempting to use the CLP algorithm to complete the coloring, as this can be done in $O(\log^* n)$ rounds for such bound on the maximum degree. This is not so immediate for the following reasons. Although the degree of v in G^* is $O(\sqrt{n})$, the graph G^* cannot be colored independently (as at that point, we ran out of colors to be solely allocate to G^*). Instead, the coloring of G^* should agree with the coloring of the rest of the graph and each v might have $\Omega(\Delta) = n^{1-\epsilon}$ neighbors in G. At first glance, it seems that this obstacle is easily solved by letting each $v \in G^*$ pick a subset of $\deg(v, G^*) = O(\sqrt{n})$ colors from its palette (i.e., removing the colors taken by its neighbors in $G \setminus G^*$). Now, one can consider only the graph G^* with maximum degree $\sqrt{n}$, where each vertex has a palette of $\deg(v, G^*)+1$ colors. Unfortunately, this seemingly plausible approach has a subtle flaw: for the CLP algorithm it is essential that each vertex receives a palette with *exactly* $\Delta(G^*)+1$ colors. This is indeed crucial and as noted by the authors adopting their algorithm to a $(\deg+1)$ coloring algorithm is highly non-trivial and probably calls for a different approach.

In our setting, allocating each vertex $v \in G$ the exact same number of colors seems to be impossible as the number of available colors of each v depends on the number of its neighbors in $G \setminus G^*$, and this number has some fluctuations due to the random partitioning of the vertices. To get out of this impasse, we show that after coloring all vertices in $G \setminus G^*$, every vertex $v \in G^*$ has $r_v \in [\Delta(G^*) \pm (\Delta(G^*))^{3/5}]$ available colors in its palette where $r_v \geq \deg(v, G^*)$. In other words, all vertices can be allocated "almost" the same number of colors, but not exactly the same. We then carefully revise the basic definitions of the CLP algorithm and show that the analysis still goes through (upon minor changes) for this narrow range of variation in the size of the palettes.

Paper Organization. In Section 2, we explain how the CLP algorithm of [7] can be simulated in $O(\log^* n)$ congested-clique rounds when $\Delta = O(\sqrt{n})$. In Section 3.1, we illustrate the degree-reduction technique on the case where $\Delta = O((n/\log n)^{3/4})$. Section 3.2 extends this approach for $\Delta = O((n/\log n)^{1-\epsilon})$ for any $\epsilon \in (0,1)$, and Section 3.3 handles the general case and provides the complete algorithm. Finally, Section 4 discusses deterministic coloring algorithms. Missing proofs are deferred to the full version.

2 The Chang-Li-Pettie (CLP) Alg. in the Congested Clique

High-level Description of the CLP Alg. in the LOCAL Model. In the description below, we focus on the main randomized part of the CLP algorithm [13].

Harris-Schneider-Su algorithm is based on partitioning the graph into an ϵ-sparse subgraph and a collection of vertex-disjoint ϵ-dense components, for a given input parameter ϵ. Since

the CLP algorithm extends this partitioning, we next formally provide the basic definitions from [13]. For an $\epsilon \in (0,1)$, an edge $e=(u,v)$ is an ϵ-friend if $|N(u) \cap N(v)| \geq (1-\epsilon)\cdot\Delta$. The endpoints of an ϵ-friend edge are ϵ-friends. A vertex v is ϵ-dense if v has at least $(1-\epsilon)\Delta$ ϵ-friends, otherwise it is ϵ-sparse. A key structure that arises from the definition of ϵ dense vertices is that of ϵ-*almost clique* which is a connected component of the subgraph induced by the ϵ-dense vertices and ϵ-friend edges. The dense components, ϵ-*almost* cliques, have some nice properties: each component C has at most $(1+\epsilon)\Delta$ many vertices, each vertex $v \in C$ has $O(\epsilon\Delta)$ neighbors outside C (called *external* neighbors) and $O(\epsilon\Delta)$ vertices in C which are not its neighbors. In addition, C has weak diameter at most 2. Coloring the dense vertices consists of $O(\log_{1/\epsilon}\Delta)$ phases. The efficient coloring of dense regions is made possible by generating a random proper coloring inside each clique so that each vertex has a small probability of receiving the same color as one of its external neighbors. To do that, in each cluster a random permutation is computed and each vertex selects a tentative color from its palette excluding the colors selected by lower rank vertices. Since each component has weak diameter at most 2, this process is implemented in 2 rounds of the LOCAL model. The remaining *sparse* subgraph is colored using a Schneider-Wattenhofer style algorithm [23] within $O(\log(1/\epsilon))$ rounds.

In Chang-Li-Pettie algorithm the vertices are partitioned into $\ell = \lceil \log\log\Delta \rceil$ layers in decreasing level of density. This hierarchical partitioning is based on a sequence of ℓ sparsity thresholds $\epsilon_1,\ldots,\epsilon_\ell$ where $\epsilon_i = \sqrt{\epsilon_{i-1}}$. Roughly speaking, level i consists of the vertices which are ϵ_i-dense but ϵ_{i-1}-sparse. Instead of coloring the vertices layer by layer, the algorithm partitions the vertices in level i into large and small components and partitions the layers into $O(\log^*\Delta)$ strata. It then colors the vertices in $O(\log^*\Delta)$ phases, giving priority to vertices in small components. The algorithms used to color these vertices are of the same flavor of the dense-coloring procedure of Harris-Schneider-Su. The key benefit in having the hierarchical structure is that the dense-coloring procedure is applied for $O(1)$ many phases on each stratum, rather than applying it for $O(\log_{1/\epsilon}\Delta)$ phases as in [13].

An $O(\log^*\Delta)$-Round Alg. for $\Delta = O(\sqrt{n})$ in the Congested Clique. We next observe that the randomized part of the CLP algorithm [7] can be implemented in the congested clique model when $\Delta = O(\sqrt{n})$ within $O(\log^*\Delta)$ rounds. We note that we obtain a round complexity of $O(\log\Delta^*)$ rather than $O(\log^* n)$ as in [7], due to the fact that the only part of the CLP algorithm that requires $O(\log^* n)$ rounds was for coloring a subgraph with maximum constant degree. In the congested-clique model such a step can be implemented in $O(1)$ rounds using Lenzen's routing algorithm. We show:

▶ **Theorem 4.** *For every graph with maximum degree $\Delta = O(\sqrt{n})$, there is an $O(\log^*\Delta)$-round randomized algorithm that computes $(\Delta+1)$-list coloring in the congested clique model.*

The main advantage of having small degrees is that it is possible for each node to collect its 2-neighborhood in $O(1)$ rounds (i.e., using Lenzen's routing [18]). As we will see, this is sufficient in order to simulate the CLP algorithm in $O(\log^*\Delta)$ rounds. The hierarchical decomposition of the vertices depends on the computation of ϵ-dense vertices. By collecting the neighbors of its neighbors, every vertex can learn its ϵ-dense friends and based on that deduce if it is an ϵ-dense vertex for every ϵ. In particular, for every edge (u,v), v can learn the minimum i such that u and v are ϵ_i-friends. To allow each vertex v compute the ϵ-almost cliques to which it belongs, we do as follows. Each vertex v sends to each of its neighbors $N(v)$, the minimum ϵ_i such that u,v are ϵ_i-friends, for every $u \in N(v)$. Since the

weak diameter of each almost-clique is at most 2, each vertex has collected all the required information from its 2^{nd} neighborhood to locally compute its ϵ_i-almost cliques for every ϵ_i. Overall, each vertex sends $O(\Delta)$ messages and receives $O(\Delta^2) = O(n)$ messages, collection this information can be done in $O(1)$ rounds for all nodes, using Lenzen's routing algorithm. The next obstacle is the simulation of the algorithm that colors the ϵ-dense vertices. Since each ϵ-almost clique C has $(1+\epsilon)\Delta = O(\sqrt{n})$ vertices, we can make the leader of each such C learn the palettes of all the vertices in its clique as well as their neighbors in $O(1)$ rounds. The leader can then locally simulate the dense-coloring procedure and notify the output color to each of its almost-clique vertices. Finally, coloring the sparse regions in a Schneider-Wattenhofer style uses messages of size $O(\Delta)$ and hence each vertex is the target of $O(\Delta^2) = O(n)$ messages which again can be implemented in $O(1)$ many rounds. A more detailed description appears in the full version [22]. By the above description, we also have:

▶ **Corollary 5.** *Given q vertex-disjoint subgraphs $G_1, \ldots, G_q$ each with maximum degree $\Delta = O(\sqrt{n})$, a $(\Delta+1)$ coloring can be computed in $O(\log^* \Delta)$ rounds, for all subgraphs simultaneously.*

Handling Non-Equal Palette Size for $\Delta = O(\sqrt{n})$. The CLP algorithm assumes that each vertex is given a list of *exactly* $(\Delta+1)$ colors. Our coloring algorithms requires a more relaxed setting where each vertex v is allowed to be given a list of $r_v \in [\Delta - \Delta^{3/5}, \Delta+1]$ colors where $r_v \geq \deg(v,G)+1$. In this subsection we show:

▶ **Lemma 6.** *Given a graph G with $\Delta = O(\sqrt{n})$, if every vertex v has a palette with $r_v \geq \deg(v,G)+1$ colors and $r_v \in [\Delta - \Delta^{3/5}, \Delta+1]$ then a list coloring can be computed in $O(\log^* \Delta)$ rounds in the congested clique model.*

The key modification for handling non-equal palette sizes is in definition of ϵ-friend (which affects the entire decomposition of the graph). Throughout, let $q = \Delta^{3/5}$ and say[6] that u, v are (ϵ, q)-friends if $|N(u) \cap N(v)| \geq (1-\epsilon)\cdot(\Delta - q)$. Clearly, if u, v are ϵ-friends, they are also (ϵ, q)-friends. A vertex v is an (ϵ, q)-*dense* if it has at least $(1-\epsilon)\cdot(\Delta - q)$ neighbors which are (ϵ, q)-friends. An (ϵ, q)-almost clique is a connected component of the subgraph induced by (ϵ, q)-dense vertices and their (ϵ, q) friends edges. We next observe that for the ϵ-values used in the CLP algorithm, the converse is also true up to some constant. The full proof of Theorem 6 is in the full version.

▶ **Observation 7.** *For any $\epsilon \in [\Delta^{-10}, K^{-1}]$, where K is a large constant, and for $q = \Delta^{3/5}$, it holds that if u, v are (ϵ, q) friends, they are (2ϵ)-friends. Also, if v is an (ϵ, q)-dense, then it is 2ϵ-dense.*

3 $(\Delta+1)$-Coloring for $\Delta = \Omega(\sqrt{n})$

In this section, we describe a new recursive degree-reduction technique. As a warm-up, we start with $\Delta = O((n/\log n)^{3/4})$. We make use of the following fact.

▶ **Theorem 8** (Simple Corollary of Chernoff Bound). *Suppose $X_1, X_2, \ldots, X_\ell \in [0,1]$ are independent random variables, and let $X = \sum_{i=1}^{\ell} X_i$ and $\mu = \mathbb{E}[X]$. If $\mu \geq 5\log n$, then w.h.p. $X \in \mu \pm \sqrt{5\mu \log n}$, and if $\mu < 5\log n$, then w.h.p. $X \leq \mu + 5\log n$.*

[6] The value of q is chosen to be a bit above the standard deviation of $\sqrt{\Delta \log n}$ that will occur in our algorithm.

3.1 An $O(\log^* \Delta)$-round algorithm for $\Delta = O((n/\log n)^{3/4})$

The algorithm partitions G into $O(\Delta^{1/3})$ subgraphs as follows. Let $\ell = \lceil \Delta^{1/3} \rceil$. We define $\ell + 1$ subsets of vertices $V_1, \ldots, V_\ell, V^*$. A vertex joins each V_i with probability

$$p_i = 1/\ell - 2\sqrt{5 \log n}/(\Delta^{1/3} \cdot \ell),$$

for every $i \in \{1, \ldots, \ell\}$, and it joins V^* with the remaining probability of $p^* = 2\sqrt{5 \log n}/\Delta^{1/3}$.

Let $G_i = G[V_i]$ be the induced subgraph for every $i \in \{1, \ldots, \ell, *\}$. Using Chernoff bound of Theorem 8, the maximum degree Δ' in each subgraph G_i, $i \in \{1, \ldots, \ell\}$ is w.h.p.:

$$\Delta' \leq \Delta/\ell - 2\Delta^{2/3}\sqrt{5 \log n}/\ell + \sqrt{5\Delta \log n/\ell} \leq \Delta/\ell - 1$$

In the first phase, all subgraphs $G_1, \ldots, G_\ell$ are colored independently and simultaneously. This is done by allocating a distinct set of $(\Delta' + 1)$ colors for each of these subgraphs. Overall, we allocate $\ell \cdot (\Delta' + 1) \leq \Delta$ colors. Since $\Delta' = O(\Delta^{2/3}) = O(\sqrt{n})$, we can apply the $(\Delta' + 1)$-coloring algorithm of Theorem 5 on all the graphs $G_1, \ldots, G_\ell$ simultaneously. Hence, all the subgraphs $G_1, \ldots, G_\ell$ are colored in $O(\log^* \Delta)$ rounds.

Coloring the remaining left-over subgraph G^*. The second phase of the algorithm completes the coloring for the graph G^*. This coloring should agree with the colors computed for $G \setminus G^*$ computed in the previous phase. Hence, we need to color G^* using a *list* coloring algorithm. We first show that w.h.p. the maximum degree Δ^* in G^* is $O(\sqrt{n})$. The probability of vertex to be in G^* is $p^* = 2\sqrt{5 \log n}/\Delta^{1/3}$. By Chernoff bound of 8, w.h.p., $\Delta^* \leq p^* \cdot \Delta + \sqrt{5p^* \cdot \Delta \cdot \log n}$. Since $\Delta \leq (n/\log n)^{3/4}$, $\Delta^* = O(\sqrt{n})$. To be able to apply the modified CLP of Theorem 6, we show:

▶ **Lemma 9.** *Every $v \in G^*$ has at least $\Delta^* - (\Delta^*)^{3/5}$ available colors in its palette after coloring all its neighbors in $G \setminus G^*$.*

Proof. First, consider the case where $\deg(v, G) \leq \Delta - (\Delta^* - \sqrt{5\Delta^* \cdot \log n})$. In such case, even after coloring all neighbors of v, it still has an access of $\Delta^* - \sqrt{5\Delta^* \cdot \log n} \geq \Delta^* - (\Delta^*)^{3/5}$ colors in its palette after coloring $G \setminus G^*$ in the first phase. Now, consider a vertex v with $\deg(v, G) \geq \Delta - (\Delta^* - \sqrt{5\Delta^* \cdot \log n})$. Using Chernoff bound, w.h.p., $\deg(v, G^*) > (\Delta - (\Delta^* - \sqrt{\Delta^* \cdot 5 \log n})) \cdot p^* - \sqrt{5 \log n \Delta p^*} \geq \Delta^* - (\Delta^*)^{3/5}$. ◀

Also note that a vertex $v \in G^*$ has at least $\deg(v, G^*) + 1$ available colors, since all its neighbors in G^* are uncolored at the beginning of the second phase and initially it was given $(\Delta + 1)$ colors. Eventhough, $v \in G^*$ might have $\Omega(\Delta)$ neighbors not in G^*, to complete the coloring of G^*, by Theorem 9, after the first phase, each v can find in its palette $r \in [\Delta^* - (\Delta^*)^{3/5}, \Delta^* + 1]$ available colors and this sub-palette is sufficient for its coloring in G^*. Since $\Delta^* = O(\sqrt{n})$, to color G^* (using these small palettes), one can apply the $O(\log^* \Delta)$ round list-coloring algorithm of Theorem 6.

3.2 An $O(\log(1/\epsilon) \cdot \log^* \Delta)$-round algorithm for $\Delta = O((n/\log n)^{1-\epsilon})$

Let $N = n/(5 \log n)$. First assume that $\Delta \leq N/2$ and partitions the range of relevant degrees $[\sqrt{n}, N/2]$ into $\ell = \Theta(\log \log \Delta)$ classes. The y^{th} range contains all degrees in $[N^{1-1/2^y}, N^{1-1/(2^{y+1})}]$ for every $y \in \{1, \ldots, \ell\}$. Given a graph G with maximum degree $\Delta = O(N^{1-1/(2^{y+1})})$, Algorithm RecursiveColoring colors G in $y \cdot O(\log^* \Delta)$ rounds, w.h.p.

Step (I): Partitioning (Defective-Coloring). For $i \in \{0, \ldots, y-1\}$, in every level i of the recursion, we are given a graph G', with maximum degree $\Delta_i = O(N^{1-1/2^{y-i+1}})$, and a palette Pal_i of $(\Delta_i + 1)$ colors. For $i = 0$, $\Delta_0 = \Delta$ and the palette $\mathsf{Pal}_0 = \{1, \ldots, \Delta + 1\}$.

The algorithm partitions the vertices of G' into $q_i + 1$ subsets: $V'_1, \ldots, V'_{q_i}$ and a special left-over set V^*. The partitioning is based on the following parameters. Set $x = 2^{y-i}$ and

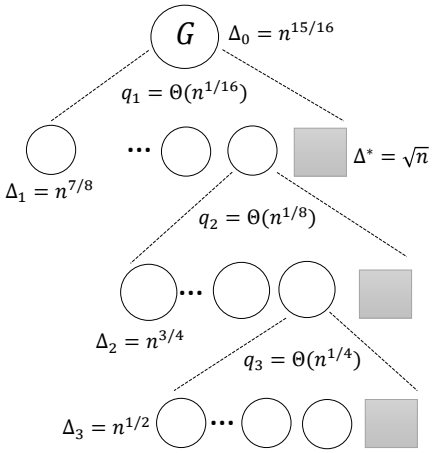
$$q_i = \lceil \Delta_i^{1/(2x-1)} \rceil \quad \text{and} \quad \delta_i = 2\sqrt{5 \log n} \cdot q_i^{3/2} / \sqrt{\Delta_i}.$$

Each vertex $v \in V(G')$ joins V'_j with probability $p_j = 1/q_i - \delta_i/(q_i)^2$ for every $j \in \{1, \ldots, q_i\}$, and it joins V^* with probability $p^* = \delta_i/q_i$. Note that $p^* \in (0, 1)$ as $x \geq 2$. For every $j \in \{1, \ldots, q_i\}$, let $G'_j = G'[V'_j]$ and let $G^* = G'[V^*]$.

Step (II): Recursive coloring of $G'_1, \ldots, G'_{q_i}$. Denote by $\widetilde{\Delta}_j$ to be the maximum degree in G'_j for every $j \in \{1, \ldots, q_i\}$ and by Δ^*, the maximum degree in G^*. The algorithm allocates a distinct subset of $(\widetilde{\Delta}_j + 1)$ colors from Pal_i for every $j \in \{1, \ldots, q_i\}$. In the analysis, we show that w.h.p. Pal_i contains sufficiently many colors for that allocation. The subgraphs $G'_1, \ldots, G'_{q_i}$ are colored recursively and simultaneously, each using its own palette. It is easy to see that the maximum degree of each G'_j is $O(N^{1-1/2^{y-i}})$ (which is indeed the desire degree for the subgraphs colored in level $i+1$ of the recursion).

Step (III): Coloring the left-over graph G^*. Since the algorithm already allocated at most Δ_i colors for coloring the G'_j subgraphs, it might run out of colors to allocate for G^*. This last subgraph is colored using a list-coloring algorithm only after all vertices of $G'_1, \ldots, G'_{q_i}$ are colored. Recall that Δ^* is the maximum degree of G^*. In the analysis, we show that w.h.p. $\Delta^* = O(\sqrt{n})$. For every $v \in G^*$, let $\mathsf{Pal}(v) \subseteq \mathsf{Pal}_i$ be the remaining set of available colors after coloring all the vertices in $V'_1, \ldots, V'_{q_i}$. Each vertex $v \in G^*$ computes a new palette $\mathsf{Pal}^*(v) \subseteq \mathsf{Pal}(v)$ such that: (i) $|\mathsf{Pal}^*(v)| \geq \deg(v, G^*) + 1$, and (ii) $|\mathsf{Pal}^*(v)| \in [\Delta^* - \Delta^{3/5}, \Delta^* + 1]$. In the analysis section, we show that w.h.p. this is indeed possible for every $v \in G^*$. The algorithm then applies the modified CLP algorithm, and G^* gets colored within $O(\log^* \Delta)$ rounds.

Example:



Assume that input graph G has maximum degree $\Delta_0 = n^{15/16}$. The algorithm partitions G into $k_0 = n^{1/15}$ subgraphs in the following manner. For sake of clarity, we omit logarithmic

factors in the explanation. With probability $1/(\Delta_0)^{1/2+o(1)}$, v joins a left-over subgraph G^*, and with the remaining probability it picks a subgraph $[1, k_0]$ uniformly at random. It is easy to see that the maximum degree in each of these subgraphs is at most $\Delta_1 = n^{7/8} + n^{7/16}$. A distinct set of Δ_1 colors from $[1, \Delta_0 + 1]$ is allocated to each of the k subgraphs. Each such subgraph is now partitioned into $k_1 = n^{1/8}$ subgraphs plus a left-over subgraph. This continues until all subgraphs have their degrees sharply concentrated around $\sqrt{n}$. At that point, the modified CLP algorithm can be applied on all the subgraphs in the last level ℓ. Once these subgraphs are colored, the left-over subgraphs in level $\ell - 1$ are colored, this continues until the final left-over subgraph of the first level is colored. We next provide a compact and high level description of the algorithm.

Algorithm 1 RecursiveColoring(G', Pal_i)

Input: Graph G' with maximum degree $\Delta_i = O(N^{1-1/2^{y-i+1}})$.
A palette Pal_i of $(\Delta_i + 1)$ colors (same for all nodes).

- Partitions G' into $q_i + 1$ vertex-disjoint subgraphs:
  - q_i vertex-subgraphs $G'_1, \ldots, G'_{q_i}$ with maximum degree $\Delta_{i+1} = O(N^{1-1/2^{y-i}})$.
  - Left-over subgraph G^* with maximum degree $\Delta^* = O(\sqrt{n})$.
- Allocate a distinct palette $\mathsf{Pal}_j \subset \mathsf{Pal}_i$ of $(\Delta_{i+1} + 1)$ colors for each $j \leq q_i$.
- Apply RecursiveColoring(G_j, Pal_j) for every $j \leq q_i$ simultaneously.
- Apply a $(\Delta_i + 1)$-list coloring restricted to Pal_i, to complete the coloring of $G[V^*]$.

Analysis.

▶ **Lemma 10.** *(i) For every $j \in \{1, \ldots, q_i\}$, w.h.p., $\widetilde{\Delta}_j = O(N^{1-1/2^{y-i}})$. (ii) One can allocate $(\widetilde{\Delta}_j + 1)$ distinct colors from Pal_i for each G'_j, $j \in \{1, \ldots, q_i\}$.*

Proof. Using Chernoff bound of Theorem 8, w.h.p., for every $j \in \{1, \ldots, q_i - 1\}$, the maximum degree $\widetilde{\Delta}_j$ in G'_j is at most $\widetilde{\Delta}_j = O(\Delta_i/q_i)$. Since $\Delta_i = O(N^{1-2^{y-i+1}})$, claim (i) follows. We now bound the sum of all colors allocated to these subgraphs:

$$\widetilde{\Delta}_j \leq \Delta_i/q_i - (\Delta_i \cdot \delta_i)/(q_i)^2 + \sqrt{5 \log n \cdot \Delta_i/q_i} \leq \Delta_i/q_i - 1 \ .$$

where the last inequality follows by the value of δ_i. We get that $\sum_{j=1}^{q_i}(\widetilde{\Delta}_j + 1) \leq \Delta_i$ and since Pal_i contains $\Delta_i + 1$ colors, claim (ii) follows. ◀

We next analyze the final step of the algorithm and begin by showing that, w.h.p., the maximum degree in the left-over graph G^* is $O(\sqrt{n})$. By Chernoff bound of Theorem 8, w.h.p., the maximum degree $\Delta^* \leq \Delta_i \cdot \delta_i/q_i + \sqrt{\log n \cdot \Delta_i \cdot \delta_i/q_i}$. Since $\Delta_i = O(N^{1-1/2^{y-i+1}})$, we get that $\Delta^* = O(\sqrt{n})$. We now claim:

▶ **Lemma 11.** *After coloring for all the vertices in $G'_1, \ldots, G'_{q_i}$, each vertex $v \in G^*$ has a palette $\mathsf{Pal}^*(v)$ of free colors such that (i) $|\mathsf{Pal}^*(v)| \geq \deg(u, G^*) + 1$, and (ii) $|\mathsf{Pal}^*(v)| \in [\Delta^* - (\Delta^*)^{3/5}), \Delta^* + 1]$.*

Proof. Since each vertex $v \in G'$ has a palette of size $(\Delta_i + 1) \geq \deg(v, G')$, after coloring all its neighbors in $G'_1, \ldots, G'_{q_i}$, it has at least $\deg(v, G^*) + 1$ free colors in its palette. Claim (ii) follows the same argument as in Theorem 9. We show that the palette of v has at least $\Delta^* - O(\sqrt{\Delta^* \cdot 5 \log n}) \geq (\Delta^* - (\Delta^*)^{3/5})$ available colors after coloring all the vertices in

$G \setminus G^*$. First, when $\deg(v, G') \leq \Delta_i - (\Delta^* - \sqrt{\Delta^* \cdot 5 \log n})$, then even after coloring all neighbors of v in G', it still has an access of $\Delta^* - \sqrt{\Delta^* \cdot 5 \log n}$ colors in its palette. Consider a vertex v with $\deg(v, G') \geq \Delta_i - (\Delta^* - \sqrt{\Delta^* \cdot 5 \log n})$. By Chernoff, w.h.p. it holds that:

$$\begin{aligned} \deg(v, G^*) &\geq (\Delta_i - (\Delta^* - \sqrt{\Delta^* \cdot 5 \log n})) \cdot p^* - \sqrt{5 \log n \cdot \Delta_i \cdot p^*} \\ &\geq \Delta_i \cdot \delta_i / q_i - O(\sqrt{\Delta_i \cdot \delta_i \log n / q_i}) \geq \Delta^* - (\Delta^*)^{3/5} \ . \end{aligned}$$

Hence, by combining with claim (i), the lemma follows. ◀

This completes the proof of Theorem 2(i).

$(\Delta + \Delta^{1/2+\epsilon})$ Coloring in Log-Star Rounds

▶ **Lemma 12.** *For any fixed $\epsilon \in (0, 1)$, one can color, w.h.p., a graph with $(\Delta + \Delta^{1/2+\epsilon})$ colors in $O(\log(1/\epsilon) \cdot \log^* \Delta)$ rounds.*

Proof. Due to Theorem 4, it is sufficient to consider the case where $\Delta = \Omega(\sqrt{n})$. Partition the graph into $k = \lfloor \Delta^\epsilon \rfloor$ subgraphs $G_1, \ldots, G_k$, by letting each vertex independently pick a subgraph uniformly at random. By Chernoff bound of Theorem 8, the maximum degree Δ_i in each subgraph G_i is at most $\Delta_i \leq \Delta^{1-\epsilon} + \Delta^{1/2-\epsilon/2} \cdot \sqrt{5 \log n}$. Allocate a distinct set of $\Delta_i + 1$ colors Pal_i to each subgraph G_i. Since $\Delta^{1-\epsilon} = O((n/\log n)^{1-\epsilon/2})$, we can apply Alg. RecursiveColoring on each of these subgraphs which takes $O(\log(1/\epsilon) \cdot \log^* \Delta)$ rounds. It is easy to see, that since the subgraphs are vertex disjoint, Alg. RecursiveColoring can be applied on all k subgraphs simultaneously with the same round complexity. Overall, the algorithm uses $\Delta + \Delta^{1/2+\epsilon}$ colors. ◀

3.3 $(\Delta + 1)$ Coloring Algorithm for General Graphs

For graphs G with $\Delta \leq n/(10 \log n)$, we simply apply Alg. RecursiveColoring. Plugging $\epsilon = 1/\log n$ in Theorem 2, we get that this is done in $O(\log \log \Delta \cdot \log^* \Delta)$ rounds. It remains to handle graphs with with $\Delta \in [n/(10 \log n), n]$. We partition the graph into $\ell = \lceil 5 \log n \rceil$ subgraphs $G_1, G_2, \ldots, G_\ell$ and a left-over graph G^* in the following manner. Each $v \in V$ joins G_i with probability $p = 1/\ell - 2\sqrt{5 \log n/(\Delta \cdot \ell)}$ for every $i \in \{1, \ldots, \ell\}$, and it joins G^* with probability $p^* = 1 - \ell \cdot p = \Theta(\log n/\sqrt{\Delta})$. By Chernoff bound, the maximum degree in G_i for $i \in \{1, 2, \ldots, \ell\}$ is $\Delta_i \leq \Delta/\ell - 2\sqrt{(\Delta \cdot 5 \log n)/\ell} + \sqrt{\Delta \cdot 5 \log n/\ell} \leq \Delta/\ell - 1$. Hence, we have the budget to allocate a distinct set Pal_i of Δ_i colors for each G_i.

The first phase applies Algorithm RecursiveColoring on each (G_i, Pal_i) simultaneously for every i. Since $\Delta_i = O(n/\log n)$, and the subgraphs are vertex-disjoint, this can be done in $O(\log \log \Delta \cdot \log^* \Delta)$ rounds for all subgraphs simultaneously (see Theorem 2(i)).

After all the vertices of $G \setminus G^*$ get colored, the second phase colors the left-over subgraph G^*. The probability of a vertex v to be in G^* is $O(\log n/\sqrt{\Delta}) = O(\log^2 n/\sqrt{n})$. Hence, G^* contains $O(\log^2 n \cdot \sqrt{n})$ vertices with high probability. We color G^* in two steps. First, we use the deg $+1$ list coloring Algorithm OneShotColoring from [2] to reduce the uncolored-degree of each vertex to be $O(\sqrt{n}/\log^2 n)$ with high probability. This can be done in $O(\log \log n)$ rounds. In the second step, the entire uncolored subgraph $G'' \subset G^*$ has $O(n)$ edges and can be solved locally in $O(1)$ rounds. Note that for each $v \in G''$, it is sufficient to consider a palette with $\deg(v, G'') + 1$ colors, and hence sending all these palettes can be done in $O(1)$ rounds as well. The complete proof of Theorem 1 is in the full-version.

Algorithm 2 FastColoring(G)

- If $\Delta \leq n/(10\log n)$, call RecursiveColoring$(G, [1, \Delta+1])$.
- Else, partition G into vertex-subgraphs as follows:
  - $G'_1, \ldots, G'_q$ with the maximum degree $\Theta(\Delta/\log n)$, and
  - a left-over subgraph G^* with maximum degree $\Delta^* = O(\sqrt{n})$.
- Allocate a distinct palette $\mathsf{Pal}_j \subset [1, \Delta+1]$ of $(\Delta(G_j)+1)$ colors for each $j \leq q_i$.
- Apply RecursiveColoring(G_j, Pal_j) for all $G_1, \ldots, G_q$ simultaneously.
- Apply a (deg +1)-list coloring algorithm on G^* for $O(\log\log n)$ rounds.
- Solve the remaining uncolored subgraph locally.

4 Deterministic Coloring Algorithms

In this section, we provide deterministic coloring algorithms using the tools of bounded inpdendent and conditional expectation introduced in [5].

▶ **Theorem 13.** *There is a deterministic $(\Delta+1)$ list coloring using $O(\log\Delta)$ rounds, in the congested clique model.*

In [5], a deterministic $(\Delta+1)$ coloring was presented only for graphs with maximum degree $\Delta = O(n^{1/3})$. Here, we handle the case of $\Delta = \Omega(n^{1/3})$. We derandomize the following simple $(\Delta+1)$-algorithm that runs in $O(\log n)$ rounds.

Algorithm 3 Round $\boldsymbol{i}$ of Algorithm **SimpleRandColor** (for node $\boldsymbol{v}$ with palette $\mathbf{Pal}_{\boldsymbol{v}}$)

- Let F_v be the set of colors taken by the colored neighbors of v.
- Pick a color c_v uniformly at random from $\mathsf{Pal}_v \setminus F_v$.
- Send colors to neighbors and if $c_v \neq 0$ and legal, halt.

▶ **Observation 14.** *The correctness of Algorithm* SimpleRandColor *is preserved, even if the coin flips are* pairwise-*independent.*

The goal of *phase* i in our algorithm is to compute a seed that would be used to simulate the random color choices of *round* i of Alg. SimpleRandColor. This seed will be shown to be good enough so that at least $1/4$ of the currently uncolored vertices, get colored when picking their color using that seed. Let V_i be the set of uncolored vertices at the beginning of phase i. We need the following construction of bounded independent hash functions:

▶ **Lemma 15.** *[25] For every $\gamma, \beta, d \in \mathbb{N}$, there is a family of d-wise independent functions $\mathcal{H}_{\gamma,\beta} = \left\{h : \{0,1\}^\gamma \to \{0,1\}^\beta\right\}$ such that choosing a random function from $\mathcal{H}_{\gamma,\beta}$ takes $d \cdot \max\{\gamma, \beta\}$ random bits, and evaluating a function from $\mathcal{H}_{\gamma,\beta}$ takes time poly(γ, β, d).*

For our purposes, we use Theorem 15 with $d = 2, \gamma = \log n, \beta = \log\Delta$ and hence the size of the random seed is $\alpha \cdot \log n$ bits for some constant α. Instead of revealing the seed bit by bit using the conditional expectation method, we reveal the assignment for a *chunk* of $z = \lfloor \log n \rfloor$ variables at a time. To do so, consider the i'th chunk of the seed $Y'_i = (y'_1, \ldots, y'_z)$. For each of the n possible assignments $(b'_1, \ldots, b'_z) \in \{0,1\}^z$ to the z variables in Y', we assign a leader u that represent that assignment and receives the conditional expectation values from

all the uncolored nodes V_i, where the conditional expectation is computed based on assigning $y'_1 = b'_1, \ldots, y'_z = b'_z$. Unlike the MIS problem, here the vertex's success depends only on its neighbors (i.e., and does not depend on its second neighborhood). Using the partial seed and the IDs of its neighbors, every vertex v can compute the probability that it gets colored based on the partial seed. It then sends its probability of being colored using a particular assignment $y'_1 = b'_1, \ldots, y'_z = b'_z$ to the leader u responsible for that assignment. The leader node u of each assignment $y'_1 = b'_1, \ldots, y'_z = b'_z$ sums up all the values and obtains the expected number of colored nodes conditioned on the assignment. Finally, all nodes send to the leader their computed sum and the leader selects the assignment $(b^*_1, \ldots, b^*_z) \in \{0,1\}^z$ of largest value. After $O(1)$ many rounds, the entire assignment of the $O(\log n)$ bits of the seed are revealed. Every yet uncolored vertex $v \in V_i$ uses this seed to simulate the random choice of Alg. SimpleRandColor, that is selecting a color in $\{0, 1, 2, \ldots, \Delta + 1\} \setminus F_v$ and broadcasts its decision to its neighbors. If the color $c_v \neq 0$ is legal, v is finally colored and it notifies its neighbors. By the correctness of the conditional expectation approach, we have that least $1/4 \cdot |V_i|$ vertices got colored. Hence, after $O(\log n) = O(\log \Delta)$ rounds, all vertices are colored. In the full version, we also show a deterministic $O(\Delta^2)$ coloring in $O(1)$ rounds.

References

1. Alkida Balliu, Juho Hirvonen, Janne H. Korhonen, Tuomo Lempiäinen, Dennis Olivetti, and Jukka Suomela. New classes of distributed time complexity. *CoRR*, abs/1711.01871, 2017. `arXiv:1711.01871`.
2. Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *Journal of the ACM (JACM)*, 63(3):20, 2016.
3. József Beck. An algorithmic approach to the lovász local lemma. i. *Random Structures & Algorithms*, 2(4):343–365, 1991.
4. Andrew Berns, James Hegeman, and Sriram V Pemmaraju. Super-fast distributed algorithms for metric facility location. In *International Colloquium on Automata, Languages, and Programming*, pages 428–439. Springer, 2012.
5. Keren Censor-Hillel, Merav Parter, and Gregory Schwartzman. Derandomizing local distributed algorithms under bandwidth restrictions. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 11:1–11:16, 2017.
6. Yi-Jun Chang, Tsvi Kopelowitz, and Seth Pettie. An exponential separation between randomized and deterministic complexity in the LOCAL model. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 615–624, 2016.
7. Yi-Jun Chang, Wenzheng Li, and Seth Pettie. An optimal distributed $(\Delta + 1)$ coloring algorithm? *arXiv preprint arXiv:1711.01361*, 2018.
8. Yi-Jun Chang and Seth Pettie. A time hierarchy theorem for the local model. *FOCS*, 2017.
9. Manuela Fischer and Mohsen Ghaffari. Sublogarithmic distributed algorithms for Lov\'asz local lemma, and the complexity hierarchy. *DISC*, 2017.
10. Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 625–634. IEEE, 2016.
11. Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 270–277. Society for Industrial and Applied Mathematics, 2016.

12 Mohsen Ghaffari. Distributed MIS via all-to-all communication. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 141–149, 2017.

13 David G Harris, Johannes Schneider, and Hsin-Hao Su. Distributed $(\delta + 1)$-coloring in sublogarithmic rounds. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 465–478. ACM, 2016.

14 James W Hegeman and Sriram V Pemmaraju. Lessons from the congested clique applied to mapreduce. *Theoretical Computer Science*, 608:268–281, 2015.

15 James W Hegeman, Sriram V Pemmaraju, and Vivek B Sardeshmukh. Near-constant-time distributed algorithms on a congested clique. In *International Symposium on Distributed Computing*, pages 514–530. Springer, 2014.

16 Janne H Korhonen and Jukka Suomela. Brief announcement: Towards a complexity theory for the congested clique. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 91. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

17 Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *Journal of the ACM (JACM)*, 63(2):17, 2016.

18 Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *PODC*, pages 42–50, 2013.

19 Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in o (log log n) communication rounds. *SIAM Journal on Computing*, 35(1):120–131, 2005.

20 Y Maus, F Kuhn, and M Ghaffari. On the complexity of local distributed graph problems. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 784–797, 2017.

21 Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.

22 Merav Parter. $(\delta + 1)$ coloring in the congested clique model. *arXiv*, 2018. `arXiv:1805.02457`.

23 Johannes Schneider and Roger Wattenhofer. A new technique for distributed symmetry breaking. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 257–266. ACM, 2010.

24 Gregory Schwartzman. Adapting sequential algorithms to the distributed setting. *arXiv preprint arXiv:1711.10155*, 2017.

25 Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012. `doi:10.1561/0400000010`.

CacheShuffle: A Family of Oblivious Shuffles

Sarvar Patel
Google LLC, Mountain View, USA
sarvar@google.com

Giuseppe Persiano
Google LLC, Mountain View, USA and Università di Salerno, Salerno, Italy
giuper@gmail.com

Kevin Yeo
Google LLC, Mountain View, USA
kwlyeo@google.com

Abstract

We consider *oblivious* two-party protocols where a *client* outsources N blocks of private data to a *server*. The client wishes to access the data to perform operations in such a way that the access pattern does not leak information about the data and the operations. In this context, we consider *oblivious shuffling* with a focus on bandwidth efficient protocols for clients with small local memory. In the shuffling problem, the N outsourced blocks, $B_1, \ldots, B_N$, are stored on the server according to an initial permutation π. The client wishes to reshuffle the blocks according to permutation σ. Oblivious shuffling is a building block in several applications that hide patterns of data access. In this paper, we introduce a generalization of the oblivious shuffling problem, the K-*oblivious shuffling* problem, and provide bandwidth efficient algorithms for a wide range of client storage requirements. The task of a K-oblivious shuffling algorithm is to shuffle N encrypted blocks that were previously randomly allocated on the server in such a way that an adversarial server learns nothing about either the new allocation of blocks or the block contents. The security guarantee must hold when an adversary has partial information on the initial placement of a subset of $K \leq N$ *revealed* blocks. The notion of oblivious shuffling is obtained for $K = N$.

We first study the N-oblivious shuffling problem and start by presenting CacheShuffleRoot, that is tailored for clients with $O(\sqrt{N})$ blocks of memory and uses approximately $4N$ blocks of bandwidth. CacheShuffleRoot is a 4x improvement over the previous best known N-oblivious shuffle for practical sizes of N. We then generalize CacheShuffleRoot to CacheShuffle that can be instantiated for any client memory size S and requires $O(N \log_S N)$ blocks of bandwidth. Next, we present K-oblivious shuffling algorithms that require $2N + f(K, S)$ blocks of bandwidth for all K and a wide range of S. Any extra bandwidth above the $2N$ lower bound depends solely on K and S. Specifically, for clients with $O(K)$ blocks of memory, we present KCacheShuffleBasic that uses exactly $2N$ blocks of bandwidth. For clients with memory $S \leq K$, we present KCacheShuffle, that requires $2N + O(K \log_S K)$ blocks of bandwidth. Finally, motivated by applications to ORAMs, we consider the case where the server stores D dummy blocks whose contents are irrelevant in addition to the N real blocks. For this case, we design algorithm KCacheShuffleDummy that shuffles $N + D$ blocks with K revealed blocks using $O(K)$ blocks of client storage and approximately $D + 2N$ blocks of bandwidth.

2012 ACM Subject Classification Security and privacy → Management and querying of encrypted data, Security and privacy → Privacy-preserving protocols, Information systems → Data encryption

Keywords and phrases Shuffling, Data-Oblivious Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.161

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 161; pp. 161:1–161:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version A full version of the paper can be found at [11], `https://arxiv.org/abs/1705.07069`.

1 Introduction

In recent years, cloud storage has increased in popularity due to the great benefits available to users. Outsourcing files to the cloud allows users to share documents conveniently. Users are able to access documents from many different machines without transferring data. The burden of data replication for recovery is placed on the storage provider. For many corporations, cloud storage is cost efficient compared to maintaining their own internal storage systems. With the widespread use of cloud storage, providing privacy for outsourced data becomes crucial. Unfortunately, encrypting outsourced data is not sufficient. Previous works [8, 9] show that learning the patterns of data access may leak information about the contents of encrypted data. This scenario provides motivation towards the study of *oblivious algorithms*.

In this paper, we focus on oblivious algorithms for *shuffling* data stored on a server. The ability to obliviously move blocks of encrypted data is an important primitive for privacy-conscious users of cloud storage. We consider the scenario where a client has outsourced the encryptions of N identically-sized blocks, $B_1, \ldots, B_N$, to a server. The N blocks are stored by the server on an array Source according to a permutation π. That is, an encryption of B_i is stored as $\mathsf{Source}[\pi(i)]$, for all $i = 1, \ldots, N$. The client wishes to shuffle the blocks into a server-stored destination array, Dest, according to a permutation σ.

An *oblivious shuffle* is an algorithm whose pattern of block movement and operations involving the server does not leak information about either σ or the contents of $B_1, \ldots, B_N$. If the client has N blocks of client memory available, oblivious shuffling is trivial. All N blocks are downloaded, decrypted, re-encrypted and re-uploaded to their correct location in Dest according to σ. Similarly, if bandwidth is unlimited, then oblivious shuffling is also trivial. The client streams all N blocks and keeps block $B_{\sigma^{-1}(1)}$ to be be placed into $\mathsf{Dest}[1]$ after all blocks have been streamed. The client repeats this algorithm for all $i = 2, \ldots, N$, which costs N^2 blocks of bandwidth. Our work focuses on oblivious shuffles that minimize bandwidth while the client has only a sublinear number of blocks of memory available.

Our Contributions. Our contribution is two-fold. We propose an obliviousness notion that abstracts the initial knowledge of π given to the adversarial server. Then, we present bandwidth efficient oblivious shuffles for any possible initial knowledge of π by the adversary and a wide range of client memory requirements.

Our work generalizes the notion of obliviousness studied in previous work by presenting the notion of *K-oblivious shuffling* that takes into account the knowledge of the adversary on the initial positioning of the blocks (that is, π). In the K-oblivious shuffling problem, the adversary fixes the position of $K \leq N$ blocks, which we denote as the *revealed* blocks. The client wishes to shuffle the N blocks into the server-stored destination array, Dest, without revealing information about σ or the contents of $B_1, \ldots, B_N$ to the adversarial server. The parameter K describes the difficulty of the problem. Intuitively, as K decreases, K-obliviously shuffling should be more efficient. In our work, we present algorithms whose bandwidth above inherent lower bounds depend only on K. For $K = N$, the notion of K-oblivious shuffling coincides with the original notion of a oblivious shuffling by Ohrimenko *et al.* [10], which assumes that the adversary has initial knowledge about all N input blocks.

The notion of K-oblivious shuffling is suitable in the context of Oblivious RAMs. Oblivious RAM (or ORAM) is a storage primitive introduced by Goldreich [4] that allows random

access to N encrypted blocks that are stored by an adversarial server. Many ORAMs use oblivious shuffling as a building block to move blocks without revealing information about the final positions of blocks as well as the contents of the blocks. In the majority of ORAM constructions, the adversary does not learn the position of all blocks that are going to be shuffled. By incorporating the initial amount of knowledge available to the adversary about the N input blocks, we can improve the bandwidth efficiency of ORAM constructions such as the Square Root ORAM [4, 5].

Before designing efficient K-oblivious shuffling algorithms, we revisit the original oblivious shuffling (that is, N-oblivious shuffling) problem and present new oblivious shuffling algorithms with improved bandwidth efficiency. Our algorithms use a client-stored *cache* to store blocks. The main technical difficulty in our algorithms is showing that the size of the cache remains small. We first apply this design principle in Section 3 by presenting an oblivious shuffling algorithm, CacheShuffleRoot, that uses approximately $4N$ blocks of bandwidth and $O(\sqrt{N})$ block of client storage. For similar client memory usage, the previous, state-of-the-art algorithm, the Melbourne Shuffle [10], uses about 4 times more bandwidth. We present a generalization of CacheShuffleRoot, $\mathsf{CacheShuffle}_S$, in Section 4 when the client has $S = \omega(\log N)$ blocks of available client storage. $\mathsf{CacheShuffle}_S$ uses $O(N \log_S N)$ blocks of bandwidth.

Next, we focus on designing K-oblivious shuffling algorithms when $K < N$. All previous oblivious shuffling algorithms have always considered the most difficult scenario when $K = N$. To our knowledge, our work is the first to separate the two problems. In Section 5, we present a simple K-oblivious shuffling algorithm, KCacheShuffleBasic, when the client has $O(K)$ blocks of available client storage. KCacheShuffleBasic uses exactly $2N$ blocks of bandwidth. In Section 6, we present $\mathsf{KCacheShuffle}_S$ for clients with only $O(S)$ blocks of client storage available. KCacheShuffle uses $2N + O(K \log_S K)$ blocks of bandwidth. For the case of $S = \sqrt{K}$, we present KCacheShuffleRoot, which uses approximately $2N + 4K$ blocks of bandwidth. In general, any K-oblivious shuffling algorithm must upload and download each block at least once meaning a lower bound of $2N$ blocks of bandwidth when $K > 0$. The amount of bandwidth used by all our K-oblivious shuffling algorithms beyond the lower bound only depends on K.

In many ORAM schemes, encryptions of dummy blocks are also outsourced to the server. The contents of a dummy block is irrelevant. However, it is important that an adversarial server cannot learn whether a block is dummy or not. In the full version [11], we consider a scenario where the client has outsourced the encryptions of D *dummy* blocks and N *real* blocks inspired by the work of Stefanov *et al.* [12]. Any K-oblivious shuffling algorithm could be used to perform shuffling by treating dummy blocks as real blocks. By using the fact that the contents of dummy blocks are irrelevant, we present KCacheShuffleDummy that uses approximately $D + 2N$ blocks of bandwidth when the client has $O(K)$ blocks of available client storage. Applying directly KCacheShuffleBasic would require $2(N + D)$ blocks of bandwidth. Therefore, KCacheShuffleDummy saves D blocks of bandwidth. The bandwidth savings come at the cost of a small amount of server computation.

We complement our theoretical analysis with experiments to show that our algorithms are of practical interest in the full version.

Previous works. The early approach to oblivious shuffling was based on oblivious sorting algorithms which could be immediately derived from any sorting circuit. To evaluate a compare-exchange gate of a sorting circuit, the client downloads the two input encrypted blocks, decrypts and re-encrypts both blocks and uploads them in the correct order. Batcher's

sort [2] is considered the most practical sorting circuit even though it has asymptotic cost of $O(N \log^2 N)$. Sorting networks such as AKS [1] and Zig-Zag [7] have $O(N \log N)$ size but large hidden constants. Randomized Shellsort [6] is another $O(N \log N)$ oblivious sort with smaller hidden constants but larger depth. Waksman [13] presents a circuit for oblivious shuffling of size $O(N \log N)$. Oblivious shuffling based on sorting circuits is interesting because the client only needs to store $O(1)$ blocks at any point. However, sorting circuits incur a large $\Omega(N \log N)$ blocks of bandwidth cost. The first oblivious shuffling algorithm not based on sorting circuits, the *Melbourne Shuffle*, was introduced by Ohrimenko *et al.* [10]. The Melbourne Shuffle uses $O(N)$ bandwidth while only requiring $O(\sqrt{N})$ blocks to be stored on the client at any time.

In the table below, we compare our algorithms with the Melbourne Shuffle [10].

Table 1 N denotes the number of blocks. Algorithm KCacheShuffleDummy receives D additional dummy blocks, for a total of $N + D$ blocks. Algorithm KCacheShuffleRoot is obtained from algorithm KCacheShuffle by setting $S = \sqrt{N}$. For all algorithms, server storage is cN, for a small constant c.

		Client Storage	Bandwidth
$K = N$	Melbourne Shuffle [10]	$O(\sqrt{N})$	$\approx 18N$
	CacheShuffleRoot	$O(\sqrt{N})$	$(4+\epsilon)N$
	CacheShuffle	$O(S)$	$O(N \log_S N)$
General K	KCacheShuffleBasic	$O(K)$	$2N$
	KCacheShuffleRoot	$O(\sqrt{K})$	$2N + (4+\epsilon)K$
	KCacheShuffle	$O(S)$	$2N + O(K \log_S K)$
	KCacheShuffleDummy	$O(K)$	$D + (2+\epsilon)N$

An algorithm similar to CacheShuffleRoot was developed in parallel and independent work in [3] for the context of privacy-preserving software monitoring.

2 Definitions

In this section, we give formal definition for *shuffling algorithms* and *oblivious shuffling algorithms*. Our reference scenario is a cloud storage model where a *client* outsources the storage of N identically-sized data blocks to a *server* with the capacity to store $M \geq N$ blocks.

We assume the N data blocks are uploaded by the Setup algorithm. As input, Setup receives N data blocks, $\mathbb{B} = (B_1, \ldots, B_N)$, each of size B and a *permutation* $\pi : [N] \to [N]$. Setup randomly selects an encryption key, `key`, whose length is determined by the security parameter λ, for a symmetric encryption scheme and uploads an encryption of each of the N data blocks to the server according to π. Formally, an encryption of B_i under `key` will be stored at server location $\pi(i)$, for all $i \in [N]$. Once Setup has uploaded all N blocks, an adversary $\mathcal{A}$ is allowed to learn the initial position of a subset of the data blocks, which we denote Revealed $\subseteq [N]$. For each index $i \in$ Revealed, $\pi(i)$ is revealed to $\mathcal{A}$. We call the data blocks in Revealed, the *revealed* data blocks.

The shuffling algorithm takes as input the encryption key `key`, the permutation map π, the set of revealed data blocks Revealed, and a new permutation σ. The task of the shuffling algorithm is to re-permute the N data blocks stored on the server according to the new permutation map σ. In particular, we are interested in *oblivious shuffling* algorithms. Roughly speaking, oblivious shuffles hide information about both the contents of $B_1, \ldots, B_N$

and σ even when the adversary has partial information on π (restricted to the input set Revealed) and observes the blocks movements performed by the shuffling algorithm.

For convenience, we will abuse the notation of array indexing and function evaluation throughout our work. For any array A and index i, $A[i]$ refers to the element stored at location i in A. For a set of indices S, we define $A[S] := \{A[s] : s \in S\}$. Similarly for any function f and input set S, we define $f(S) := \{f(s) : s \in S\}$.

2.1 Mechanics of the Shuffling Algorithm

A shuffling algorithm receives as input the initial permutation π, the final permutation σ and the set Revealed. A shuffling algorithm proceeds in steps. The state after the q-th step is described by a *server allocation map* $\rho_q : [M] \to [N] \cup \{\perp\}$ and by a *client allocation map* $L_q : [S] \to [N] \cup \{\perp\}$. Each allocation map specifies the block currently occupying each of the M server locations and S client locations, respectively. More precisely, $\rho_q(j) = i$ means that, after the q-th step is performed, the j-th server location contains an encryption of the B_i. If instead $\rho_q(j) = \perp$, then an encryption of a dummy block is stored at location j. Similar statements are true for the client allocation map, L_q.

When a shuffling algorithm starts, the server allocation map ρ_0 coincides with permutation map π on the first N storage location of the M server memory locations. That is, $\rho_0(i) = \pi^{-1}(i)$ for all $i = 1, \ldots, N$. The remaining $N - M$ locations contain encryptions of dummy blocks. All S client block locations initially contain dummy blocks. That is, $L_0(i) = \perp$ for all $i = 1, \ldots, S$. During each step, a shuffling algorithm can perform either a *move* operation or a *server computation* operation. A move operation can be either a *download* or an *upload* move and they modify the state as follows. All download and upload operations are associated with a *source* and a *destination*. Suppose the q-th operation is a download with source s_q and destination d_q. Then, an encryption of block $B_{\rho_{q-1}(s_q)}$ stored at server location s_q is copied to location d_q of client storage. Before storing in client storage, the block is decrypted and re-encrypted. As a consequence, the ρ_q is identical to ρ_{q-1}. On the other hand, L_q is identical to L_{q-1} except that $L_q(d_q) = \rho_{q-1}(s_q)$. If the q-th operation was an upload with source s_q and destination d_q, then the encryption of block $B_{L_{q-1}(s_q)}$ stored at client location s_q is copied to location d_q of server storage. In this case, L_q is identical to L_{q-1}. However, S_q is obtained by modifying S_{q-1} such that $S_q(d_q) = L_{q-1}(s_q)$. Shuffle algorithms may perform upload moves with the source as $\perp$. In this case, an encryption of a dummy block is uploaded to the destination location of server storage.

A server computation operation is specified by the server performing a circuit that uses a subset of the blocks as input and copies the circuit's output to a subset of server storage locations. In our shuffling algorithms, server computation operations consist of homomorphic operations on block ciphertexts, which reduce bandwidth by using small amounts of server computation. The circuit description sent by the client must be considered as bandwidth.

2.2 Efficiency Measures

In our work, we consider three measures of efficiency for a shuffling algorithm: bandwidth, client memory and server memory. Our work focuses on minimizing bandwidth for a given amount of client memory, which is typically sublinear. While we do not prioritize optimizing server storage, all our shuffling algorithms use server storage that is linear with small constants in the number N of blocks. For shuffling algorithms with small client storage, we assume that the input permutations π and σ are space-efficient pseudorandom permutations.

Throughout this work, we consider blocks as our unit of measure. For example, if a shuffling algorithm uses T bandwidth, it means the shuffling algorithm uses T blocks of bandwidth.

2.3 Obliviousness

We define a *transcript* produced an execution of a shuffling algorithm Sh as the information seen by the adversarial server. The transcript consists of the initial encryptions of the data blocks as stored in server memory, the ordered list of the sources of all download moves, the ordered list of the destinations of all upload moves as well as the encryptions of the uploaded block and the list of circuits uploaded by the client. We stress that a transcript only contains the server locations that are involved in each move. That is, the transcript only contains the source for downloads and the destination for uploads. The transcript does not contain the involved client locations in each upload and download. The transcript models that an adversarial server $\mathcal{A}$ cannot observe information about the client's storage such as the destination of a download and the source of an upload.

Using the definition of a transcript, we now formally define an oblivious shuffling algorithm. For every sequence of N blocks $\mathbb{B} = (B_1, \ldots, B_N)$, every subset $\mathsf{Revealed}$ of revealed blocks, and every pair of permutations (π, σ), a shuffling algorithm Sh naturally induces a probability distribution $\mathcal{T}_{\mathsf{Sh}}(\mathbb{B}, \pi, \sigma, \mathsf{Revealed})$ over all possible transcripts. We capture the notion of a *K-oblivious shuffling* algorithm by the following game $\mathsf{OSGame}^{\mathcal{A}}_{\mathsf{Sh}}$ for a shuffling algorithm Sh between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. In the formalization of our security notion, the adversary $\mathcal{A}$ receives partial information on the starting permutation map π to reflect the fact that the shuffling algorithm Sh might be part of a larger protocol whose execution leaks information about π. More precisely, $\mathcal{A}$ chooses the initial server locations of a subset of the N data blocks, $\mathsf{Revealed}$. We parametrize the security notion by the cardinality of the set $\mathsf{Revealed}$, which we denote as K. We say that an adversary $\mathcal{A}$ is *K-restricted* if it specifies the location of at most K blocks. That is, $|\mathsf{Revealed}| \leq K$. $\mathcal{C}$ fills in the remaining $N - |\mathsf{Revealed}|$ locations randomly under the constraint that each of the N blocks appears in exactly one location on the server. Then, $\mathcal{A}$ proposes two sequences of N blocks, $\mathbb{B}_0$ and $\mathbb{B}_1$, and two permutations, σ_0 and σ_1. $\mathcal{C}$ randomly picks $b \in \{0, 1\}$ and samples a transcript $\texttt{trans}$ according to $\mathcal{T}_{\mathsf{Sh}}(\mathbb{B}_b, \pi, \sigma_b, \mathsf{Revealed})$. On input $\texttt{trans}$, $\mathcal{A}$ outputs its guess b' for b. We present the formal definition below.

▶ **Definition 1.** For a shuffle algorithm Sh, we define the game $\mathsf{OSGame}^{\mathcal{A}}_{\mathsf{Sh}}(N, \lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ for N data blocks and security parameter λ as follows:

1. $\mathcal{A}$ chooses a subset $\mathsf{Revealed} \subseteq [N]$, specifies $\pi(i)$ for each $i \in \mathsf{Revealed}$, and sends $(\mathsf{Revealed}, \pi(\mathsf{Revealed})$ to $\mathcal{C}$;
2. $\mathcal{A}$ chooses two pairs $(\mathbb{B}_0, \sigma_0)$ and $(\mathbb{B}_1, \sigma_1)$ and sends them to $\mathcal{C}$;
3. $\mathcal{C}$ completes the permutation π by randomly choosing the values at the point left unspecified by $\mathcal{A}$;
4. $\mathcal{C}$ randomly selects $b \leftarrow \{0, 1\}$ and sends $\mathcal{A}$ transcript $\texttt{trans}$ drawn according to $\mathcal{T}_{\mathsf{Sh}}(\mathbb{B}_b, \pi, \sigma_b, \mathsf{Revealed})$;
5. $\mathcal{A}$ on input $\texttt{trans}$ outputs b';

The game outputs 1 iff $b = b'$.

▶ **Definition 2** (K-oblivious shuffling)**.** We say that shuffling algorithm Sh is a *K-oblivious shuffling* algorithm if for all K-restricted probabilistically polynomial time adversaries $\mathcal{A}$,

and for all $N = \text{poly}(\lambda)$

$$\Pr[\mathsf{OSGame}^{\mathcal{A}}_{\mathsf{Sh}}(N, \lambda) = 1] \leq \frac{1}{2} + \text{negl}(\lambda).$$

We refer to N-oblivious shuffling algorithms as just *oblivious shuffling* algorithms.

2.4 Move-Based shuffling Algorithms

Move-based algorithms are shuffling algorithm that only perform move operations between client and server storage; that is, the server does not perform any computation on the stored encrypted blocks. To prove K-obliviousness for move-based algorithms, it suffices to show that for every random π and for every subset $\mathsf{Revealed} \subseteq [N]$ containing at most K indices, the probability distribution consisting of both the sources of downloads and the destinations of uploads are independent of σ conditioned on $\mathsf{Revealed}$ and $\pi(\mathsf{Revealed})$. More precisely, we define $\mathcal{M}_{\mathsf{Sh}}(\mathbb{B}, \pi, \sigma, \mathsf{Revealed})$ as the distribution of the move transcript $\mathtt{Mtrans}$ obtained from a transcript $\mathtt{trans}$ drawn from $\mathcal{T}_{\mathsf{Sh}}(\mathbb{B}, \pi, \sigma, \mathsf{Revealed})$ by removing the encryption of server-stored blocks and the encrypted blocks associated with upload moves. This is equivalent to considering the blocks as opaque indistinguishable balls. It is straightforward to prove that if the encryption scheme is IND-CPA and $\mathcal{T}_{\mathsf{Sh}}(\mathbb{B}, \pi, \sigma, \mathsf{Revealed})$ is independent of σ conditioned on $\mathsf{Revealed}$ and $\pi(\mathsf{Revealed})$, then Sh is a K-oblivious shuffling algorithm.

In the full version, we show that move-based K-oblivious shuffles have an inherent lower bound of $2N$ blocks of bandwidth when $K \geq 1$.

3 Oblivious Shuffling with $O(\sqrt{N})$ Client Memory

In this section, we describe $\mathsf{CacheShuffleRoot}$, which is an oblivious shuffling algorithm that uses $O(\sqrt{N})$ blocks of client storage except with negligible probability. For every $\epsilon > 0$, we describe an algorithm $\mathsf{CacheShuffleRoot}_\epsilon$ that uses $(3 + \epsilon/2)N$ blocks of server storage, $(4 + \epsilon)N$ blocks of bandwidth and $\delta_\epsilon\sqrt{N}$ blocks of client storage except with negligible probability in N. The value δ_ϵ is a constant that depends solely on ϵ and not from N. Whenever ϵ is clear from the context, we will just call the algorithm $\mathsf{CacheShuffleRoot}$.

3.1 Intuition

We start by describing a simple algorithm, which is insufficient to perform oblivious shuffling. However, the algorithm provides a general idea of our techniques.

Let us recall the inputs to oblivious shuffling. The client is provided permutations π and σ. Note that in our security model, σ is provided privately to the client and hidden from the adversarial server. On the server, N block ciphertexts are stored in the array Source. An encryption of block B_i is stored at $\mathsf{Source}[\pi(i)]$, for all $i = 1, \ldots, N$. At the termination of an oblivious shuffling, an encryption of block B_i should appear at $\mathsf{Dest}[\sigma(i)]$, for all $i = 1, \ldots, N$.

We now describe a simple, but incorrect, algorithm to provide intuition of our techniques. The N indices of Dest are randomly partitioned into $q := \sqrt{N}$ *destination buckets*, $\mathsf{destInd}_1, \ldots, \mathsf{destInd}_q$. Each $i \in [N]$ is assigned to a uniformly and independently chosen destination bucket. The indices of Source are partitioned into $s := \sqrt{N}$ *source groups*, $\mathsf{srcInd}_1, \ldots, \mathsf{srcInd}_s$, of exactly $N/s = \sqrt{N}$ blocks. The j-th source group consists of blocks located in $\mathsf{Source}[(j-1)N/s + 1, \ldots, j \cdot N/s]$, for all $j = 1, \ldots, s$. Finally, there will be q temporary server-stored arrays, $\mathsf{temp}_1, \ldots, \mathsf{temp}_q$, each of size s and initially empty.

On average, each destination bucket will contain $N/q = \sqrt{N}$ indices. Furthermore, each destination bucket will receive one block from each of the s source groups according to σ

in expectation. For now, let us suppose that each destination bucket receives exactly one block from each of the s source groups. In this case, we show that oblivious shuffling can be easily performed. Our algorithm would process each of the s source groups one at a time. When processing srcInd_j, the algorithm downloads all $\sqrt{N}$ blocks of $\mathsf{Source}[\mathsf{srcInd}_j]$. Then, exactly one block is uploaded to each of $\mathsf{temp}_1, \ldots, \mathsf{temp}_q$ from srcInd_j. In particular, the block to temp_k is the only block from srcInd_j that will be placed to $\mathsf{Dest}[\mathsf{destInd}_k]$ according to σ. After all s source groups have been processed, each temp_k contains all the blocks that are to be placed in a location of $\mathsf{Dest}[\mathsf{destInd}_k]$ but in the incorrect order. The algorithm performs another q phases to process each of $\mathsf{temp}_1, \ldots, \mathsf{temp}_q$. When processing temp_k, all s blocks of temp_k are downloaded and re-uploaded to their correct locations in $\mathsf{Dest}[\mathsf{destInd}_k]$ in any arbitrary order (such as as increasing order of $\mathsf{destInd}_k$).

This algorithm is easily seen to be oblivious but, unfortunately, it is unlikely that each destination bucket receives exactly the expected number blocks from each group. We thus present algorithm CacheShuffleRoot, which modifies the above algorithm to handle variances in expectation. CacheShuffleRoot does not expect each source group to contain exactly one block that should be uploaded to each destination bucket. Any time more than one block from a source group should be uploaded to a destination bucket, the extra blocks will be stored in a cache in the client's private storage. To ensure that the client's cache does not grow too large, we slightly increase the number of destination buckets from $\sqrt{N}$ to $(1 + \epsilon/2)\sqrt{N}$, for any constant $\epsilon > 0$. Let us now proceed more formally.

3.2 CacheShuffleRoot Description

For any constant $\epsilon > 0$, we describe algorithm CacheShuffleRoot. As input, the client receives permutations π and σ. On the server, the N blocks are stored in the *source array* Source according to π. That is, a ciphertext of block B_i appears in $\mathsf{Source}[\pi(i)]$, for all $i = 1, \ldots, N$. CacheShuffleRoot will output a server-stored *destination array*, Dest, such that an encryption of block B_i is stored as $\mathsf{Dest}[\sigma(l)]$, for all $l = i, \ldots, N$.

CacheShuffleRoot proceeds by partitioning the indices of Source into $s := \sqrt{N}$ *source groups* $\mathsf{srcInd}_1, \ldots, \mathsf{srcInd}_s$. For all $j = 1, \ldots, s$, srcInd_j consists of blocks stored at locations $\mathsf{Source}[(j-1)s + 1, \ldots, js]$. The N indices of the destination array Dest are randomly partitioned into $q := (1 + \epsilon/2)\sqrt{N}$ *destination buckets*, $\mathsf{destInd}_1, \ldots, \mathsf{destInd}_q$. Formally, index $i \in [N]$ is assigned to uniformly and independently chosen destination bucket. CacheShuffleRoot also initializes q server-stored *temporary arrays*, $\mathsf{temp}_1, \ldots, \mathsf{temp}_q$ and q client-stored *caches*, $\mathsf{Q}_1, \ldots, Q_q$. Each temporary array initially contains s empty block locations and each cache is initially empty.

CacheShuffleRoot consists of two phases: Spray and Recalibrate. The Spray phase consists of s rounds, one for each of the s source groups. In the j-th Spray round, the algorithm downloads all blocks in $\mathsf{Source}[\mathsf{srcInd}_j]$. Each downloaded ciphertext is decrypted and re-encrypted with fresh randomness. Each downloaded block is placed into one of the q caches according to their placement by σ. If block B_i is downloaded, then the re-encryption of B_i is placed into Q_k such that $\sigma(i) \in \mathsf{destInd}_k$. After all blocks of $\mathsf{Source}[\mathsf{srcInd}_j]$ are placed into their respective queues, exactly one block from each of $\mathsf{Q}_1, \ldots, Q_q$ is uploaded to the server. In particular, one block from Q_k is uploaded to temp_k. If any Q_k is empty, a *dummy* block containing an encryption of any arbitrary block is uploaded instead. After all s rounds of the Spray phase are completed, each of $\mathsf{temp}_1, \ldots, \mathsf{temp}_q$ contains exactly s blocks. Furthermore, there might be some blocks remaining in $\mathsf{Q}_1, \ldots, \mathsf{Q}_q$. Every block that is assigned to a location of $\mathsf{Dest}[\mathsf{destInd}_k]$ according to σ appears in either Q_k or temp_k.

The Recalibrate phase will simply rearrange all non-dummy blocks of Q_k and temp_k into the correct locations of $\mathsf{Dest}[\mathsf{destInd}_k]$. Formally, Recalibrate operates in q rounds, one round for each pair of $(\mathsf{Q}_1, \mathsf{temp}_1), \ldots, (\mathsf{Q}_q, \mathsf{temp}_q)$. In the k-th round, all s blocks of temp_k are

downloaded. All non-dummy blocks of temp_k are decrypted and re-encrypted before being placed into Q_k. At this point, all blocks assigned to $\mathsf{Dest}[\mathsf{destInd}_k]$ according to σ appear in Q_k. All blocks are simply uploaded to $\mathsf{Dest}[\mathsf{destInd}_k]$ in any arbitrary order (such as increasing order of $\mathsf{destInd}_k$). After all q rounds of Recalibrate, CacheShuffleRoot finishes executing.

The proof of the following theorem can be found in the full version.

▶ **Theorem 3.** CacheShuffleRoot *is an oblivious shuffle algorithm that uses* $(4+\epsilon)N$ *blocks of bandwidth,* $(3+\epsilon/2)N$ *blocks of server memory and* $O(\sqrt{N})$ *blocks of client memory except with probability negligible in* N*.*

4 Oblivious Shuffling with Smaller Client Memory

In this section, we generalize algorithm CacheShuffleRoot from Section 3. For any $S = \omega(\log N)$, we provide an oblivious shuffling algorithm that uses $O(S)$ blocks of client memory and $O(N \log_S N)$ blocks of bandwidth.

Let us take another look at CacheShuffleRoot. Once CacheShuffleRoot completes the Spray phase, all the blocks that should be placed into $\mathsf{Dest}[\mathsf{destInd}_k]$ according to σ are stored either in temp_k or Q_k. Afterwards the k-th round of Recalibrate arranges all non-dummy blocks in temp_k and Q_k into their correct location in $\mathsf{Dest}[\mathsf{destInd}_k]$. Each round of Recalibrate requires $|\mathsf{temp}_k| + |\mathsf{Q}_k|$ blocks of client memory. Therefore, the key to a oblivious shuffling using less blocks of client memory requires a modification to the Spray phase so that less blocks are placed into each temp_k and Q_k. We present RSpray, a modification of Spray, to achieve smaller server-stored temporary arrays and client-stored caches. Additionally, the output of RSpray is structured such that RSpray may be recursively applied.

4.1 RSpray Description

First, we abstract the input to RSpray to handle recursive applications. As input, RSpray receives a server-stored *source array*, RSource, of n block ciphertexts. In addition, the client privately receives, $\mathsf{destInd} \subseteq [N]$, of d *destination indices*. For every i such that $\sigma(i) \in \mathsf{destInd}$, an encryption of block B_i appears in RSource. The remaining $n - d$ ciphertexts of RSource are dummy blocks.

RSpray is parameterized by the number of blocks of client storage available, which we denote by S. As output, RSpray outputs S *temporary arrays*, $\mathsf{temp}_1, \ldots, \mathsf{temp}_S$, which contain block ciphertexts as well as a partition of destInd into S *destination buckets*, $\mathsf{destInd}_1, \ldots, \mathsf{destInd}_S \subset \mathsf{destInd}$. Furthermore, RSpray guarantees that if $\sigma(i) \in \mathsf{destInd}_k$, then an encryption of B_i will appear in temp_k. To keep the same notation as Section 3, we set $q = S$.

We now formally describe RSpray. RSpray partitions destInd into q destination buckets, $\mathsf{destInd}_1, \ldots, \mathsf{destInd}_q$. Each index $d \in \mathsf{destInd}$ is assigned to one of the q subsets uniformly and independently at random. RSpray will initialize q server-stored temporary arrays, $\mathsf{temp}_1, \ldots, \mathsf{temp}_q$. Each temp_k will contain $s := n/q$ empty block locations. RSpray also initializes q client-stored caches, $\mathsf{Q}_1, \ldots, \mathsf{Q}_q$. All q caches are initially empty. Finally, RSpray partitions RSource into s server-stored *source buckets*, $\mathsf{srcInd}_1, \ldots, \mathsf{srcInd}_s$. Each block ciphertext in RSource is assigned to one of the s source buckets uniformly and independently at random. Unlike Spray, RSpray initializes the s source buckets randomly.

RSpray performs s rounds, one for each of $\mathsf{srcInd}_1, \ldots, \mathsf{srcInd}_s$. In the j-th round, RSpray downloads all blocks of srcInd_j. All non-dummy blocks are decrypted and re-encrypted.

If block B_i is downloaded, then the new encryption of B_i is placed into Q_k such that $\sigma(i) \in \mathsf{destInd}_k$. After all blocks in srcInd_j are placed into their corresponding cache, exactly one block ciphertext is uploaded from each $\mathsf{Q}_1, \ldots, \mathsf{Q}_q$ to the server. In particular, a block from Q_k is uploaded to temp_k, for all $k = 1, \ldots, q$. If Q_k is empty, a dummy block is uploaded instead.

After all s rounds complete, each temp_k contains exactly s block ciphertexts. Furthermore, RSpray guarantees that if $\sigma(i) \in \mathsf{destInd}_k$, then an encryption of B_i appears in either temp_k or Q_k. RSpray performs another q *adjustment rounds* to move all client-stored blocks in Q_k to the server-stored temp_k. In the k-th adjustment round, RSpray downloads all s blocks of temp_k. All dummy blocks of temp_k are discarded. The non-dummy blocks of temp_k are decrypted and re-encrypted. Now, all blocks of Q_k are combined with the non-dummy downloaded blocks of temp_k. If more than s non-dummy blocks remain, then RSpray fails. If there are less than s non-dummy blocks, dummy blocks are added until exactly s blocks remain. Finally, all s blocks are uploaded back to temp_k. RSpray terminates upon completion of the s rounds.

4.2 CacheShuffle Description

We now describe CacheShuffle, which uses RSpray and Spray as subroutines. CacheShuffle starts by running the Spray algorithm of CacheShuffleRoot with parameters $s := N/S$ and $q := (1 + \epsilon)S$. Under these parameters, Spray uses $O(S)$ client memory and outputs the following:

1. q caches $\mathsf{Q}_1, \ldots, \mathsf{Q}_q$ on the client;
2. q temporary arrays $\mathsf{temp}_1, \ldots, \mathsf{temp}_q$ on the server;
3. q destination buckets $\mathsf{destInd}_1, \ldots, \mathsf{destInd}_q$ on the client such that if $\sigma(i) \in \mathsf{destInd}_k$ then Q_k or temp_k contain an encryption of B_i;

For $j = 1, \ldots, q$, we perform the *adjustment round* of RSpray for each pair of $(\mathsf{Q}_k, \mathsf{temp}_k)$ outputted by Spray. In particular, all blocks of Q_k are placed into temp_k. After all q adjustment rounds, we have the property that if $\sigma(i) \in \mathsf{destInd}_k$, then temp_k contains an encryption of B_i.

Next, CacheShuffle recursively calls RSpray on each $\mathsf{temp}_1, \ldots, \mathsf{temp}_q$ exactly $l = O(\log_S N)$ times. Formally, each application of RSpray will produce q pairs of temporary arrays and destination buckets. RSpray will be applied to each of the q pairs independently. After l recursive applications of RSpray, we will receive q^l temporary arrays and destination buckets, $(\mathsf{temp}_{l,1}, \mathsf{destInd}_{l,1}), \ldots, (\mathsf{temp}_{l,q^l}, \mathsf{destInd}_{l,q^l})$. Furthermore, all temporary arrays and destination buckets will contain less than S^2 elements. Each $\mathsf{temp}_{l,k}$ is obliviously shuffled into $\mathsf{Dest}[\mathsf{destInd}_{l,k}]$ using CacheShuffleRoot, for all $k = 1, \ldots, q^l$. CacheShuffleRoot may be applied since the client has at least S blocks of client memory.

The proof of the following theorem can be found in the full version.

▶ **Theorem 4.** CacheShuffle *is an Oblivious Shuffle algorithm that uses $O(N \log_S N)$ blocks of bandwidth, $O(N)$ blocks of server memory and $O(S)$ blocks of client memory except with probability negligible in N.*

5 K-Oblivious Shuffling with $O(K)$ Client Memory

In this section, we present an K-oblivious shuffling algorithm, KCacheShuffleBasic, for clients with at least K blocks of client memory. The algorithm uses $2N$ blocks of bandwidth to shuffle N data blocks We remind the reader that the client of a K-oblivious shuffling receives

two permutations (π, σ) as well as a subset of indices $\mathsf{Revealed} \subseteq [N]$. Identical to Oblivious Shuffling, KCacheShuffleBasic also receives ciphertexts of the N blocks in Source according to π and must place the N blocks in Dest according to σ.

KCacheShuffleBasic starts by downloading the ciphertexts from locations $\mathsf{Source}[\pi(\mathsf{Revealed})]$. That is, the encryption of every block whose index belongs to Revealed is downloaded. Each block is decrypted and re-encrypted. Then, KCacheShuffleBasic initializes $\mathsf{tbDown} = [N] \setminus \mathsf{Revealed}$, which is the set of indices of blocks that have not been downloaded yet.

Then, KCacheShuffleBasic runs N rounds. The goal of the i-th round is to place a ciphertext of $B_{\sigma^{-1}(i)}$ into $\mathsf{Dest}[i]$. If $\sigma^{-1}(i) \in \mathsf{tbDown}$, and thus the ciphertext of $B_{\sigma^{-1}(i)}$ has not been downloaded yet, KCacheShuffleBasic simply downloads $B_{\sigma^{-1}(i)}$ from $\mathsf{Source}[\pi(\sigma^{-1}(i))]$ and removes $\sigma^{-1}(i)$ from tbDown. If $\sigma^{-1}(i) \notin \mathsf{tbDown}$, then $B_{\sigma^{-1}(i)}$ has already been downloaded from $\mathsf{Source}[\pi(\sigma^{-1}(i))]$. If $\mathsf{tbDown} \neq \emptyset$, any arbitrary index $i' \in \mathsf{tbDown}$ is removed and $B_{i'}$ is downloaded from $\mathsf{Source}[\pi(i')]$. In any of the above cases, the downloaded block is decrypted and re-encrypted. Finally, $B_{\sigma^{-1}(i)}$ is placed into $\mathsf{Dest}[i]$.

From first look, it seems that KCacheShuffleBasic requires $O(N)$ roundtrips of client-server communication. However, the number of roundtrips may be reduced by grouping indexes of Dest together. Specifically, we can group indexes of Dest into groups of size $O(K)$ and perform the required downloads and uploads in $O(N/K)$ roundtrips.

The proof of the following theorem can be found in the full version.

▶ **Theorem 5.** KCacheShuffleBasic *is a K-Oblivious Shuffle that uses $2N$ blocks of bandwidth, $2N$ blocks of server storage and $O(K)$ blocks of client storage.*

6 K-Oblivious Shuffling with Smaller Client Memory

In this section, for any constant $\epsilon > 0$, we describe $\mathsf{KCacheShuffle}_{\epsilon,S}$, a K-oblivious shuffling that uses $O(S)$ blocks of client memory. For convenience, we fix ϵ and S and refer to $\mathsf{KCacheShuffle}_{\epsilon,S}$ as simply KCacheShuffle.

KCacheShuffle will invoke a modification of CacheShuffle (using the same value of ϵ). Recall that CacheShuffle invokes CacheShuffleRoot before completion. KCacheShuffle invokes CacheShuffle such that the last Recalibrate phase of CacheShuffleRoot is skipped. Note, only the last Recalibrate phase of CacheShuffleRoot actually places ciphertexts of blocks into the destination array Dest. Therefore, the modified CacheShuffle does not actually use Dest at all. Formally, CacheShuffle invokes KCacheShuffle using Source[Revealed] as the source array and (π, σ) restricted to the input set Revealed as the input permutations. The output of the modified CacheShuffle is:

1. q client-stored *destination buckets* $\mathsf{destInd}_1, \ldots, \mathsf{destInd}_q$, which is a partition of Revealed;
2. q server-stored *temporary arrays* $\mathsf{temp}_1, \ldots, \mathsf{temp}_q$ containing exactly S block ciphertexts such that if $i \in \mathsf{Revealed}$ and $\sigma(i) \in \mathsf{destInd}_k$ then temp_k contains an encryption of B_i;

In the next step, KCacheShuffle merges the revealed and unrevealed blocks. KCacheShuffle will extend $\mathsf{destInd}_1, \ldots, \mathsf{destInd}_q$ from a partition of Revealed to a partition of $[N]$. That is, each index of $[N] \setminus \sigma(\mathsf{Revealed})$ is assigned uniformly and independent at random to one of $\mathsf{destInd}_1, \ldots, \mathsf{destInd}_q$. KCacheShuffle initializes $\mathsf{tbDown}_k = \sigma^{-1}(\mathsf{destInd}_k \setminus \sigma(\mathsf{Revealed}))$, which is all unrevealed blocks assigned to $\mathsf{Dest}[\mathsf{destInd}_k]$ by σ, for all $k = 1, \ldots, q$. Then, KCacheShuffle will run up to q rounds, one for each of $\mathsf{destInd}_1, \ldots, \mathsf{destInd}_q$. In the k-th round, KCacheShuffle downloads all S ciphertexts of temp_k.

We quickly diverge by mentioning an obvious, but incorrect, next step. In particular, it might be tempting to just download all of $\mathsf{Source}[\pi(\mathsf{tbDown}_k)]$ to upload into $\mathsf{Dest}[\mathsf{destInd}_k]$.

However, this step would reveal to the adversary the number of revealed (as well as unrevealed blocks) that are placed into $\mathsf{Dest}[\mathsf{destInd}_k]$ according to σ. This possible next step would cause our algorithm to lose K-obliviousness. In particular, our algorithm cannot leak knowledge about the number of revealed (and unrevealed) blocks that belong in the set $\mathsf{Dest}[\mathsf{destInd}_k]$.

Instead, KCacheShuffle downloads exactly $u_k := |\mathsf{destInd}_k| - (1-\epsilon)K/q$ unrevealed blocks in the k-th round. If $|\mathsf{tbDown}_k| > u_k$, then KCacheShuffle will fail and abort (in Theorem 6 we shall prove that this happens with negligible probability). On the other hand, suppose that $|\mathsf{tbDown}_k| \leq u_k$. In this case, the algorithm will download all blocks in $\mathsf{Source}[\pi(\mathsf{tbDown}_k)]$ and if more are needed, that is if $|\mathsf{tbDown}_k| < u_k$, then extra unrevealed blocks that are not assigned to $\mathsf{Source}[\mathsf{destInd}_k]$ are downloaded from the destination array with the largest index that has not been used yet. Formally, KCacheShuffle initializes $\mathsf{leftover} = q$ before any of the q rounds begins. The $|\mathsf{tbDown}_k| - u_k$ extra blocks are chosen arbitrarily from the set $\mathsf{Source}[\pi(\mathsf{tbDown}_{\mathsf{leftover}})]$, which have not been downloaded yet. If all blocks of $\mathsf{Source}[\pi(\mathsf{tbDown}_{\mathsf{leftover}})]$ have been been downloaded, then leftover is decremented.

All u_k blocks downloaded by KCacheShuffle in the k-th round will be decrypted and re-encrypted. Then, KCacheShuffle will upload all blocks belonging to $\mathsf{Dest}[\mathsf{destInd}_k]$ in any arbitrary order such as increasing in $\mathsf{destInd}_k$. Furthermore, KCacheShuffle has a set of extra unrevealed blocks, which we denote Rem_k, that are not assigned to $\mathsf{Dest}[\mathsf{destInd}_k]$ by σ. If $|\mathsf{Rem}_k| > 2\epsilon K/q$, then KCacheShuffle fails and aborts Otherwise, KCacheShuffle pads Rem_k to contain exactly $2\epsilon K/q$ ciphertexts by adding encryptions of dummy blocks. Afterwards, Rem_k is uploaded to the server.

At some point, leftover and k will be equal. Once leftover and k are the same, KCacheShuffle will stop running these rounds. However, KCacheShuffle still has to place blocks into $\mathsf{Dest}[\mathsf{destInd}_{\mathsf{leftover}}], \ldots, \mathsf{Dest}[\mathsf{destInd}_q]$ from $\mathsf{temp}_{\mathsf{leftover}}, \ldots, \mathsf{temp}_q$ and $\mathsf{Rem}_1, \ldots, \mathsf{Rem}_{\mathsf{leftover}-1}$. To achieve this, KCacheShuffle invokes CacheShuffle using $\mathsf{temp}_{\mathsf{leftover}} \cup \ldots \cup \mathsf{temp}_q \cup \mathsf{Rem}_1 \cup \ldots \mathsf{Rem}_{\mathsf{leftover}-1}$ as the source array, $\mathsf{Dest}[\mathsf{destInd}_{\mathsf{leftover}} \cup \ldots \cup \mathsf{destInd}_q]$ as the destination array and (π, σ) restricted to $\sigma^{-1}(\mathsf{destInd}_{\mathsf{leftover}}) \cup \ldots \cup \sigma^{-1}(\mathsf{destInd}_q)$ as the permutations. In the full version, we show that $\mathsf{destInd}_{\mathsf{leftover}} \cup \ldots \cup \mathsf{destInd}_q$ contains $O(K)$ indices.

If the client has $O(\sqrt{K})$ blocks of client storage, then we may replace CacheShuffle with CacheShuffleRoot above. We refer to this construction as KCacheShuffleRoot.

The proof of the following theorem can be found in the full version.

▶ **Theorem 6.** *For every $S = \omega(\log N)$ and for every ϵ, $\mathsf{KCacheShuffle}_{\epsilon,S}$ is a K-oblivious shuffling algorithm that uses $2N + O(K \log_S K)$ blocks of bandwidth, $O(N)$ blocks of server storage, $O(S)$ blocks of client storage and fails with probability negligible in N.*

7 Conclusions

In this paper, we studied the notion of *oblivious* algorithms for the problem of shuffling data. We introduce the notion of K-oblivious shuffling, a fine-grained notion of obliviousness, which accurately describes the adversary's knowledge about the initial position of the N blocks. In particular, we assume the adversary gains information about the initial position of exactly K blocks. This notion has direct application to the design of Oblivious RAMs. Previous notions only considered the extreme case where the adversary has complete knowledge about the initial positioning of all N input blocks (that is, $K = N$).

We present bandwidth efficient moved-based K-oblivious shuffling algorithms for any K and for any client with $S = \omega(\log N)$ blocks of available storage. The bandwidth required by our algorithms is of the form $2N + f(K, S)$. We also look at the previous considered the case of $K = N$ corresponding to previous oblivious shuffling notions. We present an oblivious

shuffle using essentially $4N$ blocks of bandwidth. The previous, state-of-the-art oblivious shuffle [10] required approximately $18N$ blocks of bandwidth for similar failure probabilities.

We also consider the case where we shuffle N *real* blocks along with D *dummy* blocks. In this case, the contents of dummy blocks are irrelevant. By utilizing server-side computation, we shuffle using essentially $D + (2 + \epsilon)N$ blocks of bandwidth. Thus, our algorithm uses less bandwidth than any move-based algorithm uses at least $2(N + D)$ blocks of bandwidth.

References

1. M. Ajtai, J. Komlós, and E. Szemerédi. An $O(n \log n)$ sorting network. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 1–9. ACM, 1983.
2. K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 307–314. ACM, 1968.
3. Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnés, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *SOSP*, pages 441–459. ACM, 2017.
4. O. Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 182–194, New York, NY, USA, 1987. ACM.
5. Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.
6. Michael T. Goodrich. Randomized Shellsort: A simple data-oblivious sorting algorithm. *J. ACM*, 58(6):27:1–27:26, 2011. doi:10.1145/2049697.2049701.
7. Michael T. Goodrich. Zig-Zag sort: A simple deterministic data-oblivious sorting algorithm running in $O(n \log n)$ time. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 684–693, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591830.
8. Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu-An Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Inf. Sci.*, 265:176–188, 2014.
9. Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In *CCS '15*, pages 644–655. ACM, 2015.
10. Olga Ohrimenko, Michael T. Goodrich, Roberto Tamassia, and Eli Upfal. The Melbourne shuffle: Improving oblivious storage in the cloud. In *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 556–567, 2014.
11. Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. CacheShuffle: A family of oblivious shuffles. *CoRR*, abs/1705.07069, 2017. arXiv:1705.07069.
12. Emil Stefanov, Elaine Shi, and Dawn Song. Towards practical oblivious ram. *arXiv preprint arXiv:1106.3652*, 2011.
13. Abraham Waksman. A permutation network. *J. ACM*, 15(1):159–163, 1968. doi:10.1145/321439.321449.

Brief Announcement: MapReduce Algorithms for Massive Trees

MohammadHossein Bateni
Google Research, New York

Soheil Behnezhad
University of Maryland

Mahsa Derakhshan
University of Maryland

MohammadTaghi Hajiaghayi
University of Maryland

Vahab Mirrokni
Google Research, New York

Abstract

Solving large-scale graph problems is a fundamental task in many real-world applications, and it is an increasingly important problem in data analysis. Despite the large effort in designing scalable graph algorithms, many classic graph problems lack algorithms that require only a sublinear number of machines and space in the input size. Specifically when the input graph is large and sparse, which is indeed the case for many real-world graphs, it becomes impossible to store and access all the vertices in one machine – something that is often taken for granted in designing algorithms for massive graphs. The theoretical model that we consider is the *Massively Parallel Communications* (MPC) model which is a popular theoretical model of MapReduce-like systems. In this paper, we give an algorithmic framework to adapt a large family of dynamic programs on MPC. We start by introducing two classes of dynamic programming problems, namely "(poly log)-expressible" and "linear-expressible" problems. We show that both classes can be solved efficiently using a sublinear number of machines and a sublinear memory per machine. To achieve this result, we introduce a series of techniques that can be plugged together. To illustrate the generality of our framework, we implement in $O(\log n)$ rounds of MPC, the dynamic programming solution of fundamental problems such as minimum bisection, k-spanning tree, maximum independent set, longest path, etc., when the input graph is a tree.

2012 ACM Subject Classification Theory of computation → MapReduce algorithms, Theory of computation → Massively parallel algorithms, Theory of computation → Dynamic programming

Keywords and phrases MapReduce, Trees

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.162

Acknowledgements Supported in part by NSF CAREER award CCF-1053605, NSF BIGDATA grant IIS-1546108, NSF AF:Medium grant CCF-1161365, DARPA GRAPHS/AFOSR grant FA9550-12-1-0423, and another DARPA SIMPLEX grant.

1 Introduction

With the inevitable growth of the size of datasets to analyze, the rapid advance of distributed computing infrastructure and platforms (such as MapReduce, Spark [11], Hadoop [10], Flume [6], etc.), the need for developing better distributed algorithms is felt far and wide

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 162; pp. 162:1–162:4

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

nowadays. The past decade has seen a lot of progress in studying important computer science problems in the large-scale setting, which led to either adapting the sequential algorithms to modern parallel settings or at times designing from scratch algorithms for these problems [1, 2, 4, 7, 5]. Despite this trend, we still have limited theoretical understanding of the status of several fundamental problems when it comes to designing large-scale algorithms. In fact, even simple and widely used techniques such as the greedy approach or dynamic programming seem to suffer from an inherent sequentiality that makes them difficult to adapt in parallel or distributed settings on the aforementioned platforms. Finding methods to run generic greedy algorithms or dynamic programming algorithms on MapReduce, for instance, has broad applications. This is the main goal of this paper.

We consider the Massively Parallel Communication ($\mathcal{MPC}$) model among previously studied MapReduce-like models [9, 8, 3]. Let n denote the input size and let m denote the number of available machines which is given in the input. At each round, every machine can use a space of size $s = \tilde{\mathrm{O}}(n/m)$ and run an algorithm that is preferably linear time (but at most polynomial time) in the size of its memory. Machines may only communicate between the rounds, and no machine can receive or send more data than its memory.

In this paper, we give an algorithmic framework that could be used to simulate many natural dynamic programs on trees. Indeed we formulate the properties that make a dynamic program (on trees) amenable to our techniques. These properties, we show, are natural and are satisfied by many known algorithms for fundamental optimization problems. To illustrate the generality of our framework, we design $\mathrm{O}(\log n)$ round algorithms for well-studied graph problems on trees, such as, minimum bisection, minimum k-spanning tree, maximum weighted matching, etc.

2 Main Results

We introduce a class of dynamic programming problems which we call f-expressible problems. Here, f is a function and we get classes such as (poly log)-expressible problems or linear-expressible problems. Roughly speaking, f is proportional to the amount of data that each node of the tree stores in the dynamic program. Thus, linear-expressible problems are generally harder to solve than (poly log)-expressible problems.

(poly log)-Expressible Problems. Many natural problems can be shown to be (poly log)-expressible. For example, the following graph problems are all (poly log)-expressible if defined on trees: maximum (weighted) matching, vertex cover, maximum independent set, dominating set, longest path, etc. Intuitively, the dynamic programming solution of each of these problems, for any vertex v, computes at most a constant number of values. Our first result is to show that every (poly log)-expressible problem can be efficiently solved in $\mathcal{MPC}$. As a corollary of that, all the aforementioned problems can be solved efficiently on trees using the optimal total space of $\tilde{\mathrm{O}}(n)$.

▶ **Theorem 1.** *For any given m, there exists an algorithm that w.h.p. solves any* (poly log)*-expressible problem in* $\mathrm{O}(\log n)$ *rounds of $\mathcal{MPC}$ using m machines while each machine uses a space of size at most* $\tilde{\mathrm{O}}(n/m)$.

Proof sketch. The first problem in solving dynamic programs on trees is that there is no guarantee on the depth of the tree. If the given tree has only logarithmic depth one can obtain a logarithmic round algorithm by simulating a bottom-up dynamic program in parallel, where nodes at the same level are handled in the same round simultaneously. This is

reminiscent of certain parallel algorithms whose number of rounds depends on the diameter of the graph. Unfortunately the input tree might be quite unbalanced, with superlogarithmic depth. An extreme case is a path of length n. In this case we can partition the path into equal pieces (despite not knowing a priori the depth of each particular node), handling each piece independently, and then stitching the results together. Complications arise, because the subproblems are not completely independent. Things become more nuanced when the input tree is not simply a path. To resolve this issue, we adopt a celebrated *tree contraction* method of parallel computing to our model. Roughly speaking, the algorithm decomposes the tree into pieces of size at most $\tilde{O}(m)$ (i.e., we can fit each component completely on one machine), with small interdependence. Omitting minor technical details, the latter property allows us to almost independently solve the subproblems on different machines. This results in a partial solution that is significantly smaller than the whole subtree; therefore, we can send all these partial solutions to a master machine in the next round and merge them.

Linear-Expressible Problems. Although many natural problems are indeed (poly log)-expressible, there are instances that are not. Consider for example the *minimum bisection problem*. In the natural dynamic programming solution of this problem, for a subtree T of size n_T, we store $O(n_T)$ different values. That is, for any $i \in [n_T]$, the dynamic program stores the weight of the optimal coloring that assigns blue to i vertices and red to the rest of $n_T - i$ vertices. This problem is not necessarily (poly log)-expressible unless we find another problem specific dynamic programming solution for it. However, it can be shown that minimum bisection, as well as many other natural problems, including k-spanning-tree, k-center, k-median, etc., are linear-expressible. It is notoriously more difficult to solve linear-expressible problems. However, using a slightly more total memory, we show that it is still possible to obtain the same result.

▶ **Theorem 2** (Main Result). *For any given m, there exists an algorithm that w.h.p. solves any linear-expressible problem that is splittable in $O(\log n)$ rounds of $\mathcal{MPC}$ using m machines that each uses a space of size $\tilde{O}(n^{4/3}/m)$*

Proof sketch. Recall that linear-expressibility implies that the dynamic programming data on each node, can be as large as the size of its subtree (i.e., even up to $O(n)$). Therefore, even by using the tree decomposition technique, the partial solution that is computed for each component of the tree can be linear in its size. This means that the idea of sending all these partial data to one master machine, which worked for (poly log)-expressible problems, does not work here since when aggregated, they take as much space as the original input. Therefore we have to distribute the merging step among the machines.

Assume for now that each component that is obtained by the tree decomposition algorithm is contracted into a node and call this *contracted tree*. The tree decomposition algorithm has no guarantee on the depth of the contracted tree and it can be super-logarithmic; therefore a simple bottom-up merging procedure does not work. However, it is guaranteed that the contracted tree itself (i.e., when the components are contracted) can be stored in one machine. Using this, we send the contracted tree to a machine and design a *merging schedule* that informs each component about the round at which it has to be merged with each of its neighbours. The merging schedule ensures that after $O(\log n)$ phases, all the components are merged together. The merging schedule also guarantees that the number of neighbours of the components, after any number of merging phases, remains constant. This is essential to allow (almost) independent merging for many linear-expressible problems such as minimum bisection. Observe that after a few rounds, the merged components grow to have up to

$\Omega(n)$ nodes and even the partial data of one component cannot be stored in one machine. Therefore, even merging the partial data of two components has to be distributed among the machines. For this to be possible, we use a *splitting* technique of independent interest that requires a further *splittability* property on linear-expressible problems. Indeed we show that the aforementioned linear-expressible problems have the splittability property, and therefore Theorem 2 implies they can also be solved in $O(\log n)$ rounds of $\mathcal{MPC}$.

References

1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 459–467, 2012. URL: http://portal.acm.org/citation.cfm?id=2095156&CFID=63838676&CFTOKEN=79617016.

2 Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 574–583. ACM, 2014.

3 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 273–284. ACM, 2013.

4 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1326–1344, 2016. doi:10.1137/1.9781611974331.ch92.

5 Rajesh Hemant Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1234–1251, 2015. doi:10.1137/1.9781611973730.82.

6 Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

7 Hossein Esfandiari, Mohammad Taghi Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1217–1233, 2015. doi:10.1137/1.9781611973730.81.

8 Michael T Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *International Symposium on Algorithms and Computation*, pages 374–383. Springer, 2011.

9 Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 938–948. Society for Industrial and Applied Mathematics, 2010.

10 Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 2012.

11 Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, pages 10–10, 2010. URL: http://dl.acm.org/citation.cfm?id=1863103.1863113.

Brief Announcement: Give Me Some Slack: Efficient Network Measurements

Ran Ben Basat
Technion, Haifa, Israel
sran@cs.technion.ac.il

Gil Einziger
Nokia Bell Labs, Kfar Saba, Israel
gil.einziger@nokia.com

Roy Friedman
Technion, Haifa, Israel
roy@cs.technion.ac.il

Abstract

Many networking applications require timely access to recent network measurements, which can be captured using a sliding window model. Maintaining such measurements is a challenging task due to the fast line speed and scarcity of fast memory in routers. In this work, we study the impact of allowing *slack* in the window size on the asymptotic requirements of sliding window problems. That is, the algorithm can dynamically adjust the window size between W and $W(1+\tau)$ where τ is a small positive parameter. We demonstrate this model's attractiveness by showing that it enables efficient algorithms to problems such as Maximum and General-Summing that require $\Omega(W)$ bits even for constant factor approximations in the exact sliding window model. Additionally, for problems that admit sub-linear approximation algorithms such as Basic-Summing and Count-Distinct, the slack model enables a further asymptotic improvement.

The main focus of our paper [4] is on the widely studied Basic-Summing problem of computing the sum of the last W integers from $\{0, 1 \ldots, R\}$ in a stream. While it is known that $\Omega(W \log R)$ bits are needed in the exact window model, we show that approximate windows allow an *exponential* space reduction for constant τ.

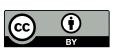
Specifically, for $\tau = \Theta(1)$, we present a space lower bound of $\Omega(\log(RW))$ bits. Additionally, we show an $\Omega(\log(W/\epsilon))$ lower bound for $RW\epsilon$ additive approximations and a $\Omega(\log(W/\epsilon) + \log\log R)$ bits lower bound for $(1+\epsilon)$ multiplicative approximations. Our work is the first to study this problem in the exact and additive approximation settings. For all settings, we provide memory optimal algorithms that operate in *worst case* constant time. This strictly improves on the work of [12] for $(1+\epsilon)$-multiplicative approximation that requires $O(\epsilon^{-1}\log(RW)\log\log(RW))$ space and performs updates in $O(\log(RW))$ worst case time. Finally, we show asymptotic improvements for the Count-Distinct, General-Summing and Maximum problems.

2012 ACM Subject Classification Networks → Network measurement

Keywords and phrases Streaming, Algorithms, Sliding window, Lower bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.163

Related Version A full version of the paper is available at [4], https://arxiv.org/abs/1703.01166.



45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 163; pp. 163:1–163:5

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

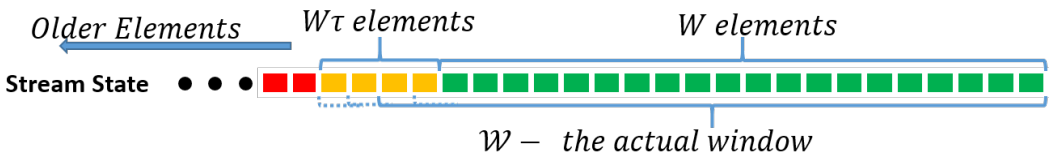


Figure 1 We need to answer each query with respect to a τ-slack window that must include the last W items, but may or may not consider a suffix of the previous $W\tau$ elements.

1 Introduction

Network algorithms in diverse areas such as traffic engineering, load balancing and quality of service [1, 8, 17, 20, 27] rely on timely link measurements. In such applications recent data is often more relevant than older data, motivating the notions of *aging* and *sliding window* [5, 10, 13, 21, 23]. For example, a sudden decrease in the average packet size on a link may indicate a SYN attack [22]. Additionally, a load balancer may benefit from knowing the current utilization of a link to avoid congestion [1].

While conceptually simple, conveying the necessary information to network algorithms is a difficult challenge due to current memory technology limitations. Specifically, DRAM memory is abundant but too slow to cope with the line rate while SRAM memory is fast enough but has a limited capacity [9, 11, 25]. Online decisions are therefore realized through space efficient data structures [6, 7, 14, 15, 3, 19, 24, 26] that store measurement statistics in a concise manner. For example, [14, 24] utilize probabilistic counters that only require $O(\log\log N)$ bits to approximately represent numbers up to N. Others conserve space using variable sized counter encoding [15, 19] and monitoring only the frequent elements [5].

Basic-Summing is one of the most basic textbook examples of such approximated sliding window stream processing problems [12]. In this problem, one is required to keep track of the sum of the last W elements, when all elements are non-negative integers in the range $\{0, 1, \ldots, R\}$. The work in [12] provides a $(1+\epsilon)$-multiplicative approximation of this problem using $O\left(\frac{1}{\epsilon}\cdot\left(\log^2 W + \log R\cdot(\log W + \log\log R)\right)\right)$ bits. The amortized time complexity is $O(\frac{\log R}{\log W})$ and the worst case is $O(\log W + \log R)$. In contrast, we previously showed an $RW\epsilon$-additive approximation with $\Theta\left(\frac{1}{\epsilon} + \log W\epsilon\right)$ bits [2].

Sliding window counters (approximated or accurate) require asymptotically more space than plain stream counters. Such window counters are prohibitively large for networking devices which already optimize the space consumption of plain counters.

This paper explores the concept of *slack*, or approximated sliding window, bridging this gap. Figure 1 illustrates a "window" in this model. Here, each query may select a *τ-slack window* whose size is between W (the green elements) and $W(1+\tau)$ (the green plus yellow elements). The goal is to compute the sum with respect to this chosen window.

Slack windows were also considered in previous works [12, 23] and we call the problem of maintaining the sum over a slack window Slack Summing. Datar et al. [12] showed that constant slack reduces the required memory from $O(\frac{1}{\epsilon}\cdot\left(\log^2 W + \log R\cdot(\log W + \log\log R)\right))$ to $O(\epsilon^{-1}\log(RW)\log\log(RW))$. For τ-slack windows they provide a $(1+\epsilon)$-multiplicative approximation using $O(\epsilon^{-1}\log(RW)(\log\log(RW) + \log\tau^{-1}))$ bits.

Our Contributions

Our paper [4] studies the space and time complexity reductions that can be attained by allowing *slack* – an error in the window size. Our results demonstrate exponentially smaller and asymptotically faster data structures compared to various problems over exact windows.

Table 1 Comparison of Basic-Summing algorithms. Our contributions are in bold. All algorithms process elements in constant time except for the rightmost column where both update in $O(\log(RW))$ time. We present matching lower bounds to all our algorithms.

	Exact Sum	Additive Error	Multiplicative Error	
$\tau = \Theta(1)$	$\mathbf{\Theta(\log(RW))}$	$\mathbf{\Theta(\log(W/\epsilon))}$	$\mathbf{\Theta(\log(W/\epsilon) + \log\log R)}$	$O(\epsilon^{-1}\log(RW)\log\log(RW))$ [12]
Exact Window	$\Theta(W \log R)$	$\Theta(\epsilon^{-1} + \log W)$ [2]	$O(\epsilon^{-1}\log^2(RW))$ [18]	$O(\epsilon^{-1}\log RW \log(W \log R))$ [12]

We start with deriving lower bounds for three variants of the Basic-Summing problem – when computing an exact sum over a slack window, or when combined with an additive and a multiplicative error in the sum. We present algorithms that are based on dividing the stream into $W\tau$-sized blocks. Our algorithms sum the elements within each block and represent each block's sum in a cyclic array of size τ^{-1}. We use multiple compression techniques during different stages to drive down the space complexity. The resulting algorithms are space optimal, substantially simpler than previous work, and reduce update time to $O(1)$.

For exact Slack Summing, we present a lower bound of $\Omega(\tau^{-1}\log(RW\tau))$ bits. For $(1+\epsilon)$ multiplicative approximations we prove an $\Omega\big(\log(W/\epsilon) + \tau^{-1}(\log(\tau/\epsilon) + \log\log(RW))\big)$ bits bound when $\tau = \Omega\left(\frac{1}{\log RW}\right)$. We show that $\Omega(\tau^{-1}\log\lfloor 1+\tau/\epsilon\rfloor + \log(W/\epsilon))$ bits are required for $RW\epsilon$ additive approximations.

Next, we introduce algorithms for the Slack Summing problem, which asymptotically reduce the required memory compared to the sliding window model. For the exact and additive error versions of the problem, we provide memory optimal algorithms. In the multiplicative error setting, we provide an $O\big(\tau^{-1}\left(\log\epsilon^{-1} + \log\log(RW\tau)\right) + \log(RW)\big)$ space algorithm. This is asymptotically optimal when $\tau = \Omega(\log^{-1} W)$ and $R = \text{poly}(W)$. It also asymptotically improves [12] when $\tau^{-1} = o(\epsilon^{-1}\log(RW))$. We further provide an asymptotically optimal solution for constant τ, even when $R = W^{\omega(1)}$. All our algorithms are deterministic and operate in worst case constant time. In contrast, the algorithm of [12] works in $O(\log RW)$ worst case time.

To exemplify our results, consider monitoring the average bandwidth (in bytes per second) passed through a router in a 24 hours window, i.e., $W \triangleq 86400$ seconds. Assuming we use a 100GbE fiber transceiver, our stream values are bounded by $R \approx 2^{34}$ bytes. If we are willing to withstand an error of $\epsilon = 2^{-20}$ (i.e., about $16KBps$), the work of [2] provides an additive approximation over the sliding window and requires about 120KB. In contrast, using a 10 minutes slack ($\tau \triangleq \frac{1}{144}$), our algorithm for **exact** Slack Summing requires only 800 bytes, 99% less than approximate summing over exact sliding window. For the same slack size, the algorithm of [12] requires more space than our **exact** algorithm even for a large 3% error. Further, if we also allow the same additive error ($\epsilon = 2^{-20}$), we provide an algorithm that requires only 240 bytes - a reduction of more than 99.8% !

Table 1 compares our results for the important case of constant slack with [12]. As depicted, our *exact* algorithm is faster and more space efficient than the multiplicative approximation of [12]. Comparing our multiplicative approximation algorithm to that of [12], we present *exponential* space reductions in the dependencies on ϵ^{-1} and R, with an asymptotic reduction in W as well. We also improve the update time from $O(\log(RW))$ to $O(1)$.

Finally, we apply the slack window approach to multiple streaming problems, including Maximum, General-Summing, Count-Distinct and Standard-Deviation. We show that, while some of these problems cannot be approximated on an exact window in sub-linear space (e.g. Maximum and General-Summing), we can easily do so for slack windows. In the count distinct problem, a constant slack yields an asymptotic space reduction over [10, 16]. The full version of our paper appears in [4].

References

1 Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. Conga: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*, ACM SIGCOMM 2014, 2014. doi:10.1145/2619239.2626316.

2 Ran Ben Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Efficient Summing over Sliding Windows. In *SWAT*, 2016.

3 Ran Ben Basat, Gil Einziger, Roy Friedman, Marcelo Caggiani Luizelli, and Erez Waisbard. Constant time updates in hierarchical heavy hitters. In *ACM SIGCOMM*, 2017.

4 R. Ben Basat, G. Einziger, and R. Friedman. Give Me Some Slack: Efficient Network Measurements. *ArXiv e-prints*, 2017. arXiv:1703.01166.

5 Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Heavy hitters in streams and sliding windows. In *IEEE INFOCOM*, 2016.

6 Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Optimal elephant flow detection. In *IEEE INFOCOM*, 2017.

7 Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Randomized admission policy for efficient top-k and frequency estimation. In *IEEE INFOCOM*, 2017.

8 Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: Fine grained traffic engineering for data centers. In *ACM CoNEXT*, 2011.

9 Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. An improved construction for counting bloom filters. In *ESA*, 2006.

10 Y. Chabchoub and G. Hebrail. Sliding hyperloglog: Estimating cardinality in a data stream over a sliding window. In *2010 IEEE ICDM Workshops*, 2010.

11 Min Chen and Shigang Chen. Counter tree: A scalable counter architecture for per-flow traffic measurement. In *IEEE ICNP*, 2015.

12 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal of Computing*, 2002.

13 G. Einziger and R. Friedman. TinyLFU: A highly efficient cache admission policy. In *PDP 2014*. IEEE Computer Society, 2014. doi:10.1109/PDP.2014.34.

14 Gil Einziger, Benny Fellman, and Yaron Kassner. Independent counter estimation buckets. In *IEEE INFOCOM*, 2015.

15 Gil Einziger and Roy Friedman. Counting with TinyTable: Every Bit Counts! In *ICDCN 2016*. ACM, 2016. doi:10.1145/2833312.2833449.

16 Éric Fusy and Frécéric Giroire. Estimating the number of active flows in a data stream over a sliding window. In *ANALCO*, 2007.

17 Pedro Garcia-Teodoro, Jesús E. Díaz-Verdejo, Gabriel Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers and Security*, 2009.

18 Phillip B. Gibbons and Srikanta Tirthapura. Distributed streams algorithms for sliding windows. In *SPAA*, 2002. doi:10.1145/564870.564880.

19 Nan Hua, Bill Lin, Jun (Jim) Xu, and Haiquan (Chuck) Zhao. Brick: A novel exact active statistics counter architecture. In *ACM/IEEE ANCS*, 2008.

20 Abdul Kabbani, Mohammad Alizadeh, Masato Yasuda, Rong Pan, and Balaji Prabhakar. Af-qcn: Approximate fairness with quantized congestion notification for multi-tenanted data centers. In *IEEE HOTI*, 2010.

21 Yang Liu, Wenji Chen, and Yong Guan. Near-optimal approximate membership query over time-decaying windows. In *IEEE INFOCOM*, 2013.

22 B. Mukherjee, L.T. Heberlein, and K.N. Levitt. Network intrusion detection. *Network, IEEE*, 1994.

23 Moni Naor and Eylon Yogev. Sliding bloom filters. In *ISAAC*. Springer, 2013. doi:10.1007/978-3-642-45030-3_48.

24 Erez Tsidon, Iddo Hanniel, and Isaac Keslassy. Estimators also need shared values to grow together. In *IEEE INFOCOM*, 2012.

25 Hao Wang, H. Zhao, Bill Lin, and Jun Xu. Dram-based statistics counter array architecture with performance guarantee. *IEEE/ACM Transactions on Networking*, 2012.

26 Li Yang, Wu Hao, Pan Tian, Dai Huichen, Lu Jianyuan, and Liu Bin. Case: Cache-assisted stretchable estimator for high speed per-flow measurement. In *IEEE INFOCOM*, 2016.

27 L. Ying, R. Srikant, and X. Kang. The power of slightly more than one sample in randomized load balancing. In *IEEE INFOCOM*, 2015.

Brief Announcement: Towards an Abstract Model of User Retention Dynamics

Eli Ben-Sasson
Department of Computer Science, Technion, Haifa, Israel
eli@cs.technion.ac.il
https://orcid.org/0000-0002-0708-0483

Eden Saig
Department of Computer Science, Technion, Haifa, Israel
edens@cs.technion.ac.il
https://orcid.org/0000-0002-0810-2218

Abstract

A theoretical model is suggested for abstracting the interaction between an expert system and its users, with a focus on reputation and incentive compatibility. The model assumes users interact with the system while keeping in mind a single *"retention parameter"* that measures the strength of their belief in its predictive power, and the system's objective is to reinforce and maximize this parameter through "informative" and "correct" predictions.

We define a natural class of *retentive scoring rules* to model the way users update their retention parameter and thus evaluate the experts they interact with. Assuming agents in the model have an incentive to report their true belief, these rules are shown to be tightly connected to truth-eliciting "proper scoring rules" studied in Decision Theory.

The difference between users and experts is modeled by imposing different limits on their predictive abilities, characterized by a parameter called *memory span*. We prove the *monotonicity theorem* ("more knowledge is better"), which shows that experts with larger memory span retain better in expectation.

Finally, we focus on the intrinsic properties of phenomena that are amenable to collaborative discovery with a an expert system. Assuming user types (or "identities") are sampled from a distribution D, the *retention complexity* of D is the minimal initial retention value (or "strength of faith") that a user must have before approaching the expert, in order for the expert to retain that user throughout the collaborative discovery, during which the user "discovers" his true "identity". We then take a first step towards relating retention complexity to other established computational complexity measures by studying retention dynamics when D is a uniform distribution over a linear space.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases information elicitation, proper scoring rules, retention complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.164

Related Version `https://eccc.weizmann.ac.il/report/2017/160/`

Funding This work was supported by the Israeli Science Foundation under grant number 1501/14.

Acknowledgements We thank Yuval Filmus for many helpful discussions.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 164; pp. 164:1–164:4

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Motivation

Aspiring gurus face the problem of attracting new followers, and *retaining* existing ones, as they journey together to a better future. This is an old problem. Moses, for instance, raised it before The Lord before assuming leadership of the Israelite Exodus from Egypt, asking: *"What if they won't believe me or listen to me?" [Exodus 4:1]*. Many gurus resolve the problem by *predicting unlikely events* as a demonstration of their powers; the Biblical Exodus story contains several such events (culminating with the crossing of the Red Sea), all of which were predicted correctly by Moses.

In today's information society, crowd-based automated gurus gather data from users on a voluntary basis in order to produce meaningful insights. The quality of insights greatly depends on the amount and quality of data provided by the users, but those users have limited attention, giving rise to the study of *attention economy* [2, 4], and the design of interactive systems taking limited attention span into account. By asking *"interesting questions"* and making *"meaningful predictions"*, an automated interactive system can retain users, but only if it "knows" how to ask "interesting" questions and provide "meaningful" feedback.

The phenomenon that motivated this research is that of *early child development*; the gurus are experts in this field and the followers are parents of newborn babies [1]. For the sake of concreteness we shall continue using this particular setting to describe our model but it may be conveniently replaced by the reader with physicians or psychologists playing the gurus as they interact with patients (followers) regarding a complex medical or mental problem, or with financial advisors as gurus and their follower clientele. In these and similar settings, gurus and followers discuss a complex phenomenon that evolves over time, which the followers wish to understand, and about which the guru claims to have an advantage of *"wisdom"* over them.

The main goal of this work is the development of a clean mathematical model which mimics the retention dynamics of interactive systems, and can be used to explain why "smarter" gurus tend to retain a larger following. A clean mathematical model often sheds light on the studied phenomena, and may facilitate the future design of more efficient and successful automated gurus.

2 The Collaborative Discovery Model

The *phenomenon* about which the guru and her followers interact is modeled by a *distribution* $\mathcal{T}$ over $\mathcal{X}^\Gamma$, where Γ is the set of properties manifested by the phenomenon and $\mathcal{X}$ is an arbitrary input space. In the context of childhood development, Γ is the set of developmental milestones (like *"first smile"*), and each follower is represented by a sample $u \in \mathcal{X}^\Gamma$ that describes the ages at which that child achieved each milestone.

The guru and follower interact over a number of *rounds*: At the start of each round of interaction, the guru picks an undisclosed property $\gamma_t \in \Gamma$, and makes a prediction by announcing a distribution P_{γ_t} over $\mathcal{X}$ that she claims is the true one for a latent attribute $\gamma_t \notin \Gamma_t$; the follower has a distribution Q_{γ_t} that he believes corresponds to γ_t. After announcing both distributions, the true value $u_{\gamma_t} \in \mathcal{X}$ is revealed.

After each round, the follower updates the strength of his *retention* by the guru. We assume this strength is given by a *retention parameter* r_t that starts with a fixed value r_0 and varies with time; once r_t turns negative, the follower will be said to have lost all faith in the guru and therefore terminate the interaction. The main objective of the guru is to maintain $r_t \geq 0$ for all $t \geq 0$.

3 Main Contributions

Retentive Scoring Rules

The retention parameter r_t mentioned above and the dynamics that form around it are the key ingredients that give the model its expressive strength. Defining the retention parameter update rules, and the surprising corollaries of the chosen definitions, are what dominates the first part of our study.

We assume the retention parameter changes in an additive manner after each round, according to a function $\mathcal{S}(\cdot,\cdot,\cdot)$ that is real-valued and takes three inputs: (i) the guru's predicted distribution P_{γ_t}; (ii) the follower's assessment of that distribution Q_{γ_t}; and (iii) the value u_{γ_t} that materialized, picked by Nature. Analysis is further facilitated by assuming that $\mathcal{S}$ belongs to a class of functions that *elicit* the true beliefs of both guru and follower regarding the distribution of the attribute γ_t. Truth eliciting rules are ones that incentivize rational players to supply the rule with what they believe to be the truth.

Truthfulness under rationality is a powerful assumption which often leads to non-trivial corollaries. We characterize the functions that can act as retentive scoring rules, showing that such rules can take a surprisingly simple form.

Memory Span and Monotonicity

To model the different predictive capacities of gurus and followers, we characterize the forecasting abilities of agents in the Collaborative Discovery model by a parameter called *memory span*.

A variety of psychological studies could be summarized by saying that the human short-term memory has a capacity of about "seven, plus-or-minus two" *chunks*, where each chunk can be roughly defined as a collection of elementary information relating to a single concept [5, 6], where the "information capacity" of a chunk depends on the knowledge of the person being tested.

We model the discrepancy between users and experts by imposing the different limits on their memory spans. In this context, we show that the definition of memory span is *monotone*, verifying that experts with larger memory span retain followers longer in expectation.

Retention Complexity and Linear Codes

A distribution $\mathcal{T}$ for which there exists a guru that, in expectation, manages to retain followers to eternity (or until $t = |\Gamma|$ for finite Γ) will be said to be r_0-*retainable* and the *retention complexity* of $\mathcal{T}$ will be the minimal r_0 such that $\mathcal{T}$ is r_0-retainable.

To initiate the study of the retention complexity of specific distribution, a class of simple-to-understand, but non-trivial distributions, is needed. Inspired by other initial works, like that of Valiant which studied machine learning in the "restricted, but nontrivial context" of boolean functions [7] and that of Goldreich, Goldwasser and Ron that studied property testing in the context of graph properties [3], we begin by studying retention complexity of uniform distributions over linear spaces.

Uniform distributions over linear spaces are such a family and our final object of discussion. While such distributions are far from ones appearing in the "real world", studying them in this context provide convenient tools and insights about the Collaborative Discovery model, and the intuition and techniques we develop here will be generalized as we move to more "applied" settings.

References

1 Ayelet Ben-Sasson, Eli Ben-Sasson, Kayla Jacobs, and Eden Saig. Baby CROINC: An online, crowd-based, expert-curated system for monitoring child development. In *Proceedings of the 11th EAI International Conference on Pervasive Computing Technologies for Healthcare*, PervasiveHealth '17, pages 110–119, New York, NY, USA, 2017. ACM. doi:10.1145/3154862.3154887.

2 Michael H. Goldhaber. The attention economy and the net. *First Monday*, 2(4), 1997. doi:10.5210/fm.v2i4.519.

3 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.

4 Richard A Lanham. *The economics of attention: Style and substance in the age of information*. University of Chicago Press, 2006.

5 George A Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

6 Endel Tulving and Fergus IM Craik. *The Oxford handbook of memory*. Oxford: Oxford University Press, 2000.

7 Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Brief Announcement: Energy Constrained Depth First Search

Shantanu Das[1]
LIS, Aix-Marseille University, University of Toulon, CNRS, Marseille, France
shantanu.das@lif.univ-mrs.fr

Dariusz Dereniowski[2]
Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Gdańsk, Poland
deren@eti.pg.edu.pl
https://orcid.org/0000-0003-4000-4818

Przemysław Uznański
Department of Computer Science, ETH Zürich, Zürich, Switzerland
przemyslaw.uznanski@inf.ethz.ch

Abstract

Depth first search is a natural algorithmic technique for constructing a closed route that visits all vertices of a graph. The length of such route equals, in an edge-weighted tree, twice the total weight of all edges of the tree and this is asymptotically optimal over all exploration strategies. This paper considers a variant of such search strategies where the length of each route is bounded by a positive integer B (e.g. due to limited energy resources of the searcher). The objective is to cover all the edges of a tree T using the minimum number of routes, each starting and ending at the root and each being of length at most B. To this end, we analyze the following natural greedy tree traversal process that is based on decomposing a depth first search traversal into a sequence of limited length routes. Given any arbitrary depth first search traversal R of the tree T, we cover R with routes $R_1, \ldots, R_l$, each of length at most B such that: R_i starts at the root, reaches directly the farthest point of R visited by R_{i-1}, then R_i continues along the path R as far as possible, and finally R_i returns to the root. We call the above algorithm *piecemeal-DFS* and we prove that it achieves the asymptotically minimal number of routes l, regardless of the choice of R. Our analysis also shows that the total length of the traversal (and thus the traversal time) of piecemeal-DFS is asymptotically minimum over all energy-constrained exploration strategies. The fact that R can be chosen arbitrarily means that the exploration strategy can be constructed in an online fashion when the input tree T is not known in advance. Each route R_i can be constructed without any knowledge of the yet unvisited part of T. Surprisingly, our results show that depth first search is efficient for energy constrained exploration of trees, even though it is known that the same does not hold for energy constrained exploration of arbitrary graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis, Theory of computation → Online algorithms

Keywords and phrases DFS traversal, distributed algorithm, graph exploration, piecemeal exploration, online exploration

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.165

Related Version `https://arxiv.org/abs/1709.10146`

[1] Author partially supported by ANR project ANCOR (anr-14-CE36-0002-01)
[2] Author partially supported by National Science Centre (Poland) grant number 2015/17/B/ST6/01887.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 165; pp. 165:1–165:5

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Consider a mobile robot (also called an *agent*) that needs to explore an initially unknown edge-weighted tree, where the weight of each edge is its length. Starting from a single vertex (the root), the robot must traverse all edges of and return to its initial location. Upon visiting a vertex v for the first time, the robot discovers the edges incident to v and can choose one of them to continue the exploration. Provided that the robot can remember the visited vertices and edges, a simple depth first search (DFS) is an efficient algorithm for exploring the tree, achieving the optimal cost of twice the sum of the lengths of edges in the tree. In a more interesting scenario, the robot has a limited source of energy (e.g. a battery) which allows it to traverse a path of length at most B (we say such a robot is *energy constrained*). Naturally, we assume that each vertex of the tree is at distance at most $B/2$ from the root, otherwise the tree cannot be fully explored. In this case, exploration is possible if the robot can recharge its battery whenever it returns back to the initial location. Thus, the exploration is a collection of routes of the robot, each of which starts and ends at the root, and has length at most B. We are interested in minimizing the number of such routes (i.e. the number of times the robot has to recharge) to completely explore the tree.

Related work. There exists extensive literature on graph traversal and exploration. We refer interested reader to works on several models that do not consider any energy limitation for the agents, including results on general graphs [22], trees [9, 17, 18, 19], lower bounds on exploration time [9, 13, 14, 20, 21], or exploration with little memory [1, 12].

The energy constrained exploration problem was first studied under the name of *Piecemeal Graph Exploration* [8], with the assumption that the route length $B \geq 2(1+\beta)r$, where r is the furthest distance from the starting vertex to other vertices, and $0 < \beta < 1$. That paper provided exploration algorithms for a special class of grid graphs with 'rectangular obstacles'. Awerbuch et al. [4] showed that, for general graphs, there exists an energy constrained exploration algorithm with a total cost of $O(m + n^{1+o(1)})$. This has been further improved (by an algorithm that is a combination of DFS and BFS) to $O(m + n\log^2 n)$ in [5]. Finally [15] provided an exploration algorithm for general unknown weighted graphs with total cost asymptotic to the sum of edge weights of the graph. Note that, as mentioned, all the above strategies require the length of each route to be strictly larger than the shortest path from the starting vertex to the farthest vertex. In other words, these algorithms fail in the extreme cases when the height of the explored tree (or the diameter of the graph) is equal to half of the energy budget, which seem to be the most challenging ones. The off-line version of the problem is NP-hard, see e.g. [18].

See [6, 11, 16] for works on the tree exploration model we consider, with one difference: each route, also of length at most B, starts at the root but may end at any vertex of the tree. Distributed algorithms for energy constrained agents have been a subject of recent investigation, see e.g. [2, 3, 7, 10].

2 Problem statement and DFS exploration

Let $T = (V(T), E(T), \omega\colon E(T) \to \mathbb{R}_+)$ be an edge-weighted tree with root r. We define a *route* $R = (v_0, v_1, \ldots, v_l)$ as a sequence of vertices that satisfies: (i) $\{v_i, v_{i+1}\} \in E(T)$ for each $i \in \{0, \ldots, l-1\}$, and (ii) $v_0 = v_l$ is the root r of T. Informally speaking, a route is a sequence of vertices forming a walk in T that starts and ends at the root. We define the *length* of R to be $\ell(R) = \sum_{i=1}^{l} \omega(\{v_{i-1}, v_i\})$. We say that a vertex v is *visited* by the route if $v = v_i$ for some $i \in \{1, \ldots, l\}$.

Given a tree T and a real number B, we say that $\mathcal{S} = (R_1, \ldots, R_k)$ is a *B-exploration strategy* for T if for each $i \in \{1, \ldots, k\}$, R_i is a route in T of length at most B, and each vertex of T is visited by some route in $\mathcal{S}$. We write $|\mathcal{S}|$ to refer to the number of routes in $\mathcal{S}$, $k = |\mathcal{S}|$. We formulate the combinatorial problem we study in this work as follows.

Energy Constrained Tree Exploration problem (ECTE)

Given a real number $B > 1$ and an edge-weighted rooted tree T of height at most $B/2$ what is the minimum integer k such that there exists a B-exploration strategy that consists of k routes?

Our goal is to analyze a particular type of solution to this problem, namely, an exploration strategy that behaves like a depth first search traversal but adopted to the fact that route lengths are bounded by B. Let $R_{\mathrm{DFS}} = (v_0, v_1, \ldots, v_l)$ be a route in T that covers the tree T and performs a depth first search traversal of T. (Note that R_{DFS} is a route and thus we consider a depth first search traversal to have vertex repetitions.) For two vertices u and v of T, $d(u, v)$ denotes the distance between u and v understood as the sum of weights of the edges of the path connecting these vertices. We refer by $\mathrm{PDFS}(T) = (R_1, \ldots, R_k)$ (*Piecemeal Depth First Search*) to the following B-exploration strategy constructed iteratively for $i := 1, \ldots, k$:

- **(i)** let $j_0 = 0$ i.e. $v_{j_0} = v_0 = r$,
- **(ii)** R_i continues DFS exploration from where R_{i-1} *stopped making progress* (from the vertex $v_{j_{i-1}}$) as long as for currently visited v_p: $d(r, v_{j_{i-1}}) + \ell((v_{j_{i-1}}, v_{j_{i-1}+1}, \ldots, v_p)) + d(v_p, r) \leq B$,
- **(iii)** furthest v_p (for $p \leq l$) that satisfies condition from ii is denoted as v_{j_i}, the vertex where R_i stopped making progress,
- **(iv)** let $R_i = P_{i-1} \circ (v_{j_{i-1}}, v_{j_{i-1}+1}, \ldots, v_{j_i-1}, v_{j_i}) \circ P_i^R$, where P_{i-1} is the path from r to $v_{j_{i-1}}$, and P_i^R is the path from v_{j_i} to r.

Such a strategy $\mathrm{PDFS}(T)$ is called a *DFS B-exploration*.

We remark that different depth first search traversals R_{DFS} may result in different values of k (different number of routes) in the resulting DFS B-exploration, although for a particular choice of R_{DFS} the corresponding $\mathrm{PDFS}(T)$ is unique. We point out that our results stated below hold for an *arbitrary* choice of R_{DFS}.

3 Our results

The following theorem provides the first main result of this work.

▶ **Theorem 1.** *Let T be a tree and let the longest path from the root to a leaf in T be at most $B/2$. It holds $|\mathrm{PDFS}(T)| \leq 10\,|\mathcal{R}|$, where $\mathcal{R}$ is a B-exploration strategy that consists of the minimum number of routes.*

The theorem refers to the number of routes in an exploration strategy. However, in order to analyze the behavior of $\mathrm{PDFS}(T)$, we introduce another parameter which turns out to be simpler to analyze. For any B-exploration strategy $\mathcal{S} = (R_1, \ldots, R_k)$ of T we will denote by $\xi(\mathcal{S})$ the *cost* of $\mathcal{S}$ defined as $\xi(\mathcal{S}) = \sum_{i=1}^{k} \ell(R_i)$. Then, $\mathrm{COPT}(T)$ is an optimal solution with respect to the cost, that is, a B-exploration strategy whose cost is minimum over all B-exploration strategies. Thus, in order to prove Theorem 1, we obtain, on route, the following second main result of our work.

▶ **Theorem 2.** *Let T be a tree and let $B/2$ be greater than or equal to the longest path from the root to a leaf in T. It holds $\xi(\mathrm{PDFS}(T)) \leq 10 \cdot \xi(\mathrm{COPT}(T))$.*

References

1 Christoph Ambühl, Leszek Gąsieniec, Andrzej Pelc, Tomasz Radzik, and Xiaohui Zhang. Tree exploration with logarithmic memory. *ACM Trans. Algorithms*, 7(2):17:1–17:21, 2011.

2 Julian Anaya, Jérémie Chalopin, Jurek Czyzowicz, Arnaud Labourel, Andrzej Pelc, and Yann Vaxès. Collecting information by power-aware mobile agents. In *Proceedings of Distributed Computing - 26th International Symposium, DISC 2012*, pages 46–60. Springer, 2012. doi:10.1007/978-3-642-33651-5_4.

3 Julian Anaya, Jérémie Chalopin, Jurek Czyzowicz, Arnaud Labourel, Andrzej Pelc, and Yann Vaxès. Convergecast and broadcast by power-aware mobile agents. *Algorithmica*, 74(1):117–155, 2016.

4 Baruch Awerbuch, Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal graph exploration by a mobile robot. *Inf. Comput.*, 152(2):155–172, 1999.

5 Baruch Awerbuch and Stephen G. Kobourov. Polylogarithmic-overhead piecemeal graph exploration. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory, COLT 1998*, pages 280–286. ACM, 1998. doi:10.1145/279943.279998.

6 Evangelos Bampas, Jérémie Chalopin, Shantanu Das, Jan Hackfeld, and Christina Karousatou. Maximal exploration of trees with energy-constrained agents. *CoRR*, abs/1802.06636, 2018. arXiv:1802.06636.

7 Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Barbara Geissmann, Daniel Graf, Arnaud Labourel, and Matús Mihalák. Collaborative delivery with energy-constrained mobile robots. In *Proceedings of Structural Information and Communication Complexity - 23rd International Colloquium, SIROCCO 2016*, volume 9988 of *Lecture Notes in Computer Science*, pages 258–274, 2016. doi:10.1007/978-3-319-48314-6_17.

8 Margrit Betke, Ronald L. Rivest, and Mona Singh. Piecemeal learning of an unknown environment. *Machine Learning*, 18(2-3):231–254, 1995.

9 Peter Brass, Flavio Cabrera-Mora, Andrea Gasparri, and Jizhong Xiao. Multirobot tree and graph exploration. *IEEE Trans. Robotics*, 27(4):707–717, 2011.

10 Jurek Czyzowicz, Krzysztof Diks, Jean Moussi, and Wojciech Rytter. Communication problems for mobile agents exchanging energy. In *Proceedings of Structural Information and Communication Complexity - 23rd International Colloquium, SIROCCO 2016*, volume 9988 of *Lecture Notes in Computer Science*, pages 275–288, 2016. doi:10.1007/978-3-319-48314-6_18.

11 Shantanu Das, Dariusz Dereniowski, and Christina Karousatou. Collaborative exploration by energy-constrained mobile robots. In *Proceedings of Structural Information and Communication Complexity - 22nd International Colloquium, SIROCCO 2015*, volume 9439 of *Lecture Notes in Computer Science*, pages 357–369. Springer, 2015. doi:10.1007/978-3-319-25258-2_25.

12 Yann Disser, Jan Hackfeld, and Max Klimm. Undirected graph exploration with $\Theta(\log \log N)$ pebbles. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 25–39. SIAM, 2016. doi:10.1137/1.9781611974331.ch3.

13 Yann Disser, Frank Mousset, Andreas Noever, Nemanja Skoric, and Angelika Steger. A general lower bound for collaborative tree exploration. *CoRR*, abs/1610.01753, 2016.

14 Stefan Dobrev, Rastislav Královic, and Euripides Markou. Online graph exploration with advice. In *Proceedings of Structural Information and Communication Complexity - 19th International Colloquium, SIROCCO 2012*, volume 7355 of *Lecture Notes in Computer Science*, pages 267–278. Springer, 2012. doi:10.1007/978-3-642-31104-8_23.

15 Christian A. Duncan, Stephen G. Kobourov, and V. S. Anil Kumar. Optimal constrained graph exploration. *ACM Trans. Algorithms*, 2(3):380–402, 2006.

16 Mirosław Dynia, Mirosław Korzeniowski, and Christian Schindelhauer. Power-aware collective tree exploration. In *Proceedings of Architecture of Computing Systems - ARCS 2006, 19th International Conference*, volume 3894 of *Lecture Notes in Computer Science*, pages 341–351. Springer, 2006. doi:10.1007/11682127_24.

17 Mirosław Dynia, Jarosław Kutyłowski, Friedhelm Meyer auf der Heide, and Christian Schindelhauer. Smart robot teams exploring sparse trees. In *Proceedings of Mathematical Foundations of Computer Science, 31st International Symposium, MFCS 2006*, volume 4162 of *Lecture Notes in Computer Science*, pages 327–338. Springer, 2006. doi:10.1007/11821069_29.

18 Pierre Fraigniaud, Leszek Gąsieniec, Dariusz R. Kowalski, and Andrzej Pelc. Collective tree exploration. *Networks*, 48(3):166–177, 2006.

19 Yuya Higashikawa, Naoki Katoh, Stefan Langerman, and Shin-ichi Tanigawa. Online graph exploration algorithms for cycles and trees by multiple searchers. *J. Comb. Optim.*, 28(2):480–495, 2014.

20 Nicole Megow, Kurt Mehlhorn, and Pascal Schweitzer. Online graph exploration: New results on old and new algorithms. *Theor. Comput. Sci.*, 463:62–72, 2012.

21 Christian Ortolf and Christian Schindelhauer. A recursive approach to multi-robot exploration of trees. In *Proceedings of Structural Information and Communication Complexity - 21st International Colloquium, SIROCCO 2014*, volume 8576 of *Lecture Notes in Computer Science*, pages 343–354. Springer, 2014. doi:10.1007/978-3-319-09620-9_26.

22 Petrisor Panaite and Andrzej Pelc. Exploring unknown undirected graphs. *J. Algorithms*, 33(2):281–295, 1999.